UNH–IOL
NVMe Test Consortium

Test Plan for NVMe Conformance
Version 1.2.1
Technical Document

NOTICE: This is a living document. All contents are subject to change. Individual tests and/or test groups may be added/deleted/renumbered in forthcoming revisions. General feedback and comments are welcome through the NVMe Consortium at UNH–IOL.

Last Updated November 23, 2015
# Table of Contents

TABLE OF CONTENTS .................................................................................................................. 2
MODIFICATION RECORD ............................................................................................................. 4
ACKNOWLEDGMENTS .................................................................................................................. 6
INTRODUCTION ............................................................................................................................ 7
REFERENCES .................................................................................................................................. 9

**GROUP 1: ADMIN COMMAND SET** ......................................................................................... 10
  Test 1.1 – IDENTIFY COMMAND .............................................................................................. 11
  Test 1.2 – SET/GET FEATURES COMMANDS ......................................................................... 14
  Test 1.3 – GET LOG PAGE COMMAND .................................................................................... 16
  Test 1.4 – CREATE/DELETE I/O SUBMISSION AND COMPLETION QUEUE COMMANDS ....... 18
  Test 1.5 – ABORT COMMAND ................................................................................................. 19
  Test 1.6 – FORMAT NVM COMMAND .................................................................................... 20

**GROUP 2: NVM COMMAND SET** ............................................................................................. 21
  Test 2.1 – COMPARE COMMAND .......................................................................................... 22
  Test 2.2 – DATASET MANAGEMENT COMMAND ................................................................. 23
  Test 2.3 – READ COMMAND .................................................................................................. 25
  Test 2.4 – WRITE COMMAND ................................................................................................ 26
  Test 2.5 – WRITE UNCORRECTABLE COMMAND ................................................................. 27
  Test 2.6 – FLUSH COMMAND ................................................................................................ 28
  Test 2.7 – WRITE ZEREOES COMMAND ................................................................................. 29

**GROUP 3: NVM FEATURES** ..................................................................................................... 30
  Test 3.1 – METADATA HANDLING ........................................................................................ 31
  Test 3.2 – END–TO–END DATA PROTECTION ...................................................................... 33
  Test 3.3 – POWER MANAGEMENT ......................................................................................... 36

**GROUP 4: CONTROLLER REGISTERS** ...................................................................................... 39
  Test 4.1 – OFFSET 00H: CAP – MEMORY PAGE SIZE MAXIMUM (MPSMAX) ....................... 40
  Test 4.2 – OFFSET 00H: CAP – MEMORY PAGE SIZE MINIMUM (MPSMIN) ..................... 41
  Test 4.3 – OFFSET 00H: CAP – COMMAND SETS SUPPORTED (CSS). ............................... 42
  Test 4.4 – OFFSET 00H: CAP – DOORBELL STRIDE (DSTRD). ............................................. 43
  Test 4.5 – OFFSET 00H: CAP – TIMEOUT (TO). ................................................................. 44
  Test 4.6 – OFFSET 00H: CAP – ARBITRATION MECHANISM SUPPORTED (AMS).............. 45
  Test 4.7 – OFFSET 00H: CAP – CONTIGUOUS QUEUES REQUIRED (CQR) ......................... 46
  Test 4.8 – OFFSET 00H: CAP – MAXIMUM QUEUE ENTRIES SUPPORTED (MQES). .......... 47
  Test 4.9 – OFFSET 0CH–10H: INTMS – INTERRUPT MASK SET AND INTMC – INTERRUPT MASK CLEAR .............................................................. 48
  Test 4.10 – OFFSET 14H: CC – I/O COMPLETIONS QUEUE ENTRY SIZE (IOCQES) .......... 49
  Test 4.11 – OFFSET 14H: CC – I/O SUBMISSION QUEUE ENTRY SIZE (IOSQES) .............. 50
  Test 4.12 – OFFSET 14H: CC – SHUTDOWN NOTIFICATION (SHN) ...................................... 51
  Test 4.13 – OFFSET 14H: CC – ARBITRATION MECHANISM SELECTED (AMS) ................. 53
  Test 4.14 – OFFSET 14H: CC – I/O COMMAND SET SELECTED (CSS) ................................ 54
  Test 4.15 – OFFSET 14H: CC – ENABLE (EN) ..................................................................... 55
  Test 4.16 – OFFSET 1CH: CSTS – SHUTDOWN STATUS (SHST) ........................................... 56
  Test 4.17 – OFFSET 1CH: CSTS – CONTROLLER FATAL STATUS (CFS) ............................ 58

**GROUP 5: SYSTEM MEMORY STRUCTURE** .............................................................................. 59
  Test 5.1 – PAGE BASE ADDRESS AND OFFSET (PBAO) ...................................................... 60
  Test 5.2 – COMPLETION QUEUE ENTRY ............................................................................... 61
APPENDIX A: DEFAULT TEST SETUP .......................................................... 94
APPENDIX B: NOTES ON TEST PROCEDURES ............................................. 95
APPENDIX C: TEST TOOLS ...................................................................... 96
MODIFICATION RECORD

2012 June 20 (Version 0.1) Initial Release
  Raju Mishra:

2012 July 20 (Version 0.2)
  Neeraj Gill: Addition of Groups 4 and 5.

2012 September 27 (Version 0.3)
  David Woolf: Editorial Fixes and addition of Group 6 tests covering resets.

2013 February 18 (Version 0.31)
  David Woolf: Added note regarding 4k sector sizes to tests 2.2 and 2.3.

2013 April 30 (Version 0.32)
  David Woolf: Added note that tests 1.5 and 2.2 cannot be performed.

2013 May 9 (Version 0.34)
  David Woolf: Added test 2.6.

2013 May 21 (Version 1.0)
  David Woolf: Changed version of test suite to 1.0 after completion of NVMe Plugfest.

2013 October 22 (Version 1.1 DRAFT)
  David Woolf: Updates to tests 2.1 and 2.4.

2013 December 16 (Version 1.1 DRAFT)
  David Woolf: Updates to tests 1.1, 1.2, 1.3, 1.5, 2.1, 2.2, 2.3, 2.4, 2.5, 4.3, 4.4, 4.5, 4.6, 4.7, and 4.8. Most of the changes involve clarifying or removing observable results that could not be easily observed.

2013 December 19 (Version 1.1)
  David Woolf: Added Appendix C with information about potential test tools. Removed references to conformance testing of NVMe Hosts, as this is not currently a requirement for including NVMe Hosts on the NVMe Integrators List.

2014 May 27 (Version 1.1)
  David Woolf: Edited procedures to test 1.2 and 4.5 to make them easier to perform.

2014 June 5 (Version 1.1)
  David Woolf: Corrected test numbering in Group 5 and Group 6. Clarified that tests 3.1, 3.2, and 4.6 are optional. Corrected reference in test 5.3.

2014 July 7 (Version 1.1)
  David Woolf: Added test 6.4.

2014 July 10 (Version 1.1)
  David Woolf: Added test 2.7, modified procedure to test 2.1.

2014 July 14 (Version 1.1b)
  David Woolf: Change specification references to 1.1b revision of specification.

2014 August 14 (Version 1.1b)
  David Woolf: Added Group 7 tests for Reservations.

2014 September 18 (Version 1.1b)
David Woolf: Added notes Group 7 tests for Reservations to indicate that these tests are only applicable to device which support reservations.

2014 September 29 (Version 1.1b)
Mike Bogochow: Updated references and fixed errata.

2014 October 16 (Version 1.1b)
David Woolf: Clarification on Mandatory and Optional tests in Group 2.

2015 April 9 (Version 1.2)
David Woolf: Prepared document for 1.2 revisions.

2015 April 13 (Version 1.2)
David Woolf: Added tests 8.1 and 8.2.

2015 June 24 (Version 1.2.1)
Mike Bogochow: Fixed errata and updated references, procedures, and observable results for all tests in groups 1–6.

2015 November 23 (Version 1.2.1)
Mike Bogochow: Added Group 9 Tests.
ACKNOWLEDGMENTS

The UNH–IOL would like to acknowledge the efforts of the following individuals in the development of this test plan:

David Woolf          UNH InterOperability Laboratory
Raju Mishra          UNH InterOperability Laboratory
Neeraj Gill          UNH InterOperability Laboratory
Qian Liu             UNH InterOperability Laboratory
Kasra Dalvand        UNH InterOperability Laboratory
Benjamin Novak       UNH InterOperability Laboratory
Mike Bogochow        UNH InterOperability Laboratory
INTRODUCTION

The University of New Hampshire’s InterOperability Laboratory (IOL) is an institution designed to improve the interoperability of standards–based products by providing a neutral environment where a product can be tested against other implementations of a common standard, both in terms of interoperability and conformance. This particular suite of tests has been developed to help implementers evaluate the NVMe functionality of their products. This test suite is aimed at validating products in support of the work being directed by the NVMe Promoters Group.

These tests are designed to determine if a product conforms to specifications defined in the NVM Express Revision 1.2 specification, hereafter referred to as the “NVMe Specification”). Successful completion of these tests provide a reasonable level of confidence that the Device Under Test (DUT) will function properly in many NVMe environments.

The tests contained in this document are organized in order to simplify the identification of information related to a test, and to facilitate in the actual testing process. Tests are separated into groups, primarily in order to reduce setup time in the lab environment, however the different groups typically also tend to focus on specific aspects of device functionality. A two–number, dot–notated naming system is used to catalog the tests. This format allows for the addition of future tests in the appropriate groups without requiring the renumbering of the subsequent tests.

The test definitions themselves are intended to provide a high–level description of the motivation, resources, procedures, and methodologies specific to each test. Formally, each test description contains the following sections:

Purpose
The purpose is a brief statement outlining what the test attempts to achieve. The test is written at the functional level.

References
This section specifies all reference material external to the test suite, including the specific references for the test in question, and any other references that might be helpful in understanding the test methodology and/or test results. External sources are always referenced by a bracketed number (e.g., [1]) when mentioned in the test description. Any other references in the test description that are not indicated in this manner refer to elements within the test suite document itself (e.g., “Appendix 5.A.”, or “Table 5.1.1–1”).

Resource Requirements
The requirements section specifies the test hardware and/or software needed to perform the test. This is generally expressed in terms of minimum requirements, however in some cases specific equipment manufacturer/model information may be provided.

Last Modification
This specifies the date of the last modification to this test.

Discussion
The discussion covers the assumptions made in the design or implementation of the test, as well as known limitations. Other items specific to the test are covered here as well.

Test Setup
The setup section describes the initial configuration of the test environment. Small changes in the configuration should not be included here, and are generally covered in the test procedure section (next).

Procedure
The procedure section of the test description contains the systematic instructions for carrying out the test. It provides a cookbook approach to testing, and may be interspersed with observable results. These procedures should be the ideal test methodology, independent of specific tool limitations or restrictions.
Observable Results
This section lists the specific observable items that can be examined by the tester in order to verify that the DUT is operating properly. When multiple values for an observable are possible, this section provides a short discussion on how to interpret them. The determination of a pass or fail outcome for a particular test is generally based on the successful (or unsuccessful) detection of a specific observable.

Possible Problems
This section contains a description of known issues with the test procedure, which may affect test results in certain situations. It may also refer the reader to test suite appendices and/or other external sources that may provide more detail regarding these issues.
REFERENCES

The following documents are referenced in this text:

1. NVM Express Revision 1.2 Specification (November 3, 2014)
Group 1: Admin Command Set

Overview:

This section describes a method for performing conformance verification for NVMe products implementing the Admin Command Set.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 1.1 – Identify Command

**Purpose:** To verify that an NVMe Host can properly issue an Identify command, and receive the specified data structure back from the NVMe Controller.

**References:**
[1] NVMe Specification 5.11

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Identify command returns a data buffer that describes the NVM subsystem, the controller or the namespace(s). The data structure is 4096 bytes in size. The host indicates as a command parameter whether to return the controller or namespace specific data structure. For the namespace data structure, the definition of the structure is specific to the I/O command set selected for use.

The data structure returned, defined in Table 1, 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 command uses the PRP Entry 1, PRP Entry 2, and Command Dword 10 fields. All other Command specific fields are reserved. A completion queue entry is posted to the Admin Completion Queue when the Identify data structure has been posted to the memory buffer indicated in PRP Entry 1.

**Test Setup:** See Appendix A.

**Case 1: Identify Namespace Data Structure**

**Test Procedure:**
1. For each namespace, configure the NVMe Host to issue an Identify command specifying CNS value 00h to the controller in order to receive back an Identify Namespace data structure for the specified namespace.

**Observable Results:**
1. Verify that the requested data structure is posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
2. If the specified namespace ID is inactive, verify that the data structure returned by the controller is zero filled.

**Case 2: Identify Controller Data Structure**

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command specifying CNS value 01h to the controller in order to receive back an Identify Controller data structure.

**Observable Results:**
1. Verify that the requested data structure is posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
Table 1 – Identify – Data Structure Returned

<table>
<thead>
<tr>
<th>CNS Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>The Identify Namespace data structure is returned to the host for the namespace specified in the Namespace Identifier (CDW1.NSID) field if the namespace is attached to this controller. If the specified namespace is an inactive namespace ID, then the controller returns a zero filled data structure. If the controller supports Namespace Management and CDW1.NSID is set to FFFFFFFFh, the controller returns an Identify Namespace data structure that specifies capabilities that are common across namespaces.</td>
</tr>
<tr>
<td>01h</td>
<td>The Identify Controller data structure is returned to the host for this controller.</td>
</tr>
<tr>
<td>02h</td>
<td>A list of 1024 namespace IDs is returned containing active namespace IDs attached to this controller in increasing order that are greater than the value specified in the Namespace Identifier (CDW1.NSID) field of the command. The data structure returned is a Namespace List (refer to section 4.8). Controllers that support specification revision 1.1 or later shall support this capability.</td>
</tr>
<tr>
<td>03h–0Fh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Namespace Management

<table>
<thead>
<tr>
<th>CNS Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>10h</td>
<td>A list of up to 1024 namespace IDs is returned to the host containing namespaces that are present in the NVM subsystem with a namespace identifier greater than the value specified in the Namespace Identifier (CDW1.NSID) field. The namespaces may or may not be attached to controller(s).</td>
</tr>
<tr>
<td>11h</td>
<td>The Identify Namespace data structure is returned to the host for the namespace specified in the Namespace Identifier (CDW1.NSID) field. The namespace may or may not be attached to this controller. If the specified namespace is invalid then the controller shall fail the command with a status code of Invalid Namespace or Format.</td>
</tr>
<tr>
<td>12h</td>
<td>A Controller List of up to 2047 controller identifiers is returned containing a controller identifier greater than or equal to the value specified in the Controller Identifier (CDW10.CNTID) field. The list contains controller identifiers that are attached to the namespace specified in the Namespace Identifier (CDW1.NSID) field.</td>
</tr>
<tr>
<td>13h</td>
<td>A Controller List of up to 2047 controller identifiers is returned containing a controller identifier greater than or equal to the value specified in the Controller Identifier (CDW10.CNTID) field. The list contains controller identifiers in the NVM subsystem that may or may not be attached to the namespace(s).</td>
</tr>
<tr>
<td>14h–1Fh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Future Definition

<table>
<thead>
<tr>
<th>CNS Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h–FFh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Case 3: Namespace List

Test Procedure:
1. Configure the NVMe Host to issue an Identify command specifying CNS value 10h and CDW1.NSID value 0 to the controller in order to receive back a Namespace List containing active namespace IDs attached to the controller.
2. For each namespace ID returned in the Namespace List, configure the host to issue an Identify command specifying CNS value 00h to the controller in order to receive back an Identify Namespace data structure for the specified namespace.

Observable Results:
1. Verify that the requested Namespace List is posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
2. For each Identify command in step 2, verify that the returned data structure is not zero filled since each namespace ID in the Namespace List should be active.
Possible Problems: None.
Test 1.2 – Set/Get Features Commands

**Purpose:** To verify that an NVMe Host can properly issue a Get Features command, and receive the specified data structure back from the NVMe Controller. Also to verify that the NVMe Controller properly sets feature values after the NVMe Host issues a Set Features command.

**References:**  
[1] NVMe Specification 5.9, 5.12

**Resource Requirements:**  
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Get Features command retrieves the attributes of the Feature indicated. The Get Features command uses the PRP Entry 1, PRP Entry 2, and Command Dword 10 fields. All other command specific fields are reserved.

The Set Features command specifies the attributes of the Feature indicated. The Set Features command uses the PRP Entry 1, PRP Entry 2, Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. All other command specific fields are reserved.

The desired feature is specified in the Feature Identifier (FID) field of CDW10. Valid Feature Identifiers are described in Table 2 and Table 3.

<table>
<thead>
<tr>
<th>Feature Identifier</th>
<th>Optional/Mandatory</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>01h</td>
<td>Mandatory</td>
<td>Arbitration</td>
</tr>
<tr>
<td>02h</td>
<td>Mandatory</td>
<td>Power Management</td>
</tr>
<tr>
<td>03h</td>
<td>Optional</td>
<td>LBA Range Type</td>
</tr>
<tr>
<td>04h</td>
<td>Mandatory</td>
<td>Temperature Threshold</td>
</tr>
<tr>
<td>05h</td>
<td>Mandatory</td>
<td>Error Recovery</td>
</tr>
<tr>
<td>06h</td>
<td>Optional</td>
<td>Volatile Write Cache</td>
</tr>
<tr>
<td>07h</td>
<td>Mandatory</td>
<td>Number of Queues</td>
</tr>
<tr>
<td>08h</td>
<td>Mandatory</td>
<td>Interrupt Coalescing</td>
</tr>
<tr>
<td>09h</td>
<td>Mandatory</td>
<td>Interrupt Vector Configuration</td>
</tr>
<tr>
<td>0Ah</td>
<td>Mandatory</td>
<td>Write Atomicity Normal</td>
</tr>
<tr>
<td>0Bh</td>
<td>Mandatory</td>
<td>Asynchronous Event Configuration</td>
</tr>
<tr>
<td>0Ch</td>
<td>Optional</td>
<td>Autonomous Power State Transition</td>
</tr>
<tr>
<td>0Dh</td>
<td>Optional</td>
<td>Host Memory Buffer</td>
</tr>
<tr>
<td>0Eh – 77h</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>78h – 7Fh</td>
<td>N/A</td>
<td>Refer to Namespace Management</td>
</tr>
<tr>
<td>80h – BFh</td>
<td>N/A</td>
<td>Command Set Specific (Reserved)</td>
</tr>
<tr>
<td>C0h – FFh</td>
<td>N/A</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>
Table 3 – NVM Command Set Specific Feature Identifiers

<table>
<thead>
<tr>
<th>Feature Identifier</th>
<th>Optional/ Mandatory</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>80h</td>
<td>Optional</td>
<td>Software Progress Marker</td>
</tr>
<tr>
<td>81h</td>
<td>Optional</td>
<td>Host Identifier</td>
</tr>
<tr>
<td>82h</td>
<td>Optional</td>
<td>Reservation Notification Mask</td>
</tr>
<tr>
<td>83h</td>
<td>Optional</td>
<td>Reservation Persistence</td>
</tr>
<tr>
<td>84h – BFh</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

A completion queue entry is posted to the Admin Completion Queue when the controller has completed returning any attributes associated with the Feature. Depending on the Feature Identifier, Dword 0 of the completion queue entry may contain feature information.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. For each of the features described in Table 2 and Table 3:
   a. Configure the NVMe Host to issue a Get Features command indicating the specified FID value in CDW10.
   b. Configure the NVMe Host to issue a Set Features command indicating the specified FID value in CDW10 setting valid values for the feature.
   c. Configure the NVMe Host to issue a second Get Features command indicating the specified FID value in CDW10.

**Observable Results:**

1. Verify that a completion queue entry is posted to the Admin Completion Queue when the controller completes returning any attributes associated with the Feature Identifiers for Get Features commands.
2. Verify that the controller returns the Feature Information for each FID as specified in section 5.12 of the NVMe Specification.
3. Verify that the Feature Information returned in the second Get Features command matches the values which were set by the NVMe Host.

**Possible Problems:** None.
Test 1.3 – Get Log Page Command

Purpose: To verify that an NVMe Host can properly issue a Get Log Page command, and receive the specified data structure back from the NVMe Controller.

References:
[1] NVMe Specification 5.10

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: The Get Log Page command returns a data buffer that contains the log page requested. The Get Log Page command uses the PRP Entry 1, PRP Entry 2, and Command Dword 10 fields. All other command specific fields are reserved.

The desired log page is specified in the Log Page Identifier (LID) field of CDW10. Valid Log Identifiers are described in Table 4 and Table 5.

<table>
<thead>
<tr>
<th>Log Identifier</th>
<th>Optional/ Mandatory</th>
<th>Log Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>01h</td>
<td>Mandatory</td>
<td>Error Information</td>
</tr>
<tr>
<td>02h</td>
<td>Mandatory</td>
<td>SMART / Health Information</td>
</tr>
<tr>
<td>03h</td>
<td>Mandatory</td>
<td>Firmware Slot Information</td>
</tr>
<tr>
<td>04h</td>
<td>Optional</td>
<td>Changed Namespace List</td>
</tr>
<tr>
<td>05h</td>
<td>Optional</td>
<td>Command Effects Log</td>
</tr>
<tr>
<td>06h – 7Fh</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>80h – BFh</td>
<td>N/A</td>
<td>I/O Command Set Specific</td>
</tr>
<tr>
<td>C0h – FFh</td>
<td>N/A</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>

Table 4 – Log Page Identifiers

<table>
<thead>
<tr>
<th>Log Identifier</th>
<th>Optional/ Mandatory</th>
<th>Log Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>80h</td>
<td>Optional</td>
<td>Reservation Notification</td>
</tr>
<tr>
<td>81h – BFh</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 5 – NVM Command Set Specific Log Page Identifiers

A completion queue entry is posted to the Admin Completion Queue when the log has been posted to the memory buffer indicated in PRP Entry 1.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to issue a Get Log Page command specifying each of the log identifiers described in Table 4 and Table 5 to the NVMe Controller.

Observable Results:
1. Verify that a completion queue entry is posted to the Admin Completion Queue when the log has been posted to the memory buffer indicated in PRP Entry 1.
2. In each case verify that the information returned in the Log Page information is formatted according to the respective descriptions for the log pages in section 5.10 of the NVMe specification.
Possible Problems: None.
Test 1.4 – Create/Delete I/O Submission and Completion Queue Commands

**Purpose:** To verify that an NVMe Host can properly create and delete I/O Submission and Completion Queues.

**References:**

[1] NVMe Specification 5.3, 5.4, 5.5, 5.6

**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Create I/O Completion Queue command is used to create I/O Completion Queues while the Delete I/O Completion Queue command is used to delete I/O Submission Queues. Likewise, the Create I/O Submission Queue command is used to create I/O Submission Queues while the Delete I/O Submission Queue command is used to delete I/O Submission Queues. Host software shall ensure that any associated I/O Submission Queue is deleted prior to deleting a Completion Queue.

The Create I/O Submission Queue and Create I/O Completion Queue commands use the PRP Entry 1, Command Dword 10, and Command Dword 11 fields. The Delete I/O Submission Queue and Delete I/O Completion Queue commands use the Command Dword 10. All other command specific fields are reserved.

A completion queue entry is posted to the Admin Completion Queue when the specified queue has been created or deleted.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to issue a Create I/O Completion Queue command.
2. Configure the NVMe Host to issue a Create I/O Submission Queue command.
3. Configure the NVMe Host to issue 10 NVMe Read Commands assigned to the queues created in steps 1 and 2.
4. Configure the NVMe Host to issue a Delete I/O Submission Queue command specifying the Submission Queue ID of the queue created in step 2.
5. Configure the NVMe Host to issue a Delete I/O Completion Queue command specifying the Completion Queue ID of the queue created in step 1.

**Observable Results:**

1. Verify that after the completion of each Admin command, the controller posts a completion queue entry to the Admin Completion Queue.
2. Verify that the queues are properly created and deleted.
3. Verify that the NVMe Read command statuses are properly updated to the respective I/O Submission and Completion Queues.

**Possible Problems:** None.
Test 1.5 – Abort Command

**Purpose:** To determine the conditions under which an NVMe system can successfully abort a given command.

**References:**

[1] NVMe Specification 5.1

**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Abort command is used to abort a specific command previously issued to the Admin Submission Queue or an I/O Submission Queue. Host software may have multiple Abort commands outstanding, subject to the constraints of the Abort Command Limit indicated in the Identify Controller data structure. An abort is a best effort command; the command to abort may have already completed, currently be in execution, or may be deeply queued. It is implementation specific if/when a controller chooses to complete the command when the command to abort is not found. Whether the command was successfully aborted or not is determined by examining bit 0 of command Dword 0 of the completion queue entry for the Abort command. If the command was successfully aborted, then bit 0 of Dword 0 is cleared to ‘0’. If the command was not aborted, then bit 0 of Dword 0 is set to ‘1’. The completion queue entry for the Abort command is only posted to the Admin Completion Queue after the completion queue entry for the command specified in the Abort command has been posted to its corresponding Admin or I/O Completion Queue.

The Abort command uses the Command Dword 10 field. All other command specific fields are reserved.

Since the Abort Command is a best effort command, this test is designed to determine under what conditions a command can successfully be aborted, and is therefore considered an informative test.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to issue a Create I/O Completion Queue command.
2. Configure the NVMe Host to issue a Create I/O Submission Queue command.
3. Configure the NVMe Host to issue 10 NVMe Read commands assigned to the queues created in steps 1 and 2.
4. Configure the NVMe Host to attempt to issue an Abort command for one of the issued Read Commands.
5. After the commands issued in step 3 have been completed, configure the NVMe Host to delete the Submission and Completion Queues created in steps 1 and 2.

**Observable Results:**

1. Verify that after the NVMe Controller posts a completion queue entry to the Admin Completion Queue for the Abort command, that a completion queue entry has been posted to the I/O Completion Queue for the Read command specified in the Abort command.
2. Determine whether the requested command was successfully aborted by examining bit 0 of Dword 0 of the completion queue entry for the Abort command. If the command was successfully aborted, verify that the status of the aborted command is Command Abort Requested (07h).

**Possible Problems:** None.
Test 1.6 – Format NVM Command

**Purpose:** To verify that an NVMe system can properly execute the Format NVM command.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Format NVM command is used to low level format the NVM media. This is used when the host wants to change the LBA data size and/or metadata size. A low level format may destroy all data and metadata associated with all namespaces or only the specific namespace associated with the command. After the Format NVM command successfully completes, the controller shall not return any user data that was previously contained in an affected namespace.

The settings specified in the Format NVM command are reported as part of the Identify Namespace data structure. If the controller supports multiple namespaces, then the host may specify the value of FFFFFFFFh for the namespace in order to apply the format operation to all namespaces accessible by the controller regardless of the value of the Format NVM Attribute field in the Identify Controller data structure.

The Format NVM command uses the Command Dword 10 field. All other command specific fields are reserved.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue an NVM Write command to write a known data pattern.
2. Configure the NVMe Host to issue an NVM Read command specifying the same LBA range which was written in step 1.
3. Configure the NVMe Host to issue a Format NVM Command with no secure erase operation requested and FFFFFFFFh for the namespace ID.
4. Configure the NVMe Host to issue an NVM Read command specifying the same LBA range which was written in step 1.

**Observable Results:**
1. Verify that the same data pattern which was written in step 1 is read in step 2.
2. Verify that a completion queue entry was posted to the Admin Completion Queue for the Format NVM command.
3. Verify that the data pattern written in step 1 is not returned by the controller in step 4.

**Possible Problems:** None.
Group 2: NVM Command Set

Overview:

This section describes a method for performing conformance verification for NVMe products implementing the NVM Command Set.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 2.1 – Compare Command

Purpose: To verify that an NVMe system can properly execute the Compare command.

References:
[1] NVMe Specification 6.6

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: The Compare command reads the logical blocks specified by the command from the medium and compares the data read to a comparison data buffer transferred as part of the command. If the data read from the controller and the comparison data buffer are equivalent with no miscompares, then the command completes successfully. If there is any miscompare, the command completes with an error of Compare Failure. If metadata is provided, then a comparison is also performed for the metadata.

The command uses Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 14, and Command Dword 15. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used. All other command specific fields are reserved.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to issue an Identify command to the NVMe Controller specifying CNS value 01h in order to retrieve the Identify Controller data structure.
2. Configure the NVMe Host to issue a Write command in order to write the same Controller data structure received in response to the Identify Command to the LBA 0000.
3. Configure the NVMe Host to issue a Compare command between the PRP Address used for the Identify command in step1 and LBA 0000.

Observable Results:
1. Verify that the Compare command completes successfully.

Possible Problems: This is an optional test, depending on device feature support.
Test 2.2 – Dataset Management Command

**Purpose:** To verify that an NVMe system can properly execute the Dataset Management command.

**References:**


**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Dataset Management command is used by the host to indicate attributes for ranges of logical blocks. This includes attributes like frequency that data is read or written, access size, and other information that may be used to optimize performance and reliability. This command is advisory; a compliant controller may choose to take no action based on information provided.

The command uses Command Dword 10 and Command Dword 11 fields. If the command uses PRPs for the data transfer, then the PRP Entry 1 and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, the SGL Entry 1 field is used. All other command specific fields are reserved.

The data that the Dataset Management command provides is a list of ranges with context attributes. Each range consists of a starting LBA, a length of logical blocks that the range consists of and the context attributes to be applied to that range. The definition of the Dataset Management command Range field is specified in Table 6. The maximum case of 256 ranges is shown.

<table>
<thead>
<tr>
<th>Range</th>
<th>Byte</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03:00</td>
<td>Context Attributes</td>
</tr>
<tr>
<td></td>
<td>07:04</td>
<td>Length in logical blocks</td>
</tr>
<tr>
<td></td>
<td>15:08</td>
<td>Starting LBA</td>
</tr>
<tr>
<td>1</td>
<td>19:16</td>
<td>Context Attributes</td>
</tr>
<tr>
<td></td>
<td>23:20</td>
<td>Length in logical blocks</td>
</tr>
<tr>
<td></td>
<td>31:24</td>
<td>Starting LBA</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>255</td>
<td>4083:4080</td>
<td>Context Attributes</td>
</tr>
<tr>
<td></td>
<td>4087:4084</td>
<td>Length in logical blocks</td>
</tr>
<tr>
<td></td>
<td>4095:4088</td>
<td>Starting LBA</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Case 1: Basic Operation**

**Test Procedure:**

1. Configure the NVMe Host to issue the Dataset Management command that indicates attributes for ranges of logical blocks.

**Observable Results:**

1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.
Case 2: Deallocate

The Dataset Management command may also be used to deallocate ranges of logical blocks by setting the Attribute – Deallocate (AD) field of Command Dword 11 to ‘1’. When the controller receives a Dataset Management command with the AD field set to ‘1’, it may deallocate all provided ranges. If a read occurs to a deallocated range, the controller shall return all zeros, all ones, or the last data written to the associated LBA. If the deallocated or unwritten logical block error is enabled and a read occurs to a deallocated range, then the read shall fail with the Unwritten or Deallocated Logical Block status code.

An LBA that has been deallocated using the Dataset Management command is no longer deallocated when the LBA is written. Read operations do not affect the deallocation status of an LBA. The value read from a deallocated LBA shall be deterministic; specifically, the value returned by subsequent reads of that LBA shall be the same until a write occurs to that LBA.

Test Procedure:
1. Configure the NVMe Host to issue a Write command to write a known data pattern to the NVM.
2. Configure the NVMe Host to issue a Dataset Management command with the Attribute – Deallocate (AD) field set to ‘1’ and specifying the same LBA range written to in step 1.
3. Configure the NVMe Host to issue a Read command to read the same LBA range written to in step 1.
4. Configure the NVMe Host to issue another Read command to read the same LBA range written to in step 1.
5. Configure the NVMe Host to issue a Write command to write a known data pattern to the NVM specifying the same LBA range previously deallocated.
6. Configure the NVMe Host to issue a Read command to read the same LBA range written to in step 5.

Observable Results:
1. Verify that the data returned by the controller from the Read command in step 3 is either all zeros, all ones, or the data written to the associated LBA in step 1.
2. Verify that the data returned by the controller from the Read command in step 4 is identical to the data returned by the controller from the Read command in step 3.
3. Verify that the data returned by the controller from the Read command in step 6 matches the data written in step 5.

Possible Problems: This is an optional test, depending on device feature support.
Test 2.3 – Read Command

**Purpose:** To verify that an NVMe system can properly execute the Read command.

**References:**
- [1] NVMe Specification 6.9

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Read command reads data and metadata, if applicable, from the NVM controller for the LBAs indicated. The command may specify protection information to be checked as part of the read operation.

The command uses Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used. If supported, this test may be performed using 4k sector sizes.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue a Write command to write a known data pattern to one LBA on the NVMe Device.
2. Configure the NVMe Host to read from the same LBA on the NVMe Device which was written in step 1.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.
2. Verify that the known data pattern was read correctly from the NVMe Device exactly as it was written.

**Possible Problems:** None.
Test 2.4 – Write Command

**Purpose:** To verify that an NVMe system can properly execute the Write command.

**References:**

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Write command writes data and metadata, if applicable, to the NVM controller for the logical blocks indicated. The host may also specify protection information to include as part of the operation.

The command uses Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 13, Command Dword 14, and Command Dword 15 fields. If the command uses PRPs for the data transfer, then the Metadata Pointer, PRP Entry 1, and PRP Entry 2 fields are used. If the command uses SGLs for the data transfer, then the Metadata SGL Segment Pointer and SGL Entry 1 fields are used. If supported, this test may be performed using 4k sector sizes.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue a Write command to write a known data pattern to one LBA on the NVMe Device.
2. Configure the NVMe Host to read from the same LBA on the NVMe Device which was written in step 1.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.
2. Verify that the known data pattern was read correctly from the NVMe Device exactly as it was written.

**Possible Problems:** None.
Test 2.5 – Write Uncorrectable Command

**Purpose:** To verify that an NVMe system can properly execute the Write Uncorrectable command.

**References:**

[1] NVMe Specification 6.15

**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Write Uncorrectable command is used to mark a range of logical blocks as invalid. When the specified logical blocks(s) are read after this operation, a failure is returned with Unrecovered Read Error status. To clear the invalid logical block status, a write operation is performed for those logical blocks.

The fields used are Command Dword 10, Command Dword 11, and Command Dword 12 fields. All other command specific fields are reserved.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to issue a Write Uncorrectable command to a particular LBA on the NVMe Device.
2. Configure the NVMe Host to issue a Read command for the LBA on which the Write Uncorrectable command was performed.

**Observable Results:**

1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.
2. Verify that the Read command attempted to the LBA returns Unrecovered Read Error status.

**Possible Problems:** This is an optional test, depending on device feature support.
Test 2.6 – Flush Command

Purpose: To verify that an NVMe system can properly execute the Flush command.

References:
[1] NVMe Specification 6.8

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: May 9, 2013

Discussion: The Flush command shall commit data and metadata associated with the specified namespace(s) to non-volatile media. The flush applies to all commands completed prior to the submission of the Flush command. The controller may also flush additional data and/or metadata from any namespace.

All command specific fields are reserved.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to issue a Flush command to the NVMe Device.

Observable Results:
1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.

Possible Problems: None.
Test 2.7 – Write Zeroes Command

**Purpose:** To verify that an NVMe system can properly execute the Write Zeroes command.

**References:**

[1] NVMe Specification 6.16

**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Write Zeroes command is used to set a range of logical blocks to zero. After successful completion of this command, the value returned by subsequent reads of logical blocks in this range shall be zeroes until a write occurs to this LBA range. The metadata for this command shall be all zeroes and the protection information is updated based on CDW12.PRINFO.

The fields used are Command Dword 10, Command Dword 11, Command Dword 12, Command Dword 14, and Command Dword 15 fields.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to issue a Write command to a particular LBA on the NVMe Device. The written data should be either all 1’s, or an alternating pattern of 1’s and 0’s.
2. Configure the NVMe Host to issue a Write Zeroes command to the same LBA on the NVMe Device which was previously written to.
3. Configure the NVMe Host to issue a Read command to the same LBA which the Write Zeroes command was performed.

**Observable Results:**

1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.
2. Verify that the Read command attempted to the LBA after the Write Zeroes operation was performed returns data of zeroes only.

**Possible Problems:** This is an optional test, depending on device feature support.
Group 3: NVM Features

Overview:

This section describes a method for performing conformance verification for NVMe products implementing NVM Features.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 3.1 – Metadata Handling

Purpose: To verify that an NVMe system properly handles metadata.

References:

[1] NVMe Specification 8.2

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: Metadata is additional data allocated on a per logical block basis. There is no requirement for how the host makes use of the metadata area. One of the most common usages for metadata is to convey end-to-end protection information. The metadata may be transferred by the controller to or from the host in one of two ways. The mechanism used is selected when the namespace is formatted.

One of the transfer mechanisms shall be selected for each namespace when it is formatted; transferring a portion of metadata with one mechanism and a portion with the other mechanism is not supported.

Test Setup: See Appendix A.

Case 1: Extended LBA

The first mechanism for transferring the metadata is as a contiguous part of the logical block that it is associated with. The metadata is transferred at the end of the associated logical block, forming an extended logical block. This mechanism is illustrated in Figure 1. In this case, both the logical block data and logical block metadata are pointed to by the PRP1 and PRP2 pointers (or SGL Entry 1 if SGLs are used).

Figure 1 – Contiguous Metadata with LBA Data, Forming Extended LBA

Test Procedure:
1. Instruct the host to issue a Write command to an LBA in a namespace which has been formatted to allow metadata via extended LBAs with known data patterns for both the data and metadata.
2. Instruct the host to issue a Read command to the LBA which was previously written to.

Observable Results:
1. Verify that the data and metadata returned by the controller exactly match the data and metadata patterns which were written.

Case 2: Separate Buffer

The second mechanism for transferring the metadata is as a separate buffer of data. This mechanism is illustrated in Figure 2. In this case, the metadata is pointed to with the Metadata Pointer, while the logical block data is pointed to by the Data Pointer. When a command uses PRPs for the metadata in the command, the metadata is required to be physically contiguous. When a command uses SGLs for the metadata in the command, the metadata is not required to be physically contiguous.
Figure 2 – Metadata Transferred as Separate Buffer

Test Procedure:
1. Instruct the host to issue a Write command to an LBA in a namespace which has been formatted to allow metadata via a separate buffer with known data patterns for both the data and metadata.
2. Instruct the host to issue a Read command to the LBA which was previously written to.

Observable Results:
1. Verify that the data and metadata returned by the controller exactly match the data and metadata patterns which were written.

Possible Problems: This is an optional test; the DUT may or may not support the necessary test requirements.
Test 3.2 – End–to–end Data Protection

**Purpose:** To verify that an NVMe system can properly handle end–to–end data protection.

**References:**
[1] NVMe Specification 8.3

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** To provide robust data protection from the application to the NVM media and back to the application itself, end–to–end data protection may be used. If this optional mechanism is enabled, then additional protection information (e.g. CRC) is added to the LBA that may be evaluated by the controller and/or host software to determine the integrity of the logical block.

This additional protection information, if present, is either the first eight bytes of metadata or the last eight bytes of metadata, based on the format of the namespace. For metadata formats with more than eight bytes, if the protection information is contained within the first eight bytes of metadata, then the CRC does not cover any metadata bytes. For metadata formats with more than eight bytes, if the protection information is contained within the last eight bytes of metadata, then the CRC covers all metadata bytes up to but excluding these last eight bytes.

**Test Setup:** See Appendix A.

**Case 1: Write Command Processing**

Figure 3 illustrates the protection information processing that may occur as a side effect of a Write command. If the namespace was formatted with protection information and the PRACT bit is cleared to ‘0’, then logical block data and metadata containing protection information are transferred from the host to NVM. As the logical block and metadata pass through the controller, the protection information is checked. If a protection information check error is detected, the command completes with the status code of the error detected (i.e., End–to–end Guard Check, End–to–end Application Tag Check or End–to end Reference Tag Check). If the namespace was formatted with protection information and the PRACT bit is set to ‘1’, then logical block data is transferred from the host to the controller. The controller inserts protection information and the logical block data and metadata containing protection information are written to NVM.

**Test Procedure:**
1. Configure the NVMe Host to issue a Write command with the PRACT bit cleared to ‘0’ to an LBA in a namespace which has been formatted with protection information with properly formatted protection information with a known data pattern.
2. Configure the NVMe Host to issue a Write command with the PRACT bit cleared to ‘0’ to an LBA in a namespace which has been formatted with protection information with improperly formatted protection information.
3. Configure the NVMe Host to issue a Write command with the PRACT bit set to ‘1’ to an LBA in a namespace which has been formatted with protection information with a known data pattern.

**Observable Results:**
1. For steps 1 and 3, verify that the command completes successfully.
2. For step 2, verify that the command fails with appropriate status code (i.e., End–to–end Guard Check, End–to–end Application Tag Check or End–to end Reference Tag Check).
Figure 3 – Write Command Protection Information Processing

(a) No Protection Information

(b) Protection Information with PRACT bit cleared to ‘0’ (i.e., pass)

(c) Protection Information with PRACT bit set to ‘1’ (i.e., insert)

Case 2: Read Command Processing

Figure 4 illustrates the protection information processing that may occur as a side effect of a Read command. The processing parallels Write command processing with the exception that logical block data flows in the opposite direction. When the PRACT bit is cleared to ‘0’ and the namespace was formatted with protection information, logical block data and metadata are transferred from NVM to the host and checked by the controller. When the PRACT bit is set to ‘1’ and the namespace was formatted with protection information, logical block data and metadata are transferred from the NVM to the controller. The controller checks the protection information and then removes it from the metadata before passing the LBA to the host. If the namespace format contains metadata beyond the protection information, then the protection information is not stripped regardless of the state of the PRACT bit (i.e., the metadata field remains the same size in the host as that in NVM).

Test Procedure:

1. Configure the NVMe Host to issue a Read command with the PRACT bit cleared to ‘0’ to the LBA which was written to in Case 1 step 1.
2. Configure the NVMe Host to issue a Read command with the PRACT bit cleared to ‘0’ to the LBA which was written to in Case 1 step 3.
3. Configure the NVMe Host to issue a Write command with the PRACT bit set to ‘1’ to the LBA which was written to in Case 1 step 1.
Figure 4 – Read Command Protection Information Processing

(a) No Protection Information

(b) Protection Information with PRACT bit cleared to ‘0’ (i.e., pass)

(c) Protection Information with PRACT bit set to ‘1’ (i.e., strip)

**Observable Results:**

1. For step 1, verify that data returned by the controller exactly matches the data and metadata which was written.
2. For step 2, verify that the data returned by the controller exactly matches the data which was written and contains additional metadata containing properly formatted and correct protection information.
3. For step 3, verify that the data returned by the controller contains no protection information.

**Possible Problems:** This is an optional test; the DUT may or may not support the necessary test requirements.
Test 3.3 – Power Management

**Purpose:** To verify that an NVMe system can properly handle power management.

**References:**

[1] NVMe Specification 8.4

**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The power management capability allows the host to manage NVM subsystem power statically or dynamically. Static power management consists of the host determining the maximum power that may be allocated to an NVM subsystem and setting the NVM Express power state to one that consumes this amount of power or less. Dynamic power management is illustrated in Figure 5 and consists of the host modifying the NVM Express power state to best satisfy changing power and performance objectives. This power management mechanism is meant to complement and not replace autonomous power management performed by a controller.

![Figure 5 – Dynamic Power Management](image)

The number of power states implemented by a controller is returned in the Number of Power States Supported (NPSS) field in the Identify Controller data structure. A controller shall support at least one power state and may optionally support up to a total of 32 power states. Power states are contiguously numbered starting with zero such that each subsequent power state consumes less than or equal to the maximum power consumed in the previous state. Thus, power state zero indicates the maximum power that the NVM subsystem is capable of consuming.

Associated with each power state is a Power State Descriptor in the Identify Controller data structure. The descriptors for all implemented power states may be viewed as forming a table as shown in Table 7 which contains sample power state values for a controller with seven implemented power states.

**Test Setup:** See Appendix A.

**Case 1: Relative Write Latency**

Relative Write Latency (RWL): This field indicates the relative write latency associated with this power state. The value in this field shall be less than the number of supported power states (e.g., if the controller supports 16 power states, then valid values are 0 through 15). A lower value means lower write latency.
Table 7 – Example Power State Descriptor Table

<table>
<thead>
<tr>
<th>Power State</th>
<th>Maximum Power (MP) (W)</th>
<th>Entry Latency (ENTLAT) (µs)</th>
<th>Exit Latency (EXLAT) (µs)</th>
<th>Relative Read Throughput (RRT)</th>
<th>Relative Read Latency (RRL)</th>
<th>Relative Write Throughput (RWT)</th>
<th>Relative Write Latency (RWL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>5</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>50</td>
<td>50</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>20</td>
<td>5000</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Test Procedure:
1. Configure the NVMe Host to issue an Identify command specifying CNS value 01h to the controller in order to receive back an Identify Controller data structure.
2. For each power state supported by the controller as indicated by the NPSS field, read the RWL field of the associated Power State Descriptor.

Observable Results:
1. Verify that the relative write latency values are all less than the number of supported power states.

Case 2: Relative Write Throughput
Relative Write Throughput (RWT): This field indicates the relative write throughput associated with this power state. The value in this field shall be less than the number of supported power states (e.g., if the controller supports 16 power states, then valid values are 0 through 15). A lower value means higher write throughput.

The amount of data an application can write to stable storage on the server over a period of time is a measurement of the write throughput of a distributed file system. Write throughput is therefore an important aspect of performance.

Test Procedure:
1. Configure the NVMe Host to issue an Identify command specifying CNS value 01h to the controller in order to receive back an Identify Controller data structure.
2. For each power state supported by the controller as indicated by the NPSS field, read the RWL field of the associated Power State Descriptor.

Observable Results:
1. Verify that the relative write throughput values are all less than the number of supported power states.

Case 3: Relative Read Latency
Relative Read Latency (RRL): This field indicates the relative read latency associated with this power state. The value in this field shall be less than the number of supported power states (e.g., if the controller supports 16 power states, then valid values are 0 through 15). A lower value means lower read latency.

Test Procedure:
1. Configure the NVMe Host to issue an Identify command specifying CNS value 01h to the controller in order to receive back an Identify Controller data structure.
2. For each power state supported by the controller as indicated by the NPSS field, read the RWL field of the associated Power State Descriptor.

Observable Results:
1. Verify that the relative read latency values are all less than the number of supported power states.
Case 4: Relative Read Throughput
Relative Read Throughput (RRT): This field indicates the relative read throughput associated with this power state. The value in this field shall be less than the number of supported power states (e.g., if the controller supports 16 power states, then valid values are 0 through 15). A lower value means higher read throughput.

Test Procedure:
1. Configure the NVMe Host to issue an Identify command specifying CNS value 01h to the controller in order to receive back an Identify Controller data structure.
2. For each power state supported by the controller as indicated by the NPSS field, read the RWL field of the associated Power State Descriptor.

Observable Results:
1. Verify that the relative read throughput values are all less than the number of supported power states.

Case 5: Power Management Feature
The host may dynamically modify the power state using the Set Features command and determine the current power state using the Get Features command. The host may directly transition between any two supported power states.

The Entry Latency (ENTLAT) field in the power management descriptor indicates the maximum amount of time in microseconds that it takes to enter that power state and the Exit Latency (EXLAT) field indicates the maximum amount of time in microseconds that it takes to exit that state. The maximum amount of time to transition between any two power states is equal to the sum of the old state’s exit latency and the new state’s entry latency. The host is not required to wait for a previously submitted power state transition to complete before initiating a new transition.

Test Procedure:
1. Configure the NVMe Host to issue a Set Features command for the Power Management Feature for each of the supported Power State values supported by the NVMe Controller.
2. After each Set Features command completes, configure the NVMe Host to issue a Get Features command for the Power Management Feature.

Observable Results:
1. Verify that after the completion of the command, the controller posts a completion queue entry to the Admin Completion Queue indicating the status for the command.
2. For each Get Features command, verify that the controller successfully transitioned to the selected power state.

Possible Problems: None.
Group 4: Controller Registers

Overview:

This section describes a method for performing conformance verification for NVMe products implementing the NVMe Controller Registers.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 4.1 – Offset 00h: CAP – Memory Page Size Maximum (MPSMAX)

**Purpose:** To validate the MPSMAX field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field indicates the maximum host memory page size that the controller supports. The maximum memory page size is \(2^{(12 + MPSMAX)}\) (i.e. \(2^{12+MPSMAX}\)). Therefore, the maximum memory page size which a controller may support is 128MB. The host shall not configure a memory page size in CC.MPS that is larger than this value.

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to read the CAP.MPSMAX field (bits 55:52) of the NVMe Controller.

**Observable Results:**

1. Verify that the value of CAP.MPSMAX is greater than or equal to the value of CAP.MPSMIN.

**Possible Problems:** None
Test 4.2 – Offset 00h: CAP – Memory Page Size Minimum (MPSMIN)

Purpose: To validate the MPSMIN field of the Controller Capabilities (CAP) register.

References:
[1] NVMe Specification 3.1.1

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: This field indicates the minimum host memory page size that the controller supports. The minimum memory page size is \(2^{(12 + \text{MPSMIN})}\) (i.e. \(2^{12+\text{MPSMIN}}\)). Therefore, the minimum memory page size which a controller may support is 4KB. The host shall not configure a memory page size in CC.MPS that is smaller than this value.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to read the CAP.MPSMIN field (bits 51:48) of the NVMe Controller.

Observable Results:
1. Verify that the value of CAP.MPSMAX is greater than or equal to the value of CAP.MPSMIN.

Possible Problems: None.
Test 4.3 – Offset 00h: CAP – Command Sets Supported (CSS)

**Purpose:** To validate the Command Sets Supported (CSS) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** December 16, 2013

**Discussion:** This field indicates the I/O Command Set(s) that the controller supports. A minimum of one command set shall be supported. The field is bit significant (bit definitions can be found in Table 8). If a bit is set to ‘1’, then the corresponding I/O Command Set is supported. If a bit is cleared to ‘0’, then the corresponding I/O Command Set is not supported.

### Table 8 – Command Sets Supported Bit Definitions

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>NVM command set</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.CSS field (bits 37:44) of the NVMe Controller.

**Observable Results:**
1. Verify that NVM Command Set is supported by the controller (i.e. bit 37 is set to ‘1’).

**Possible Problems:** None.
Test 4.4 – Offset 00h: CAP – Doorbell Stride (DSTRD)

**Purpose:** To validate the Doorbell Stride (DSTRD) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** Each Submission Queue and Completion Queue Doorbell register is 32–bits in size. This register indicates the stride between doorbell registers. The stride is specified as \(2^{(2 + \text{DSTRD})}\) (i.e. \(2^{2+\text{DSTRD}}\)) in bytes. A value of 0h indicates a stride of 4 bytes, where the doorbell registers are packed without reserved space between each register.

Since there is no means to validate the value stored in this register field, this test is designed purely to determine the value the NVMe Controller returns when the register is read, and is therefore considered an informative test.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.DSTRD field (bits 35:32) of the NVMe Controller.

**Observable Results:**
1. Report the value of the CAP.DSTRD field.

**Possible Problems:** None
Test 4.5 – Offset 00h: CAP – Timeout (TO)

**Purpose:** To validate the Timeout (TO) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:**
This is the worst case time that host software shall wait for CSTS.RDY to transition from:

a) ‘0’ to ‘1’ after CC.EN transitions from ‘0’ to ‘1’; or
b) ‘1’ to ‘0’ after CC.EN transitions from ‘1’ to ‘0’

This worst case time may be experienced after events such as an abrupt shutdown or activation of a new firmware image; typical times are expected to be much shorter. This field is in 500 millisecond units.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.TO field (bits 31:24) of the NVMe Controller.
2. Configure the NVMe Host to clear the CC.EN field of the NVMe Controller to ‘0’ to initiate a controller reset.
3. Configure the NVMe Host to query the CSTS.RDY field every 100 milliseconds until it transitions from ‘1’ to ‘0’.
4. Configure the NVMe Host to set the CC.EN field of the NVMe Controller to ‘1’ to re-enable the controller.
5. Configure the NVMe Host to query the CSTS.RDY field every 100 milliseconds until it transitions from ‘0’ to ‘1’.

**Observable Results:**
1. Determine the elapsed time it takes for the CSTS.RDY value to transition after toggling the value of the CC.EN field. Verify that this value less than or equal to the timeout value calculated from the CAP.TO field read from the NVMe Controller.

**Possible Problems:** None
Test 4.6 – Offset 00h: CAP – Arbitration Mechanism Supported (AMS)

**Purpose:** To validate the Arbitration Mechanism Supported (AMS) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field is bit significant (bit definitions can be found in Table 9) and indicates the optional arbitration mechanisms supported by the controller. If a bit is set to ‘1’, then the corresponding arbitration mechanism is supported by the controller. Refer to section 4.11 of the NVMe Specification for arbitration details.

**Table 9 – CAP.AMS Bit Definitions**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Weighted Round Robin with Urgent Priority Class</td>
</tr>
<tr>
<td>18</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>

The round robin arbitration mechanism is not listed since all controller shall support this arbitration mechanism.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.AMS field (bits 18:17) of the NVMe Controller.
2. For each arbitration mechanism supported by the NVMe Controller as indicated by the CAP.AMS field, configure the NVMe Host to set the CC.AMS field to the corresponding value.

**Observable Results:**
1. For each supported arbitration mechanism, verify that the NVMe Host is able to successfully set the CC.AMS field to the corresponding value.

**Possible Problems:** The weighted Round Robin with Urgent and Vendor Specific arbitration mechanisms are optional to support. Therefore, it is valid for both bits of the CAP.AMS field to be cleared to ‘0’.
Test 4.7 – Offset 00h: CAP – Contiguous Queues Required (CQR)

**Purpose:** To validate the Contiguous Queues Required (CQR) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field is set to ‘1’ if the controller requires that I/O Submission Queues and I/O Completion Queues are required to be physically contiguous. This field is cleared to ‘0’ if the controller supports I/O Submission Queues and I/O Completion Queues that are not physically contiguous. If this field is set to ‘1’, then the Physically Contiguous bit (CDW11_PC) in the Create I/O Submission Queue and Create I/O Completion Queue commands shall be set to ‘1’.

Since there is no means to validate the value stored in this register field, this test is designed purely to determine the value the NVMe Controller returns when the register is read, and is therefore considered an informative test.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.CQR field (bit 16) of the NVMe Controller.

**Observable Results:**
1. Report the value of the CAP.CQR field.

**Possible Problems:** None
Test 4.8 – Offset 00h: CAP – Maximum Queue Entries Supported (MQES)

**Purpose:** To validate the Maximum Queue Entries Supported (MQES) field of the Controller Capabilities (CAP) register.

**References:**
[1] NVMe Specification 3.1.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field indicates the maximum individual queue size that the controller supports. This value applies to each of the I/O Submission Queues and I/O Completion Queues that host software may create. This is a 0’s based value. The minimum value is 1h, indicating two entries.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.MQES field (bits 15:00) of the NVMe Controller.

**Observable Results:**
1. Verify that the value of the CAP.MQES field is 1h or greater indicating 2 or more entries.

**Possible Problems:** None.
Test 4.9 – Offset 0Ch–10h: INTMS – Interrupt Mask Set and INTMC – Interrupt Mask Clear

**Purpose:** To validate feature of Interrupt Mask Set (INTMS) register.

**References:**
[1] NVMe Specification 3.1.3/3.1.4

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The INTMS register is used to mask interrupts when using pin–based interrupts, single message MSI, or multiple message MSI. When using MSI–X, the interrupt mask table defined as part of MSI–X should be used to mask interrupts. Host software shall not access this register when configured for MSI–X; any accesses when configured for MSI–X is undefined.

The INTMC register is used to unmask interrupts when using pin–based interrupts, single message MSI, or multiple message MSI. When using MSI–X, then interrupt mask table defined as part of MSI–X should be used to unmask interrupts. Host software shall not access this register when configure for MSI–X; any accesses when configured for MSI–X is undefined.

The INTMS and INTMC registers both have a single field each: the Interrupt Vector Mask Set (IVMS) and Interrupt Vector Mask Clean (IVMC) fields respectively. Both fields are bit significant. If a ‘1’ is written to a bit in the IVMS field, then the corresponding interrupt vector is masked from generating an interrupt or reporting a pending interrupt in the MSI Capability Structure. If a ‘1’ is written to a bit in the IVMC field, then the corresponding interrupt vector is unmasked. Writing a ‘0’ to a bit has no effect. When read, these fields return the current interrupt mask value within the controller (not the value of their register). If a bit has a value of ‘1’, then the corresponding interrupt vector is masked. If a bit has a value of ‘0’, then the corresponding interrupt vector is not masked.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the INTMS and INTMC registers of the NVMe Controller.
2. Configure the NVMe Host to write all zeros to both the INTMS and INTMC registers of the NVMe Controller.
3. Configure the NVMe Host to read the INTMS and INTMC registers of the NVMe Controller once again.

**Observable Results:**
1. Verify that the values returned by the registers do not change on subsequent reads after writing zeros to their bits.

**Possible Problems:** None
Test 4.10 – Offset 14h: CC – I/O Completions Queue Entry Size (IOCQES)

**Purpose:** To validate the I/O Completion Queue Entry Size (IOCQES) field of the Controller Configuration (CC) register.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field defines the I/O Completion Queue entry size that is used for the selected I/O Command Set. The required and maximum values for this field are specified in the Identify Controller data structure for each I/O Command Set. The value is in bytes and is specified as a power of two \((2^n)\) (i.e. \(2^n\)).

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CC.IOCQES field (bits 23:20) of the NVMe Controller.

**Observable Results:**
1. Verify that the sizes of I/O completion queue entries are equal to \(2^n\) the power of CC.IOCQES.

**Possible Problems:** None
Test 4.11 – Offset 14h: CC – I/O Submission Queue Entry Size (IOSQES)

**Purpose:** To validate the I/O Submission Queue Entry Size (IOSEQS) field of the Controller Configuration (CC) register.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field defines the I/O Submission Queue entry size that is used for the selected I/O Command Set. The required and maximum values for this field are specified in the Identify Controller data structure for each I/O Command Set. The value is in bytes and is specified as a power of two ($2^n$) (i.e. $2^n$).

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CC.IOSQES field (bits 19:16) of the NVMe Controller.

**Observable Results:**
1. Verify that the sizes of I/O submission queue entries are equal to $2^n$ to the power of CC.IOSQES.

**Possible Problems:** None.
Test 4.12 – Offset 14h: CC – Shutdown Notification (SHN)

Purpose: To validate the Shutdown Notification (SHN) field of the Controller Configuration (CC) register.

References:

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: This field is used to initiate shutdown processing when a shutdown is occurring, (i.e., a power down condition is expected). For a normal shutdown notification, it is expected that the controller is given time to process the shutdown notification. For an abrupt shutdown notification, the host may not wait for shutdown processing to complete before power is lost. The shutdown notification values are defined in Table 10.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
<td>No notification; no effect.</td>
</tr>
<tr>
<td>01b</td>
<td>Normal shutdown notification.</td>
</tr>
<tr>
<td>10b</td>
<td>Abrupt shutdown notification.</td>
</tr>
<tr>
<td>11b</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

This field should be written by host software prior to any power down condition and prior to any change of the PCI power management state. It is recommended that this field also be written prior to a warm reboot. To determine when shutdown processing is complete, refer to CSTS.SHST.

It is recommended that the host wait a minimum of the RTD3 Entry Latency reported in the Identify Controller data structure for the shutdown operations to complete; if the value reported in RTD3 Entry Latency is 0h, then the host should wait for a minimum of one second. It is not recommended to disable the controller via the CC.EN field. This causes a Controller Reset which may impact the time required to complete shutdown processes.

It is safe to power off the controller when CSTS.SHST indicates shutdown processing is complete. It remains safe to power off the controller until CC.EN transitions from ‘0’ to ‘1’. To start executing commands on the controller after a shutdown operation, a Controller Reset is required. This initialization sequence should then be executed.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to read the CC.SHN field (bits 15:14) of the NVMe Controller.
2. Configure the NVMe Host to write a value of 01b to the CC.SHN register field in order to initiate shutdown processing with a normal shutdown notification.
3. After CSTS.SHST transitions back to a value of 10b (shutdown processing complete), configure the NVMe Host to perform a Controller Reset.
4. Configure the NVMe Host to issue an Identify command to the NVMe Controller.
5. Configure the NVMe Host to write a value of 10b to the CC.SHN register field in order to initiate shutdown processing with an abrupt shutdown notification.
6. After CSTS.SHST transitions back to a value of 10b (shutdown processing complete), configure the NVMe Host to perform a Controller Reset.
7. Configure the NVMe Host to issue an Identify command to the NVMe Controller.
Observable Results:
1. Verify that the value of the CC.SHN field is set to its default value of 00b (no notification) after a Controller Level Reset.
2. After the NVMe Host sets the value of the CC.SHN field, verify that the controller sets the CSTS.SHST field to its appropriate value as indicated in Table 13.
3. Verify that, after the NVMe Host performs a Controller Reset while CSTS.SHST indicates normal operation, the NVMe Host is able to issue commands to the NVMe Controller.

Possible Problems: None
Test 4.13 – Offset 14h: CC – Arbitration Mechanism Selected (AMS)

**Purpose:** To validate the Arbitration Mechanism Selected (AMS) field of the Controller Configuration (CC) register.

**References:**


**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field selects the arbitration mechanism to be used. This value shall only be changed when EN is cleared to ‘0’. Host software shall only set this field to supported arbitration mechanisms indicated in CAP.AMS. If this field is set to an unsupported value, the behavior is undefined. The arbitration mechanism values are defined in Table 11.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000b</td>
<td>Round Robin</td>
</tr>
<tr>
<td>001b</td>
<td>Weighted Round Robin with Urgent</td>
</tr>
<tr>
<td>010b – 110b</td>
<td>Reserved</td>
</tr>
<tr>
<td>111b</td>
<td>Vendor Specific.</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to read the CC.AMS field (bits 13:11) of the NVMe Controller.
2. For each optional arbitration mechanism reported to be supported by the CAP.AMS field:
   a. Configure the NVMe Host to set the CC.AMS to the appropriate value for that mechanism.
   b. Configure the NVMe Host to read the CC.AMS field of the NVMe Controller.

**Observable Results:**

1. Verify that the value of the CC.AMS field is set to its default value of 000b after a Controller Level Reset.
2. Verify that the NVMe Controller properly allows the NVMe Host to set supported values in the CC.AMS field.

**Possible Problems:** None
Test 4.14 – Offset 14h: CC – I/O Command Set Selected (CSS)

**Purpose:** To validate the I/O Command set Selected (CSS) field of the Controller Configuration (CC) register.

**References:**


**Resource Requirements:**

Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field specifies the I/O Command Set that is selected for use for the I/O Submission Queues. Host software shall only select a supported I/O Command Set, as indicated in CAP.CSS. This field shall only be changed when the controller is disabled (CC.EN is cleared to ‘0’). The I/O Command Set selected shall be used for all I/O Submission Queues. The command set values are defined in Table 12.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>000b</td>
<td>NVM Command Set</td>
</tr>
<tr>
<td>001b – 111b</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Test Procedure:**

1. Configure the NVMe Host to read the CC.CSS field (bits 06:04) of the NVMe Controller.
2. For each I/O Command Set supported by the NVMe Controller as indicated by the CAP.CSS field:
   a. Configure the NVMe Host to set the CC.CSS value to the appropriate value for that I/O Command Set.
   b. Configure the NVMe Host to read the CC.CSS field of the NVMe Controller.

**Observable Results:**

1. Verify that the value of the CC.CSS field is set to its default value of 000b after a Controller Level Reset.
2. Verify that the NVMe Controller properly allows the NVMe Host to set supported values in the CC.CSS field.

**Possible Problems:** None
Test 4.15 – Offset 14h: CC – Enable (EN)

**Purpose:** To validate the Enable (EN) field of the Controller Configuration (CC) register.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 21, 2012

**Discussion:** When set to ‘1’, the controller shall process commands based on Submission Queue Tail doorbell writes. When cleared to ‘0’, the controller shall not process commands nor post completion queue entries to Completion Queues. When this field transitions from ‘1’ to ‘0’, the controller is reset (referred to as a Controller Reset). The reset deletes all I/O Submission Queues and I/O Completion Queues, resets the Admin Submission Queue and Completion Queue, and brings the hardware to an idle state. The reset does not affect PCI Express registers nor the Admin Queue registers (AQA, ASQ, or ACQ). All other controller registers defined in section 3 of the NVMe specification and internal controller state (e.g., Feature values defined in section 5 of the NVMe specification that are not persistent across power states) are reset to their default values. The controller shall ensure that there is no data loss for commands that have had corresponding completion queue entries posted to an I/O Completion Queue prior to the reset operation. Refer to section 7.3 for reset details.

When this field is cleared to ‘0’, the CSTS.RDY bit is cleared to ‘0’ by the controller once the controller is ready to be re–enabled. When this field is set to ‘1’, the controller sets CSTS.RDY to ‘1’ when it is ready to process commands. CSTS.RDY may be set to ‘1’ before namespace(s) are ready to be accessed.

Setting this field from a ‘0’ to a ‘1’ when CSTS.RDY is a ‘1’, or setting this field from a ‘1’ to a ‘0’ when CSTS.RDY is a ‘0’ has undefined results. The Admin Queue registers (AQA, ASQ, and ACQ) shall only be modified when this field is cleared to ‘0’.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CC.EN field (bit 00) of the NVMe Controller.

**Observable Results:**
1. Verify that if the Bit 00 is set to value of ‘1’ then the controller shall process commands based on Submission Queue Tail doorbell writes.
2. Verify that if the Bit 00 is set to value of ‘0’ then the controller shall not process commands nor post completion queue entries to Completion Queues.

**Possible Problems:** None
Test 4.16 – Offset 1Ch: CSTS – Shutdown Status (SHST)

**Purpose:** To validate the Shutdown Status (SHST) field of the Controller Status (CSTS) register.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field indicates the status of shutdown processing that is initiated by the host setting the CC.SHN field. The shutdown status values are defined in Table 13.

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b</td>
<td>Normal operation (no shutdown has been requested)</td>
</tr>
<tr>
<td>01b</td>
<td>Shutdown processing occurring</td>
</tr>
<tr>
<td>10b</td>
<td>Shutdown processing complete</td>
</tr>
<tr>
<td>11b</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 13 – CSTS.SHST Field Values

To start executing commands on the controller after a shutdown operation (CSTS.SHST set to 10b), a reset (CC.EN cleared to ‘0’) is required. If host software submits commands to the controller without issuing a reset, the behavior is undefined.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CSTS.SHST field (bits 03:02) of the NVMe Controller.
2. Configure the NVMe Host to write a value of 01b to the CC.SHN register field in order to initiate shutdown processing with a normal shutdown notification.
3. Configure the NVMe Host to read the CSTS.SHST field of the NVMe Controller.
4. After CSTS.SHST transitions back to a value of 10b (shutdown processing complete), configure the NVMe Host to perform a Controller Reset.
5. Configure the NVMe Host to read the CSTS.SHST field of the NVMe Controller.
6. Configure the NVMe Host to issue an Identify command to the NVMe Controller.
7. Configure the NVMe Host to write a value of 10b to the CC.SHN register field in order to initiate shutdown processing with an abrupt shutdown notification.
8. Configure the NVMe Host to read the CSTS.SHST field of the NVMe Controller.
9. After CSTS.SHST transitions back to a value of 10b (shutdown processing complete), configure the NVMe Host to perform a Controller Reset.
10. Configure the NVMe Host to read the CSTS.SHST field of the NVMe Controller.
11. Configure the NVMe Host to issue an Identify command to the NVMe Controller.

**Observable Results:**
1. Verify that the value of the CSTS.SHST field is set to its default value of 00b (normal operation) after a Controller Level Reset.
2. After the NVMe Host sets the value of the CC.SHN field to either 01b or 10b, verify that the NVMe Controller sets the CSTS.SHST field to 01b to indicate that shutdown processing is occurring.
3. Record the time it takes for the NVMe Controller to transition the value of CSTS.SHST from 01b to 10b (i.e. the shutdown processing time) for both normal and abrupt shutdown notifications.
4. Verify that, after the NVMe Host performs a Controller Reset while CSTS.SHST indicates normal operation, the NVMe Host is able to issue commands to the NVMe Controller.
Possible Problems: None
Test 4.17 – Offset 1Ch: CSTS – Controller Fatal Status (CFS)

**Purpose:** To validate the Controller Fatal Status (CFS) field of the Controller Status (CSTS) register.

**References:**

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** This field is set to ‘1’ when a fatal controller error occurred that could not be communicated in the appropriate Completion Queue. This field is cleared to ‘0’ when a fatal controller error has not occurred. Refer to section 9.5.

The reset value of this field is ‘1’ when a fatal controller error is detected during controller initialization.

Since there is no means for a host to trigger a controller fatal status, there is no means to validate the value stored in this register field, and so this test is designed purely to determine the value the NVMe Controller returns when the register is read, and is therefore considered an informative test.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CSTS.CFS field (bit 01) of the NVMe Controller.

**Observable Results:**
1. Report the value of the CSTS.CFS field.

**Possible Problems:** None
Group 5: System Memory Structure

Overview:

This section describes a method for performing conformance verification for NVMe products implementing the NVMe System Memory Structure.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 5.1 – Page Base Address and Offset (PBAO)

**Purpose:** To validate the Page Base Address and Offset field of PRP entries.

**References:**
- NVMe Specification 4.3

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** A physical region page (PRP) entry is a point to a physical memory page. PRPs are used as a scatter/gather mechanism for data transfers between the controller and memory. To enable efficient out of order data transfers between the controller and the host, PRP entries are a fixed size.

The size of the physical memory page is configured by host software in CC.MPS. Figure 6 shows the layout of a PRP entry that consists of a Page Base Address and an Offset. The size of the Offset field is determined by the physical memory page size configured in CC.MPS.

The Page Base Address and Offset (PBAO) field indicates the 64–bit physical memory page address. The lower bits (n:2) of this field indicate the offset within the memory page. If the memory page size is 4KB, then bits 11:02 form the Offset; if the memory page size is 8KB, then bits 12:02 form the Offset, etc (i.e. n = CC.MPS + 11). If this entry is not the first PRP entry in the command or a PRP List pointer in a command, then the Offset portion of this field shall be cleared to 0h.

![Figure 6 – PRP Entry Layout](image)

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue a Write command with properly formatted PRP entries to the NVMe Controller.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the associated I/O Completion Queue indicating the status for the command.

**Possible Problems:** None.
Test 5.2 – Completion Queue Entry

**Purpose:** To verify that an NVMe Device can properly post a completion queue entry.

**References:**
NVMe Specification 4.6

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** An entry in the Completion Queue is 16 bytes in size. Figure 7 describes the layout of the Completion Queue Entry data structure. The contents of Dword 0 are command specific. If a command uses Dword 0, then the definition of this Dword is contained within the associated command definition. If a command does not use Dword 0, then the field is reserved. Dword 1 is reserved. Dword 2 is defined in Figure 26 and Dword 3 is defined in Figure 27 of the NVMe specification. Any additional I/O Command Set defined in the future may use an alternate Completion Queue entry size or format.

**Figure 7 – Completion Queue Entry Layout – Admin and NVM Command Set**

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>23</th>
<th>15</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DW0</strong></td>
<td>Command Specific</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DW1</strong></td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DW2</strong></td>
<td>SQ Identifier</td>
<td>SQ Head Pointer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DW3</strong></td>
<td>Status Field</td>
<td>P</td>
<td>Command Identifier</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Status Field (SF) of Dword 3 is tested in Test 5.3 – Status Field Definition.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command to the NVMe Controller.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the Admin Completion Queue indicating the status for the command.
2. Verify that the completion queue entry in the Completion Queue is 16 bytes in size.
3. Verify that the Submission Queue Identifier (SQID) field (bits 31:16) of Dword 2 is 0 (i.e. the Admin Submission Queue).
4. Verify that the Phase Tag (P) field (bit 16) of Dword 3 is set to 1.
5. Verify that the Command Identifier (CID) field (bits 15:00) of Dword 3 matches the command ID assigned to the command by the NVMe Host when it was issued to the Submission Queue.

**Possible Problems:** None.
Test 5.3 – Status Field Definition

**Purpose:** To verify that an NVMe Controller can properly return the status for a command in the completion queue entry.

**References:**
NVMe Specification 4.6.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** The Status Field (SF) defines the status for the command indicated in the completion queue entry. If the SCT and SC fields are cleared to 0h, then this indicates a successful command completion, with no fatal or non-fatal error conditions.

The Status Field is bits 31:17 of Dword 3 of all completion queue entries.

The SCT and SC fields are further tested in subsequent tests.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command to the NVMe Controller.
2. After the NVMe Controller has processed the command, configure the NVMe Host to reap the completion queue entry for the Identify command from the Admin Completion Queue.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the Admin Completion Queue indicating the status for the command.
2. Verify that the Status Code Type (SCT) field of the Status Field is 0h to indicate a Generic Command Status. Verify that the Status Code (SC) field of the Status Field is 0h to indicate successful completion of the command.

**Possible Problems:** None.
Test 5.4 – Generic Command Status Definition

**Purpose:** To verify that an NVMe Host can properly return Status Code values for the Generic Command Status type.

**References:**
NVMe Specification 4.6.1.2.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** Completion queue entries with a Status Code type of Generic Command Status (0h) indicate a status value associated with the command that is generic across all command types. The Status Code values for the Generic Command Status type are defined in Table 15 and Table 14.

**Table 14 – Generic Command Status Values, NVM Command Set**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>80h</td>
<td>LBA Out of Range: The command references an LBA that exceeds the size of the namespace.</td>
</tr>
<tr>
<td>81h</td>
<td>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.</td>
</tr>
<tr>
<td>82h</td>
<td>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.</td>
</tr>
<tr>
<td>83h</td>
<td>Reservation Conflict: The command was aborted due to a conflict with a reservation held on the accessed namespace.</td>
</tr>
<tr>
<td>84h</td>
<td>Format In Progress: The namespace is currently being formatted. The Do Not Retry bit shall be cleared to ‘0’ to indicate that the command may succeed if it is resubmitted.</td>
</tr>
<tr>
<td>85h – BFh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Test Procedure:**
1. For each of the Status Code values defined in Table 14 and Table 15, configure the NVMe Host to issue a command to the NVMe Controller which will cause the controller to return the Status Code in the completion queue entry for the command.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the SCT field of the Status Field is cleared to 0h. Verify that the SC field of the Status field matches the expected Status Code.

**Possible Problems:** The NVMe specification does not explicitly state the exact conditions for when an NVMe Controller should use some of the defined Status Codes. Such status codes cannot be tested.
Table 15 – Generic Command Status Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td><strong>Successful Completion:</strong> The command completed successfully.</td>
</tr>
<tr>
<td>01h</td>
<td><strong>Invalid Command Opcode:</strong> The associated command opcode field is not valid.</td>
</tr>
<tr>
<td>02h</td>
<td><strong>Invalid Field in Command:</strong> An invalid field in the command parameters.</td>
</tr>
<tr>
<td>03h</td>
<td><strong>Command ID Conflict:</strong> The command identifier is already in use. Note: It is implementation specific how many commands are searched for a conflict.</td>
</tr>
<tr>
<td>04h</td>
<td><strong>Data Transfer Error:</strong> Transferring the data or metadata associated with a command had an error.</td>
</tr>
<tr>
<td>05h</td>
<td><strong>Commands Aborted due to Power Loss Notification:</strong> Indicates that the command was aborted due to a power loss notification.</td>
</tr>
<tr>
<td>06h</td>
<td><strong>Internal Error:</strong> The command was not completed successfully due to an internal error. Details on the internal device error are returned as an asynchronous event. Refer to section 5.2.</td>
</tr>
<tr>
<td>07h</td>
<td><strong>Command Abort Requested:</strong> The command was aborted due to a Command Abort command being received that specified the Submission Queue Identifier and Command Identifier of this command.</td>
</tr>
<tr>
<td>08h</td>
<td><strong>Command Aborted due to SQ Deletion:</strong> The command was aborted due to a Delete I/O Submission Queue request received for the Submission Queue to which the command was submitted.</td>
</tr>
<tr>
<td>09h</td>
<td><strong>Command Aborted due to Failed Fused Command:</strong> The command was aborted due to the other command in a fused operation failing.</td>
</tr>
<tr>
<td>0Ah</td>
<td><strong>Command Aborted due to Missing Fused Command:</strong> The command was aborted due to the companion fused command not being found as the subsequent Submission Queue entry.</td>
</tr>
<tr>
<td>0Bh</td>
<td><strong>Invalid Namespace or Format:</strong> The namespace or the format of that namespace is invalid.</td>
</tr>
<tr>
<td>0Ch</td>
<td><strong>Command Sequence Error:</strong> The command was aborted due to a protocol violation in a multi-command sequence (e.g. a violation of the Security Send and Security Receive sequencing rules in the TCG Storage Synchronous Interface Communications protocol).</td>
</tr>
<tr>
<td>0Dh</td>
<td><strong>Invalid SGL Last Segment Descriptor:</strong> The command includes an invalid SGL Last Segment. This may occur when the SGL segment pointed to by an SGL Last Segment descriptor contains an SGL Segment descriptor or an SGL Last Segment descriptor. This may occur when an SGL Last Segment descriptor contains an invalid length (i.e., a length of zero or one that is not a multiple of 16).</td>
</tr>
<tr>
<td>0Eh</td>
<td><strong>Invalid Number of SGL Descriptors:</strong> The number of SGL descriptors or SGL Last Segment descriptors in a SGL segment is greater than one.</td>
</tr>
<tr>
<td>0Fh</td>
<td><strong>Data SGL Length Invalid:</strong> The length of a Data SGL is too short or too long.</td>
</tr>
<tr>
<td>10h</td>
<td><strong>Metadata SGL Length Invalid:</strong> The length of a Metadata SGL is too short or too long.</td>
</tr>
<tr>
<td>11h</td>
<td><strong>SGL Descriptor Type Invalid:</strong> The type of an SGL Descriptor is a type that is not supported by the controller.</td>
</tr>
<tr>
<td>12h</td>
<td><strong>Invalid Use of Controller Memory Buffer:</strong> The attempted use of the Controller Memory Buffer is not supported by the controller.</td>
</tr>
<tr>
<td>13h</td>
<td><strong>PRP Offset Invalid:</strong> The Offset field for a PRP entry is invalid. This may occur when there is a PRP entry with a non-zero offset after the first entry.</td>
</tr>
<tr>
<td>14h</td>
<td><strong>Atomic Write Unit Exceeded:</strong> The length specified exceeds the atomic write unit size.</td>
</tr>
<tr>
<td>15h–7Fh</td>
<td>Reserved</td>
</tr>
<tr>
<td>80h–BFh</td>
<td>I/O Command Set Specific</td>
</tr>
<tr>
<td>C0h–FFh</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>
Test 5.5 – Command Specific Errors Definition

Purpose: To verify that an NVMe Controller can properly return the Status Code values for the Command Specific Status type.

References:
NVMe Specification 4.5.1.2.2

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: Completion queue entries with a Status Code Type of Command Specific Errors (01h) indicate an error that is specific to a particular command opcode. These status values may indicate additional processing is required. The Status Code values for the Command Specific Status Type are defined in Table 17 and Table 16.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Commands Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>80h</td>
<td>Conflicting Attributes</td>
<td>Dataset Management, Read, Write</td>
</tr>
<tr>
<td>81h</td>
<td>Invalid Protection Information</td>
<td>Compare, Read, Write, Write Zeroes</td>
</tr>
<tr>
<td>82h</td>
<td>Attempted Write to Read Only Range</td>
<td>Dataset Management, Write, Write Uncorrectable, Write Zeroes</td>
</tr>
<tr>
<td>83h – BFh</td>
<td>Reserved</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 16 – Command Specific Status Values, NVM Command Set

Test Setup: See Appendix A.

Test Procedure:
1. For each of the Status Code values defined in Table 17 and Table 16, configure the NVMe Host to issue a command to the NVMe Controller which will cause the controller to return the Status Code in the completion queue entry for the command.

Observable Results:
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the SCT field of the Status Field is set to 1h. Verify that the SC field of the Status field matches the expected Status Code.

Possible Problems: The NVMe specification does not explicitly state the exact conditions for when an NVMe Controller should use some of the defined Status Codes. Such status codes cannot be tested. Additionally, some of the Status Codes can only be used for optional commands which the NVMe controller may or may not support.
## Table 17 – Command Specific Status Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Commands Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Completion Queue Invalid</td>
<td>Create I/O Submission Queue</td>
</tr>
<tr>
<td>01h</td>
<td>Invalid Queue Identifier</td>
<td>Create I/O Submission Queue, Create I/O Completion Queue, Delete I/O Submission Queue, Delete I/O Completion Queue</td>
</tr>
<tr>
<td>02h</td>
<td>Invalid Queue Size</td>
<td>Create I/O Submission Queue, Create I/O Completion Queue</td>
</tr>
<tr>
<td>03h</td>
<td>Abort Command Limit Exceeded</td>
<td>Abort</td>
</tr>
<tr>
<td>04h</td>
<td>Reserved</td>
<td>N/A</td>
</tr>
<tr>
<td>05h</td>
<td>Asynchronous Event Request Limit Exceeded</td>
<td>Asynchronous Event Request</td>
</tr>
<tr>
<td>06h</td>
<td>Invalid Firmware Slot</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>07h</td>
<td>Invalid Firmware Image</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>08h</td>
<td>Invalid Interrupt Vector</td>
<td>Create I/O Completion Queue</td>
</tr>
<tr>
<td>09h</td>
<td>Invalid Log Page</td>
<td>Get Log Page</td>
</tr>
<tr>
<td>0Ah</td>
<td>Invalid Format</td>
<td>Format NVM</td>
</tr>
<tr>
<td>0Bh</td>
<td>Firmware Activation Requires Conventional Reset</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>0Ch</td>
<td>Invalid Queue Deletion</td>
<td>Delete I/O Completion Queue</td>
</tr>
<tr>
<td>0Dh</td>
<td>Feature Identifier Not Saveable</td>
<td>Set Features</td>
</tr>
<tr>
<td>0Eh</td>
<td>Feature Not Changeable</td>
<td>Set Features</td>
</tr>
<tr>
<td>0Fh</td>
<td>Feature Not Namespace Specific</td>
<td>Set Features</td>
</tr>
<tr>
<td>10h</td>
<td>Firmware Activation Required NVM Subsystem Reset</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>11h</td>
<td>Firmware Activation Requires Reset</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>12h</td>
<td>Firmware Activation Requires Maximum Time Violation</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>13h</td>
<td>Firmware Activation Prohibited</td>
<td>Firmware Commit</td>
</tr>
<tr>
<td>14h</td>
<td>Overlapping Range</td>
<td>Firmware Commit, Firmware Image Download, Set Features</td>
</tr>
<tr>
<td>15h</td>
<td>Namespace Insufficient Capacity</td>
<td>Namespace Management</td>
</tr>
<tr>
<td>16h</td>
<td>Namespace Identifier Unavailable</td>
<td>Namespace Management</td>
</tr>
<tr>
<td>17h</td>
<td>Reserved</td>
<td>N/A</td>
</tr>
<tr>
<td>18h</td>
<td>Namespace Already Attached</td>
<td>Namespace Attachment</td>
</tr>
<tr>
<td>19h</td>
<td>Namespace Is Private</td>
<td>Namespace Attachment</td>
</tr>
<tr>
<td>1Ah</td>
<td>Namespace Not Attached</td>
<td>Namespace Attachment</td>
</tr>
<tr>
<td>1Bh</td>
<td>Thin Provisioning Not Supported</td>
<td>Namespace Management</td>
</tr>
<tr>
<td>1Ch</td>
<td>Controller List Invalid</td>
<td>Namespace Attachment</td>
</tr>
<tr>
<td>1Dh – 7Fh</td>
<td>Reserved</td>
<td>N/A</td>
</tr>
<tr>
<td>80h – BFh</td>
<td>I/O Command Set Specific</td>
<td></td>
</tr>
<tr>
<td>C0h – FFh</td>
<td>Vendor Specific</td>
<td></td>
</tr>
</tbody>
</table>
Test 5.6 – Media and Data Integrity Errors Definition

**Purpose:** To verify that an NVMe Host can properly return the status for the command in the completion queue entry.

**References:**
- NVMe Specification 4.5.1.2.3

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** Completion queue entries with a Status Code Type of Media and Data Integrity Errors (02h) indicate a media specific error that occurred in the NVM or data integrity type errors. The Status Code values for the Media and Data Integrity Errors Status Type are defined in Table 18 and Table 19.

### Table 18 – Media and Data Integrity Error Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h – 7Fh</td>
<td>Reserved</td>
</tr>
<tr>
<td>80h – BFh</td>
<td>I/O Command Set Specific</td>
</tr>
<tr>
<td>C0h – FFh</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>

### Table 19 – Media and Data Integrity Error Values, NVM Command Set

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>80h</td>
<td><strong>Write Fault:</strong> The write data could not be committed to the media.</td>
</tr>
<tr>
<td>81h</td>
<td><strong>Unrecovered Read Error:</strong> The read data could not be recovered from the media.</td>
</tr>
<tr>
<td>82h</td>
<td><strong>End–to–end Guard Check Error:</strong> The command was aborted due to an end–to–end guard check failure.</td>
</tr>
<tr>
<td>83h</td>
<td><strong>End–to–end Application Tag Check Error:</strong> The command was aborted due to an end–to–end application tag check failure.</td>
</tr>
<tr>
<td>84h</td>
<td><strong>End–to–end Reference Tag Check Error:</strong> The command was aborted due to an end–to–end reference tag check failure.</td>
</tr>
<tr>
<td>85h</td>
<td><strong>Compare Failure:</strong> The command failed due to a miscompare during a Compare command.</td>
</tr>
<tr>
<td>86h</td>
<td><strong>Access Denied:</strong> Access to the namespace and/or LBA range is denied due to lack of access rights. Refer to TCG SIIS.</td>
</tr>
<tr>
<td>87h</td>
<td><strong>Deallocated or Unwritten Logical Block:</strong> The command failed due to an attempt to read from an LBA range containing a deallocated or unwritten logical block.</td>
</tr>
<tr>
<td>88h – BFh</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Test Setup:** See Appendix A.

**Test Procedure:**
1. For each of the Status Code values defined in Table 18 and Table 19, configure the NVMe Host to issue a command to the NVMe Controller which will cause the controller to return the Status Code in the completion queue entry for the command.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the SCT field of the Status Field is set to 2h. Verify that the SC field of the Status field matches the expected Status Code.
Possible Problems: The NVMe specification does not explicitly state the exact conditions for when an NVMe Controller should use some of the defined Status Codes. Such status codes cannot be tested. Additionally, some of the Status Codes can only be used for optional commands which the NVMe controller may or may not support.
Group 6: Controller Architecture

Overview:

This section describes a method for performing conformance verification for NVMe products implementing the NVMe Controller Architecture.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 6.1 – Controller Level Reset – Conventional Reset

Purpose: To verify that an NVMe system performs the proper actions when a Conventional Reset occurs.

References:
NVMe Specification 7.3.2

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: When a Controller Level Reset occurs, the Host and Controller are required to take specific action.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Device to perform a Conventional Controller Level Reset. Repeat this for each of the following cases:
   a. PCI Express Warm reset (if there is a means provided by the vendor)
   b. PCI Express Cold reset
2. When the reset is complete, configure the NVMe Host to issue a Write and then a Read command to the NVMe Controller.

Observable Results:
1. Verify that the NVMe Controller is able to properly execute NVMe Write and Read commands after the reset is complete.
2. Verify that the NVMe Controller performs the following actions when each reset case defined above is initiated:
   a. The controller stops processing any outstanding Admin or I/O commands.
   b. All I/O Submission Queues are deleted.
   c. All I/O Completion Queues are deleted.
   d. The controller is brought to an Idle state. When this is complete, CSTS.RDY is cleared to ‘0’.
   e. All controller registers defined in section 3 of the NVMe Specification and internal controller state are reset.

Possible Problems: A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 6.2 – Controller Level Reset – Function Level Reset

Purpose: To verify that an NVMe system performs the proper actions when a Function Level Reset occurs.

References:
NVMe Specification 7.3.2

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: June 24, 2015

Discussion: When a Controller Level Reset occurs, the Host and Controller are required to take specific action.

Support for the Function Level Reset mechanism is indicated in the Function Level Reset Capability (FLRC) field (bit 28) of the PCI Express Device Capabilities (PXDCAP) (offset PXCAP + 4h) PCI Express Register. All NVMe Controllers must support Function Level Reset and so this register value should be set to ‘1’ for all NVMe Controllers.

A Function Level Reset is initiated by the NVMe Host by writing a value of ‘1’ to the Initiate Function Level Reset field (bit 15) of the PCI Express Device Control (PXDC) (offset PXCAP + 8h) PCI Express Register. The value read by software from this field shall always be ‘0’.

Test Setup: See Appendix A.

Test Procedure:
1. Configure the NVMe Host to read the PCDCAP.FLRC PCI Express register field.
2. Configure the NVMe Host to read the Initiate Function Level Reset field of the PXDC PCI Express register.
3. Configure the NVMe Host to write a value of ‘1’ to the Initiate Function Level Reset field of the PXDC PCI Express register in order to initiate a Function Level Reset NVMe Host.
4. When the reset is complete, configure the NVMe Host to issue a Write and then a Read command to the NVMe Controller.

Observable Results:
1. Verify that the value read from the PCDCAP.FLRC PCI Express register field is ‘1’ to indicate support for the Function Level Reset mechanism.
2. Verify that the value read from the Initiate Function Level Reset field of the PXDC PCI Express register is ‘0’.
3. Verify that the NVMe Controller is able to properly execute NVMe Write and Read commands after the reset is complete.
4. Verify that the NVMe Controller performs the following actions when the Function Level Reset is initiated:
   a. The controller stops processing any outstanding Admin or I/O commands.
   b. All I/O Submission Queues are deleted.
   c. All I/O Completion Queues are deleted.
   d. The controller is brought to an Idle state. When this is complete, CSTS.RDY is cleared to ‘0’.
   e. All controller registers defined in section 3 of the NVMe Specification and internal controller state are reset.

Possible Problems: A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 6.3 – Controller Level Reset – Controller Reset

**Purpose:** To verify that an NVMe system performs the proper actions when a Controller Reset occurs.

**References:**
NVMe Specification 7.3.2

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** When a Controller Level Reset occurs, the Host and Controller are required to take specific action.

A Controller Reset is initiated when the CC.EN controller register field transitions from ‘1’ to ‘0’. This is performed by the NVMe Host by writing a value of ‘0’ to the CC.EN while the CC.EN field is set to ‘1’.

**Test Setup:** See Appendix A.

**Test Procedure**
1. Configure the NVMe Host to write a value of ‘0’ to the CC.EN controller register field.
2. When the reset is complete, configure the NVMe Host to issue a Write and then a Read command to the NVMe Controller.

**Observable Results:**
1. Verify that the NVMe Controller is able to properly execute NVMe Write and Read commands after the reset is complete.
2. Verify that the NVMe Controller performs the following actions when the Controller Reset is initiated:
   a. The controller stops processing any outstanding Admin or I/O commands.
   b. All I/O Submission Queues are deleted.
   c. All I/O Completion Queues are deleted.
   d. The controller is brought to an Idle state. When this is complete, CSTS.RDY is cleared to ‘0’.
   e. The Admin Queue registers (AQA, ASQ, or ACQ) are not reset as part of a controller reset. All other controller registers defined in section 3 of the NVMe Specification and internal controller state are reset.

**Possible Problems:** None.
Test 6.4 – Controller Level Reset – NVM Subsystem Reset

**Purpose:** To verify that an NVMe system performs the proper actions when an NVM Subsystem Reset occurs.

**References:**
NVMe Specification 7.3.1 and 7.3.2

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** June 24, 2015

**Discussion:** When a Controller Level Reset occurs, the Host and Controller are required to take specific action.

An NVM Subsystem Reset is initiated when:
1. Power is applied to the NVM System,
2. A value of 4E564D65h (“NVMe”) is written to the NSSR.NSSRC controller register field, or
3. A vendor specific event occurs.

When an NVM Subsystem Reset occurs, the entire NVM subsystem is reset. This includes the initiation of a Controller Level Reset on all controllers that make up the NVM subsystem and a transition to the Detect LTSSM state by all PCI Express ports of the NVM subsystem.

The occurrence of an NVM Subsystem Reset while power is applied to the NVM subsystem is reported by the initial value of the CSTS.NSSRO field following the NVM Subsystem Reset. This field may be used by host software to determine if the sudden loss of communication with a controller was due to an NVM Subsystem Reset or some other condition.

This test is only applicable if the CAP.NSSRS field is set ‘1’ to indicate support for writing to the NSSR.NSSRC field.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Configure the NVMe Host to read the CAP.NSSRS field to determine if the DUT supports writing to the NSSR.NSSRC field. If the NSSRS field is cleared to ‘0’, the test is not applicable. If the CAP.NSSRS field is set to ‘1’, continue to step 2.
2. Configure the NVMe host to write a value of 4E564D65h (“NVMe”) to the NSSR.NSSRC field.
3. When the reset is complete, configure the NVMe Host to issue a Write and then a Read command to the NVMe Controller.

**Observable Results:**
1. Verify that the NVMe Controller is able to properly execute NVMe Write and Read commands after the reset is complete.
2. Verify that the NVMe Controller performs the following actions when the NVM Subsystem Reset is initiated:
   a. The controller stops processing any outstanding Admin or I/O commands.
   b. All I/O Submission Queues are deleted.
   c. All I/O Completion Queues are deleted.
   d. The controller is brought to an Idle state. When this is complete, CSTS.RDY is cleared to ‘0’.
   e. All controller registers defined in section 3 of the NVMe Specification and internal controller state are reset.

**Possible Problems:** The DUT may or may not support NVM Subsystem Reset, an optional NVMe feature.
Group 7: Reservations

Overview:

This section describes a method for performing conformance verification for NVMe products implementing NVMe Reservations. These tests are not applicable to devices that do not claim to support reservations.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 7.1 – Controller and Namespace Support for Reservations

Purpose: To determine if an NVMe Controller properly supports reservations.

References:
NVMe Specification 8.8

Resource Requirements:
Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: September 18, 2014

Discussion: If a controller supports reservations, then the controller shall:
- Indicate support for reservations by returning a ‘1’ in bit 5 of the Optional NVM Command Support (ONCS) field in the Identify Controller data structure;

If a namespace supports reservations, then the namespace shall:
- Report a non-zero value in the Reservation Capabilities (RESCAP) field in the Identify Namespace data structure.

If a host submits a command associated with reservations (i.e., Reservation Report, Reservation Register, Reservation Acquire, and Reservation Release) to a controller or a namespace that do not both support reservations, then the command is aborted by the controller with status Invalid Command Opcode.

This test is not applicable to devices that do not claim to support reservations.

Test Setup: See Appendix A.

Test Procedure:
1. The Testing Station, acting as an NVMe host, requests the Identify Controller Data Structure.
2. The Testing Station, acting as an NVMe host, requests the Identify Namespace Data Structure.
3. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
4. The Testing Station issues a Reservation Report command.

Observable Results:
1. Verify that if the NVMe Device indicates support for Reservations:
   a. The NVMe device returns a ‘1’ in bit 5 of the Optional NVM Command Support (ONCS) field in the Identify Controller data structure;
   b. The NVMe device reports a non-zero value in the Reservation Capabilities (RESCAP) field in the Identify Namespace data structure.
   c. The NVMe device responds to the Reservation Report command with status complete.
2. Verify that if the NVMe Device indicates no support for Reservations:
   a. The NVMe device returns a ‘0’ in bit 5 of the Optional NVM Command Support (ONCS) field in the Identify Controller data structure;
   b. The NVMe device reports a zero value in the Reservation Capabilities (RESCAP) field in the Identify Namespace data structure.
   c. The NVMe device responds to the Reservation Report command with status ‘Invalid Command Opcode’.

Possible Problems: A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 7.2 – Register New Reservation

Purpose: To determine if an NVMe Controller properly supports reservations.

References:
   NVMe Specification 8.8

Resource Requirements:
   Tools capable of monitoring and decoding traffic on the NVMe interface.

Last Modification: September 18, 2014

Discussion: A host registers a reservation key by executing a Reservation Register command on the namespace with the Reservation Register Action (RREGA) field set to 000b (i.e., Register Reservation Key) and supplying a reservation key in the New Reservation Key (NRKEY) field.

A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

Test Setup: See Appendix A.

Test Procedure:
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host registers a reservation key by executing a Reservation Register command with the RREGA field set to 000b and supplying a NRKEY.
3. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY matching the NRKEY value provided in the previous step.

Observable Results:
1. Verify that the Reservation Register command completes successfully.

Possible Problems: A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 7.3 – Replace Reservation

**Purpose:** To determine if an NVMe Controller properly supports reservations.

**References:**
- NVMe Specification 8.8

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** September 18, 2014

**Discussion:** A host that is a registrant of a namespace may replace its existing reservation key by executing a Reservation Register command on the namespace with the RREGA field set to 010b (i.e., Replace Reservation Key), supplying the current reservation key in the Current Reservation Key (CRKEY) field, and the new reservation key in the NRKEY field.

A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host registers a reservation key by executing a Reservation Register command with the RREGA field set to 000b and supplying a NRKEY.
3. The Testing Station, acting as an NVMe host replaces its reservation key by executing a Reservation Register command with the RREGA field set to 010b and supplying a CRKEY matching the NRKEY supplied in the previous step.
4. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY matching the CRKEY value provided in the previous step.

**Observable Results:**
1. Verify that both the first and second Reservation Register commands complete successfully.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 7.4 – Replace Reservation – Key Mismatch

**Purpose:** To determine if an NVMe Controller properly supports reservations.

**References:**
NVMe Specification 8.8

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** September 18, 2014

**Discussion:** A host that is a registrant of a namespace may replace its existing reservation key by executing a Reservation Register command on the namespace with the RREGA field set to 010b (i.e., Replace Reservation Key), supplying the current reservation key in the Current Reservation Key (CRKEY) field, and the new reservation key in the NRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host registers a reservation key by executing a Reservation Register command with the RREGA field set to 000b and supplying a NRKEY.
3. The Testing Station, acting as an NVMe host replaces its reservation key by executing a Reservation Register command with the RREGA field set to 010b and supplying a CRKEY not matching the NRKEY supplied in the previous step.
4. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY matching the NRKEY value provided in step 2.

**Observable Results:**
1. Verify that the first Reservation Register command completes successfully.
2. Verify that the second Reservation Register command completes with status ‘Reservation Conflict’.
3. Verify that the third Reservation Register command (unregister) completes successfully.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 7.5 – Replace Reservation – Ignore Existing Key

**Purpose:** To determine if an NVMe Controller properly supports reservations.

**References:**
NVMe Specification 8.8

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** September 18, 2014

**Discussion:**
A host may replace its reservation key without regard to its registration status or current reservation key value by setting the Ignore Existing Key (IEKEY) bit to ‘1’ in the Reservation Register command. Replacing a reservation key has no effect on any reservation that may be held on the namespace.

A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host registers a reservation key by executing a Reservation Register command with the RREGA field set to 000b and supplying a NRKEY.
3. The Testing Station, acting as an NVMe host replaces its reservation key by executing a Reservation Register command with the RREGA field set to 010b and supplying a CRKEY not matching the NRKEY supplied in the previous step, and setting the IEKEY bit to ‘1’.
4. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY matching the NRKEY value provided in step 2.

**Observable Results:**
1. Verify that all 3 Reservation Register commands (register, replace and ignore key, unregister) complete successfully.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
### Test 7.6 – Unregister – Key Mismatch

**Purpose:** To determine if an NVMe Controller properly supports reservations.

**References:**
- NVMe Specification 8.8

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** September 18, 2014

**Discussion:** A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host registers a reservation key by executing a Reservation Register command with the RREGA field set to 000b and supplying a NRKEY.
3. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY not matching the NRKEY value provided in step 2.
4. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b and supplying a CRKEY matching the NRKEY value provided in step 2.

**Observable Results:**
1. Verify that the Reservation Register commands steps 2 and 4 (register, unregister with matching key) complete successfully.
2. Verify that the Reservation Register command in step 3 (unregister with non–matching key) completes with status Reservation Conflict.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 7.7 – Unregister – Non Registrant

**Purpose:** To determine if an NVMe Controller properly supports reservations.

**References:**
- NVMe Specification 8.8

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** September 18, 2014

**Discussion:** A host that is a registrant of a namespace may unregister with the namespace by executing a Reservation Register command on the namespace with the RREGA field set to 001b (i.e., Unregister Reservation Key) and supplying its current reservation key in the CRKEY field. If the contents of the CRKEY field do not match the key currently associated with the host, then the command is aborted with a status of Reservation Conflict. If the host is not a registrant, then the command is aborted with a status of Reservation Conflict.

This test is not applicable to devices that do not claim to support reservations.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. The Testing Station, acting as an NVMe host, registers a Host Identifier with each controller using the Set Features command.
2. The Testing Station, acting as an NVMe host unregisters a reservation key by executing a Reservation Register command with the RREGA field set to 001b.

**Observable Results:**
1. Verify that the Reservation Register command completes with status Reservation Conflict.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Group 8: Autonomous Power State Transitions

Overview:

This section describes a method for performing conformance verification for NVMe products implementing NVMe Autonomous Power State Transitions (APST). These tests are not applicable to devices that do not claim to support APST.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 8.1 – Autonomous Power State Transitions Enabled

**Purpose:** To determine if an NVMe Controller properly supports Autonomous Power State Transitions.

**References:**
NVMe Specification 5.14.1.12, Fig 90, Fig 108, Fig 124

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** April 13, 2015

**Discussion:** An NVMe device may change power states autonomously without host intervention if Autonomous Power State Transitions are supported by the device and enabled by the Host. There are 32 allowable power states defined sequentially in the 256 byte data structure entry.

Each entry is 64 bits long and describes the Idle Time Prior to Transition (ITPTT) and Idle Transition Power State (ITPS). The Idle Transition Power State specifies the next power state the device will transition to after there is a continuous period of idle time in the current power state that exceeds the time specified in the Idle Time Prior to Transition field.

This test is not applicable to devices that do not claim to support Autonomous Power State Transitions.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Check that the DUT supports Autonomous Power State Transitions by setting Bit 0 of Byte 265 of the Identify Controller Data Structure, to 1. If this bit is set to 0, the test is not performed.
2. Using the Set Features Command, enable Feature Identifier 0Ch for Autonomous Power State Transition.
3. Perform the Get Feature Command to see that the Autonomous Power State Transitions feature was enabled (APSTE), and receive the Autonomous Power State Transition data structure.

**Observable Results:**
1. Verify that the controller returns a properly formatted Autonomous Power State Transition data structure.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Test 8.2 – Return from Non–Operational State

**Purpose:** To determine if an NVMe Controller properly supports Autonomous Power State Transitions.

**References:**
NVMe Specification 8.4.1

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** April 13, 2015

**Discussion:** When in a non–operational power state, regardless of whether autonomous power state transitions are enabled, the controller shall autonomously transition back to the last operational power state when an I/O Submission Queue Tail Doorbell is written.

**Test Setup:** See Appendix A.

**Test Procedure:**
1. Check that the DUT supports Autonomous Power State Transitions by setting Bit 0 of Byte 265 of the Identify Controller Data Structure, to 1. If this bit is set to 0, the test is not performed.
2. Using the Set Features Command, enable Feature Identifier 0Ch for Autonomous Power State Transition.
3. Perform the Get Feature Command to see that the Autonomous Power State Transitions feature was enabled (APST), and receive the Autonomous Power State Transition data structure.
4. If the Autonomous Power State Transition Data Structure indicates that the device supports entering a non–operational state via APST, allow the DUT to remain idle for ITPT for each power state successively until the DUT enters a non–operational state.
5. Perform Identify Power State Descriptor Data Structure, check the NOPS field.
6. Perform an NVMe I/O Command, such as NVMe Write to the DUT.
7. Perform Identify Power State Descriptor Data Structure, check the NOPS field.
8. Write to an I/O Submission Queue Tail Doorbell
9. Perform Identify Power State Descriptor Data Structure, check the NOPS field.

**Observable Results:**
1. Verify that the controller returns a properly formatted Autonomous Power State Transition data structure.
2. Verify that after the I/O command in step 6, the DUT does not change power states.
3. Verify that after the writing of the I/O Submission Queue Tail Doorbell, the DUT returns to the last operational power state.

**Possible Problems:** A reliable means of performing this test has not been determined. Therefore, this test should not be included in any industry approved determination of conformance.
Group 9: Namespace Management

Overview:

This section describes a method for performing conformance verification for NVMe products implementing NVMe Namespace Management.

Notes:

The preliminary draft descriptions for the tests defined in this group are considered complete, and the tests are pending implementation (during which time additional revisions/modifications are likely to occur).
Test 9.1 – Namespace Management Identify Command

**Purpose:** To determine if an NVMe Controller properly implements the Namespace Management Identify Command.

**References:**
NVMe Specification 8.11

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** November 23, 2015

**Discussion:** Additional CNS values were added to the Identify command relating to namespace management. Refer to table 1.

**Test Setup:** See Appendix A.

**Case 1:** CNS 10h & 11h - Namespace Lists

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command specifying CNS value 10h and CDW1.NSID 00h to the controller in order to receive back a Namespace List containing namespaces present within the NVM subsystem.
2. For each namespace returned in the Namespace List from the previous step, configure the NVMe Host to issue an Identify command specifying CNS value 11h and setting CDW1.NSID to the namespace identifier of the namespace to the controller in order to receive back an Identify Namespace data structure for the specified namespace.

**Observable Results:**
1. Verify that the requested data structures are posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
2. If a specified namespace ID is inactive, verify that the data structure returned by the controller is zero filled.
3. Verify that unused entries in the Namespace List are zero filled.

**Case 2:** CNS 12h - Controller List 1

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command specifying CNS value 02h and CDW1.NSID 00h to the controller in order to receive back a Namespace List containing active namespaces attached to the controller.
2. For each namespace returned in the Namespace List from the previous step, configure the NVMe Host to issue an Identify command specifying CNS value 12h, CDW10.CNTID value 00h, and setting CDW1.NSID to the namespace identifier of the namespace to the controller in order to receive back a Controller List containing the controller identifiers of all controllers attached to the namespace.

**Observable Results:**
1. Verify that the requested data structures are posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
2. Verify that the controller identifier of the controller is contained within each Controller List returned by the controller.
3. Verify that each Controller List contains valid values and that unused entries in the Controller List are zero filled.

**Case 3: CNS 13h - Controller List 2**

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command specifying CNS value 13h and CDW10.CNTID value 00h to the controller in order to receive back a Controller List containing the controller identifiers of all controllers in the NVM subsystem.

**Observable Results:**
1. Verify that the requested data structure is posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.
2. Verify that the controller identifier of the controller is contained within the Controller List returned by the controller.
3. Verify that the Controller List contains valid values and that unused entries in the Controller List are zero filled.

**Case 4: Common Namespace Data Structure**

**Test Procedure:**
1. Configure the NVMe Host to issue an Identify command specifying CNS value 00h and CDW1.NSID value FFFFFFFFh to the controller in order to receive back an Identify Namespace data structure that specifies capabilities that are common across namespaces.

**Observable Results:**
1. Verify that the requested data structure is posted to the memory buffer indicated in PRP Entry 1, PRP Entry 2, and Command Dword 10, and that a command completion queue entry is posted to the Admin Completion Queue.

**Possible Problems:** None.
Test 9.2 – Namespace Management Command

**Purpose:** To determine if an NVMe Controller properly implements the Namespace Management Command.

**References:**
NVMe Specification 8.11

**Resource Requirements:**
Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** November 23, 2015

**Discussion:** The Namespace Management command is used for managing namespaces. It can be used for creating and deleting namespaces. The Select (SEL) field of Command Dword 10 is used to specify the type of operation for the Namespace Management Command.

The Namespace Management command uses the PRP Entry 1, PRP Entry 2, and Dword 10 fields. All other command specific fields are reserved.

**Case 1: Namespace Creation**

The create operation creates a new namespace. The new namespace will not be attached to any controller. The Namespace Attachment command must be used to attach a new namespace to a controller if desired.

The data structure used for the created operation is defined in Figure 102 and has the same format as the Identify Namespace data structure. However, the host is only allowed to set the following fields in the data structure:

* Namespace Size (NSZE)
* Namespace Capacity (NCAP)
* Formatted LBA Size (FLBAS)
* End-to-end Data Protection Type Settings (DPS)
* Namespace Multi-path I/O and Namespace Sharing Capabilities (NMIC)

All other fields are reserved and shall be cleared to 0.

For the creation operation, the CDW1.NSID field is reserved and shall be cleared to 0. The controller selects the next available namespace identifier to use for the new namespace. The namespace identifier of the new namespace is returned in Dword 0 of the completion queue entry for the command.

If creation of a new namespace would cause the number of namespace to exceed the number of supported namespaces, then the controller shall return status Namespace Identifier Unavailable.

If the size of the new namespace exceeds the amount of available space on the device, then the controller shall return status Namespace Insufficient Capacity and the Command Specific Information field of the Error Information Log specifies the total amount of NVM capacity required to create the namespace in bytes.

**Case 2: Namespace Deletion**

The delete operation deletes an existing namespace.

As a side effect of the delete operation, the namespace is detached from any controller as it is no longer present in the system. Namespaces detached due to a delete operation will become an inactive namespace.

There is no data structure transferred for the delete operation.
The CDW1.NSID field specifies the namespace to delete.

**Test Setup:** See Appendix A.

**Case 1: Namespace Creation**

**Test Procedure:**
1. Configure the NVMe Host to issue a Namespace Management command specifying Select field value 0h (Create) and valid values for the attached data structure to the controller in order to create a new namespace.
2. Configure the NVMe Host to issue an Identify command specifying CNS field value 11h and CDW1.NSID value set to the namespace identifier of the newly created namespace to the controller in order to receive back the Identify Namespace data structure for that namespace.
3. Configure the NVMe Host to issue Namespace Management commands specifying Select field value 0h (Create) and valid values for the attached data structure to the controller until the number of namespaces exceeds the number of namespaces supported by the NVM subsystem or the amount of available space on the device is exceeded.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that a new namespace is created which is not attached to any controller and that the capabilities returned in the Identify Namespace data structure match the appropriate values set using the Namespace Management command.
3. Verify that the Namespace Identifier Unavailable status is returned if the number of namespaces would exceed the number of supported namespaces or Namespace Insufficient Capacity if the size of a new namespace would exceed the available space on the device.

**Case 2: Namespace Deletion**

**Test Procedure:**
1. Configure the NVMe Host to issue a Namespace Management command specifying Select field value 1h (Delete) and CDW1.NSID field set to an active namespace attached to the controller in order to delete the specified namespace.
2. Configure the NVMe Host to issue an Identify command specifying CNS field value 02h and CDW1.NSID value 00h to the controller in order to receive back a Namespace List containing active namespace IDs attached to the controller.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the deleted namespace is not contained within the Namespace List.

**Possible Problems:** None.
Test 9.3 – Namespace Attachment Command

**Purpose:** To determine if an NVMe Controller properly implements the Namespace Attachment Command.

**References:**
- NVMe Specification 8.11

**Resource Requirements:**
- Tools capable of monitoring and decoding traffic on the NVMe interface.

**Last Modification:** November 23, 2015

**Discussion:** The Namespace Attachment command is used to attach and detach controllers from a namespace.

The Select field of command Dword 10 of the Namespace Attachment command is used to specify the operation of the command (attach or detach). The data structure used in the command is a 4096 byte Controller List which specifies the controllers that are to be attached or detached as part of the command. If the Controller List data structure transferred with the Namespace Attachment command is not properly formatted or is otherwise invalid then the command shall return status Controller List Invalid.

The Namespace Attachment command uses the Command Dword 10 field. All other command specific fields are reserved.

**Case 1: Namespace Attachment**

If a Namespace Attachment command is issued with the Controller Attach action selected and the namespace is already attached to a controller specified in the Controller List data structure transferred with the command, then the command shall return status Namespace Already Attached.

If a Namespace Attachment command is issued with the Controller Attach action selected and the namespace is private and already attached to another controller, then the namespace shall not be attached to any of the controllers specified in the Controller List data structure transferred with the command and the command shall return status Namespace Is Private.

**Case 2: Namespace Detachment**

When a namespace is detached from a controller it becomes an inactive namespace on that controller. Previously submitted but uncompleted or subsequently submitted commands to the affected namespace are handled by the controller as if they were issued to an inactive namespace.

If a Namespace Attachment command is issued with the Controller Detach action selected and the namespace is not attached to a controller specified in the Controller List data structure transferred with the command, then the command shall return status Namespace Not Attached.

**Test Setup:** See Appendix A.

**Case 1: Namespace Attachment**

**Test Procedure:**
1. Configure the NVMe Host to issue a Namespace Attachment command, specifying Select field value 0h (Attach), CDW0.NSID field value of an unattached namespace, and the controller identifier of the controller in the attached data structure, to the controller in order to attach the specified namespace to the controller.
2. Configure the NVMe Host to issue an Identify command specifying CNS field value 02h and CDW1.NSID value 00h to the controller in order to receive back a Namespace List containing active namespace IDs attached to the controller.
3. Configure the NVMe Host to issue a Namespace Attachment command, specifying Select field value 0h (Attach), CDW0.NSID field value of the same namespace used in the previous step, and the controller identifier of the controller in the attached data structure, to the controller.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the namespace attached to the controller via the Namespace Attachment command is contained within the returned Namespace List to signify that the namespace was successfully attached.
3. Verify that the status of the second Namespace Attachment command is Namespace Already Attached.

**Case 2: Namespace Detachment**

**Test Procedure:**
1. Configure the NVMe Host to issue a Namespace Attachment command, specifying Select field value 1h (Detach), CDW0.NSID field value of a namespace attached to the controller, and any controller identifier other than that of the controller in the attached data structure, to the controller in order to detach the specified namespace from the controller.
2. Configure the NVMe Host to issue a Namespace Detachment command, specifying Select field value 1h (Detach), CDW0.NSID field value of a namespace attached to the controller, and the controller identifier of the controller in the attached data structure, to the controller in order to detach the specified namespace from the controller.
3. Configure the NVMe Host to issue an Identify command specifying CNS field value 02h and CDW1.NSID value 00h to the controller in order to receive back a Namespace List containing active namespace IDs attached to the controller.

**Observable Results:**
1. Verify that after the completion of the command, the controller posts a completion queue entry to the appropriate Completion Queue indicating the status for the command.
2. Verify that the status of the first Namespace Attachment command is Namespace Not Attached.
3. Verify that the namespace detached from the controller via the Namespace Attachment command is not contained within the returned Namespace List to signify that the namespace was successfully detached.

**Possible Problems:** None.
Appendix A: DEFAULT TEST SETUP

Except where otherwise specified, all tests will require the DUT to have one of the following default physical configuration at the beginning of each test case:

Test Setup for NVMe Device:

Report status to user

NVMe Testing Station (acting as NVMe Host)  Connector (i.e. SFF-8639, PCIe CEM, NGFF)  NVMe Device Under Test (SSD)
Appendix B: NOTES ON TEST PROCEDURES

There are scenarios where in test procedures it is desirable to leave certain aspects of the testing procedure as general as possible. In these cases, the steps in the described test procedure may use placeholder values, or may intentionally use non–specific terminology, and the final determination of interpretation or choice of values is left to the discretion of the test technician. The following is an attempt to capture and describe all such instances used throughout the procedures.

Ports on Testing Station and Device Under Test

In general, any PCIe Port on the Testing Station or Device Under Test may be used as an interface with a test station or interoperability partner. There is assumed to be no difference in behavior, with respect to the protocols involved in this test suite, between any two PCIe ports on the Testing Station or Device Under Test. Hence, actual ports used may be chosen for convenience. However, it is recommended that the PCIe port used in the test configuration is recorded by the test technician.

Use of “various”

To maintain generality, some steps will specify that “various other values” (or the like) should be used in place of a given parameter. Ideally, all possible values would be tested in this case. However, limits on available time may constrain the ability of the test technician to attempt this. Given this, a subset of the set of applicable values must generally be used.

When deciding how many values should be used, it should be noted that the more values that are tested, the greater the confidence of the results obtained (although there is a diminishing return on this).

When deciding which specific values to use, it is generally recommended to choose them at pseudo–randomly yet deterministically. However, if there exist subsets of the applicable values with special significance, values from each subset should be attempted.
Appendix C: TEST TOOLS

The Tests described in this document can be performed using available PCIe and NVMe Test Tools. UNH–IOL maintains a reference list of tools in use at UNH–IOL for the execution of these tests at https://www.iol.unh.edu/services/testing/NVMe/tools.php.