

# UNH-IOL NVMe Test Consortium

**Test Suite for NVMe Interoperability**

*Version 6.0*

*Target Specification: NVMe 1.2*

*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 : August 30, 2016*

---

***UNH-IOL NVMe Consortium  
21 Madbury Rd, Suite 100  
Durham, NH 03824***

***Tel: +1 603-862-0090  
Fax: +1 603-862-4181  
Email: [nvmelab@iol.unh.edu](mailto:nvmelab@iol.unh.edu)***

---

## TABLE OF CONTENTS

|   |           |
|---|-----------|
| <b>MODIFICATION RECORD</b> .....  | <b>4</b>  |
| <b>ACKNOWLEDGMENTS</b> .....  | <b>7</b>  |
| <b>INTRODUCTION</b> .....   | <b>8</b>  |
| <b>REFERENCES</b> .....   | <b>10</b> |
| GROUP 1: OS BASED INTEROP TESTS .....   | 11        |
| Test 1.1 – Storage Devices Identified (Mandatory with 5/6 Interop Partners).....  | 12        |
| Test 1.2 – Format Storage Devices (Mandatory with 5/6 Interop Partners) .....   | 13        |
| Test 1.3 – Write Read Compare (Mandatory with 5/6 Interop Partners).....  | 14        |
| Test 1.4 – Hotplug NVMe Device (Mandatory for U.2 Devices with 1 Interop Partner, FYI for Hosts) .....                                | 16        |
| Test 1.5 – Boot from NVMe Device (Mandatory for Devices with 2 Interop Partners, Mandatory if Supported<br>for NVMe IP Devices) ..... | 17        |
| Test 1.6 – Dual Port Device (FYI) .....   | 19        |
| Test 1.7 – Dual Port Device with Multiple Namespaces (FYI) .....  | 22        |
| Test 1.8 – Dual Port Device with Single Namespaces (FYI).....   | 25        |
| Appendix A - Write/Read/Compare Utility for Windows and Linux .....   | 27        |
| Appendix B - Formatting Drive.....  | 29        |
| Appendix C - Write/Read/Compare Utility for UEFI .....  | 30        |
| Appendix D – Using Non-CEM Form Factors .....   | 32        |
| Appendix E - Non Integrated UEFI Boot from NVMe Device .....  | 33        |
| Appendix F – UNH-IOL Interop Test Bed.....  | 35        |
| Appendix G – NVMe Integrators List Requirements.....  | 36        |



## MODIFICATION RECORD

2012 May 7 (Version 0.1) Initial Release

David Woolf:

2012 June 21 (Version 0.2)

Raju Mishra:

2012 November 5 (Version 0.3)

David Woolf: Editorial Fixes

2013 April 30 (Version 0.4)

David Woolf: Preparation for plugfest, reorganization of interop tests to be more OS focused.

2013 May 21 (Version 1.0)

David Woolf: Clarification of test steps refined during May 2013 NVMe Plugfest.

2013 September 10 (Version 1.1)

David Woolf: Changes to test 1.3 to allow for use of stressing data patterns and varying transfer sizes. Addition of tests 1.4, 1.5, 1.6.

2013 December 16 (Version 1.1 DRAFT)

David Woolf: Modified Test 1.5 to clarify what OS media will be used, and that the OS install will occur using the UEFI NVMe driver. Modified Tests 1.1 and 1.2 to clarify procedure for identifying and formatting a drive in Windows and Linux operating systems. Updated Appendix A for latest version of vdbench. Added Appendix B. Added Appendix C.

2013 December 19 (Version 1.1 DRAFT)

David Woolf: Modified Appendix A

2013 December 23 (Version 1.1 DRAFT)

David Woolf: Corrected link to vdbench parameter file in Appendix A. Modified the command to start vdbench to include the '-vr' modifier to cause all writes to be validated immediately.

2014 March 11 (Version 1.1)

David Woolf: Corrected link to vdbench parameter file in Appendix A to account for corner case discovered during February 2014 NVMe Plugfest.

2014 March 31 (Version 1.1)

David Woolf: Added notes to the Possible Problems section of Test 1.3, and Appendix A, to account for problems that may arise when testing devices that are less than 512 MB in storage capacity.

2014 April 7 (Version 1.1)

David Woolf: Added information to Appendix C and Test 1.5.

2014 July 10 (Version 1.1)

David Woolf: Added test 1.7

2014 July 14 (Version 1.1b)

David Woolf: Added Appendix D. Added note to all tests to refer to Appendix D if using a non-CEM form factor. Renamed document to version 1.1b, to match latest NVMe specification and the IL policy to be used for the next plugfest.

2014 August 21 (Version 1.1b)

David Woolf: Edited Appendix A, and test 1.2 and 1.3 to clarify that an NVMe device should be unformatted when performing Test 1.3.

2014 August 25 (Version 1.1b)

David Woolf: Further clarifications to test 1.3 and 1.4 to clarify that an NVMe device should be unformatted when performing these tests.

2014 September 18 (Version 1.1b)

David Woolf: Clarifications to test 1.6 (hotplug) procedure and observable results.

2014 October 30 (Version 1.1b)

David Woolf: Clarifications to Appendix A and B on preparing drives for testing.

2014 December 9 (Version 1.1b)

David Woolf: Corrected URL for vdbench parameter file downloads

2015 April 7 (Version 1.2)

David Woolf: Added additional adapter types to Appendix D.

2015 May 7 (Version 1.2)

David Woolf: Updated links to VDBENCH Parameter files in Appendix A.

2015 August 31 (Version 1.2)

David Woolf: Clarified procedure in tests 1.3 and 1.5.

2015 September 28 (Version 1.2)

David Woolf: Clarified procedure in tests 1.6.

2015 November 5 (Version 1.2.1)

David Woolf: Test 1.5 now mandatory for hosts. References in *Introduction* and *References* sections now refer to NVMe 1.2 specification. Clarified step 6 in test procedure for test 1.3, to show that any partition information on the drive should survive the host restart.

2015 November 19 (Version 1.2.1)

Jeff Hensel: Updates to procedure for Test 1.5 reflect booting from an NVMe device using UEFI in a more modern motherboard than the previous procedure.

2015 December 11 (Version 1.2.1)

David Woolf: Updates to Test 1.5 to reflect that this test is not mandatory for NVMe IP Devices. Updated UNH-IOL address.

2016 January 19 (Version 1.2.1)

David Woolf: Updates to accommodate dual port testing in tests 1.1, 1.2, 1.6, 1.7.

2016 March 1 (Version 1.2.1)

Mike Bogochow: Fixed typos, clarified language, added links.

2016 May 19 (Version 6.0 r01)

David Woolf: Adopted new document numbering scheme. Removed test 1.4 ‘Multiple Devices on Bus’. Aligned procedures across all tests to use 2 devices at once and test them simultaneously in parallel.

2016 May 24 (Version 6.0 r02)

David Woolf: Added FYI tags to tests that are not mandatory. Added Appendix F with link to UNH-IOL Interop Test bed page, to help readers understand what hosts may be available for testing. Added Appendix G to outline NVMe Integrators List Requirements.

2016 June 7 (Version 6.0 r03)

David Woolf: Added test “Dual Port Device with Multiple Namespaces”

2016 June 13 (Version 6.0 r04)

David Woolf: Edited Test Procedure for test 1.7. Moved comments on Dual Port Device test setup from the Possible Problems section to the Test Setup section in tests 1.1, 1.2, 1.3, 1.4.

2016 June 16 (Version 6.0 r05)

David Woolf: Edited Test Procedure for Test 1.7 Procedure step 12a, to indicate that any values provided regarding sector size and namespace size are provided as examples only. These values can be adjusted according to device and driver support. Added complete version of Test 1.8.

2016 Jul y 5 (Version 6.0 r06)

David Woolf: Edited Test Procedure for Test 1.2, and Appendix G, to allow, but not require, the use of multiple LBA formats which are supported by the DUT. Added FYI step to Test 1.4, to check that IO is functional after the hotplug event occurs.

2016 Jul y 5 (Version 6.0 r07)

David Woolf: Edited Test names to include Mandatory vs. FYI requirements. Updated Appendix G to be consistent with how Mandatory vs. FYI requirements are indicated in the Conformance Test Suite.

2016 August 30 (Version 6.0)

David Woolf: Final version published to UNH-IOL site ahead of October 2016 NVMe Plugfest #6.

## **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

## 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 interoperates with other products designed to 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)
- 2.** PCI Express Base Specification Revision 3.0 (November 10, 2010)

### **Group 1: OS Based Interop Tests**

**Overview:** This section describes a method for performing interoperability verification for NVMe products using features readily available in the most common operating systems.

**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 – Storage Devices Identified (Mandatory with 5/6 Interop Partners)

**Purpose:** To verify that an NVMe Host can properly identify an attached NVMe Storage Device.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.  
If using a non-CEM form factor, see Appendix D

**Last Modification:** July 14, 2016

**Discussion:** NVMe Hosts should be able to identify all attached NVMe Devices and display these in the operating system.

**Test Setup:** All products are powered off. Two NVMe devices are physically attached to the NVMe Host Platform. If the NVMe devices are dual-port devices, the test should be performed using a port isolator to ensure that the Host communicates to Device 0 on Port 0 only, and that the Host communicates to Device 1 on Port 1 only.

**Test Procedure:**

1. Power on the NVMe Host and allow the OS to boot and all necessary drivers to be loaded.
  - a. In Windows based systems the attached NVMe device should be visible under Control Panel > Administrative Tools > Computer Management > Disk Management.
  - b. In Linux based systems the ‘lspci’ command can be used to list all attached PCI devices, followed by ‘ls /dev/nvme\*’.

**Observable Results:**

1. Verify that all NVMe devices are identified by the Host and can be displayed to the user through the operating system.

**Possible Problems:** Methods for performing this test may vary across different operating systems and drivers.

## Test 1.2 – Format Storage Devices (Mandatory with 5/6 Interop Partners)

**Purpose:** To verify that an NVMe Host can properly format an attached NVMe Storage Device.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.  
If using a non-CEM form factor, see Appendix D

**Last Modification:** July 14, 2016

**Discussion:** NVMe Hosts should be able to format all attached NVMe Devices and display these in the operating system.

**Test Setup:** Two NVMe devices are physically attached to the NVMe Host Platform, have been identified by the Host Platform, and are visible to the user through the operating system. If the NVMe Device Under Test is a dual-port device, the test should be performed using a port isolator to ensure that the Host communicates to Device 0 on Port 0 only, and that the Host communicates to Device 1 on Port 1 only.

**Test Procedure:**

1. Use available operating system tools to format all attached NVMe Devices. See Appendix B for how to do this in Windows and Linux operating systems. If possible perform this test with all LBA formats supported by the DUT.
2. Delete the partition created when the NVMe Devices were formatted.

**Observable Results:**

1. Verify that each NVMe Device can be partitioned and formatted, and the partitions can be deleted.

**Possible Problems:** Methods for performing this test may vary across different operating systems and drivers. Be sure to delete the created partition when the test is complete. It is allowable to perform this test with multiple LBA formats supported by the DUT, however only 1 LBA format is required to meet the test requirements.

### Test 1.3 – Write Read Compare (Mandatory with 5/6 Interop Partners)

**Purpose:** To verify that an NVMe Host can write and read to an attached NVMe Storage Device without error.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.  
If using a non-CEM form factor, see Appendix D

**Last Modification:** July 14, 2016

**Discussion:** NVMe Hosts should be able to write and read files to all attached NVMe Devices without error.

**Test Setup:** Two NVMe devices are physically attached to the NVMe Host Platform, have been identified by the Host Platform, and are visible to the user through the operating system. If the NVMe Device Under Test is a dual-port device, the test should be performed using a port isolator to ensure that the Host communicates to Device 0 on Port 0 only, and that the Host communicates to Device 1 on Port 1 only..

**Test Procedure:**

1. Remove or delete any formatting on the NVMe devices.
2. Use available tools to perform write/read/compare data operations containing a stressing data pattern to each attached NVMe Device for 5 minutes. The data operations should be of varying transfer sizes. See Appendix A.
3. Shutdown the Host Platform. Power on the Host Platform.
4. Repeat steps 1, 2 and 3, 3 times. Ensure that any partition information on the drive survives the shut-down.
5. Restart the Host Platform.
6. Repeat steps 1, 2 and 5, 3 times. Ensure that any partition information on all attached drives survives the restart.

**Observable Results:**

1. Verify that for all cases, the write/read/compare operations complete without error to all attached NVMe Devices.
2. Verify that the steps in the test procedure are performed correctly, and there are no errors reported during the performing of the tests.

**Possible Problems:** Methods for performing this test may vary across different operating systems and drivers. If the DUT has a storage capacity less than 512MB, it will be necessary to modify the threads parameter in the vdbench parameter file. Change the ‘threads=128’ to ‘threads=16’. See Appendix A for more detail on the use of vdbench.



### Test 1.4 – Hotplug NVMe Device (Mandatory for U.2 Devices with 1 Interop Partner, FYI for Hosts)

**Purpose:** To verify that an NVMe Host properly handles hotplugging of an NVMe Storage Device.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device supporting U.2/SFF-8639 form factor.

**Last Modification:** July 5, 2016

**Discussion:** NVMe Devices should be able to be hotplugged from/to an NVMe Host without error. At this time, this test is not used to determine qualification of products for the NVMe Integrator’s List. This test is considered optional, and for information only.

**Test Setup:** Two NVMe devices are physically attached to the NVMe Host Platform, have been identified by the Host Platform, and are visible to the user through the operating system. If the NVMe Device Under Test is a dual-port device, the test should be performed using a port isolator to ensure that the Host communicates to Device 0 on Port 0 only, and that the Host communicates to Device 1 on Port 1 only.

**Test Procedure:**

1. Power on the NVMe Host and allow the operating system (OS) to boot and all necessary drivers to be loaded.
2. Ensure that the OS has not loaded from the NVMe Device Under Test, but has loaded from another source.
3. Ensure that the NVMe Device can be accessed from the OS, and that IO can be performed.
4. Physically remove and reinsert each device from the NVMe Host, as follows:
  - a. 10 cycles of removal and reinsertion, with 25 ms removal/reinsertion time
  - b. 10 cycles of removal and reinsertion, with 10 ms removal/reinsertion time
  - c. 10 cycles of removal and reinsertion, with 100 ms removal/reinsertion time
  - d. 10 cycles of removal and reinsertion, with 500 ms removal/reinsertion time
5. Ensure that the NVMe Device can still be accessed from the OS, and that IO still can be performed.

**Observable Results:**

1. After each removal, verify that the removed device is no longer visible in the OS.
2. After each reinsertion, verify that the Host is able to access the NVMe device.
3. Ensure that after all steps, the Host still exhibits normal operation.
4. Ensure that after the hotplug event, IO operations are still functional. This observable result step can be considered FYI.

**Possible Problems:** Earlier versions of Linux Kernels have been observed to not enumerate NVMe devices properly after a hotplug event. If problems like this are observed, it is recommended to upgrade to the latest Linux Kernel.



**Test 1.5 – Boot from NVMe Device (Mandatory for Devices with 2 Interop Partners, Mandatory if Supported for NVMe IP Devices)**

**Purpose:** To verify that an NVMe Host can boot from an attached NVMe Storage Device.

**References:**

- [1] NVMe Specification

**Resource Requirements:**

- NVMe Host Platform and Device.
- If using a non-CEM form factor, see Appendix D

**Last Modification:** July 14, 2016

**Discussion:** NVMe Hosts should be able to boot from an attached NVMe Device without error. This test is only applicable to a single port NVMe Device.

**Test Setup:** A single port NVMe Device is physically attached to the NVMe Host Platform. The Host platform will use the UEFI NVMe Driver to boot from the NVMe Device.

**Test Procedure:** The procedure below describes installing Windows Server 2012 R2 on an NVMe host platform with UEFI support. The procedure will vary for different operating systems. See **Appendix E** for an example of performing this test in a host system without integrated UEFI support. The system used below is an **ASRock Z97 Extreme 6 with BIOS Revision 2.3**. Other host platforms are acceptable, but the procedure may vary.

1. Choose the method of Install:
  - a. Windows 8 bootable USB
  - b. Windows 8 bootable installation disk
2. Install NVMe device
3. Install selected installation media
4. On boot, launch the Boot Menu. This can be done by pressing the “F11” key on boot
5. Select the option that begins with “UEFI: “; if using a disk press any key after to boot
6. Once the Windows installer launches press “Next”
7. Select “Install Now”
8. Check the “I Accept...” box below the licensing scroll box
9. Select “Next”
10. Select “Custom: Install Windows Only (advanced)”
11. Select the NVMe Device and select “Next”
12. Ensure the device finishes installation without error
13. Remove the installation media and ensure the device is able to boot properly

**Observable Results:**

1. Verify that the operating system booted as expected, the NVMe device was identified properly by the operating system for each power down or reboot cycle.

**Possible Problems:** Methods for performing this test may vary across different operating systems and drivers. Not all operating systems may have boot support for NVMe. Additionally different hardware platforms may behave differently, with slightly different steps necessary. Methods that vary from the procedure outlined here on different host platforms are acceptable.

This test may not be applicable or possible to perform for certain types of NVMe IP Devices, such as those implemented on an FPGA platform. Therefore, for an NVMe IP Device, this test is not considered a required test for the NVMe Integrator’s List.



### Test 1.6 – Dual Port Device (FYI)

**Purpose:** To verify that an NVMe Host properly handles a NVMe Storage Device with Dual Ports.

**References:**

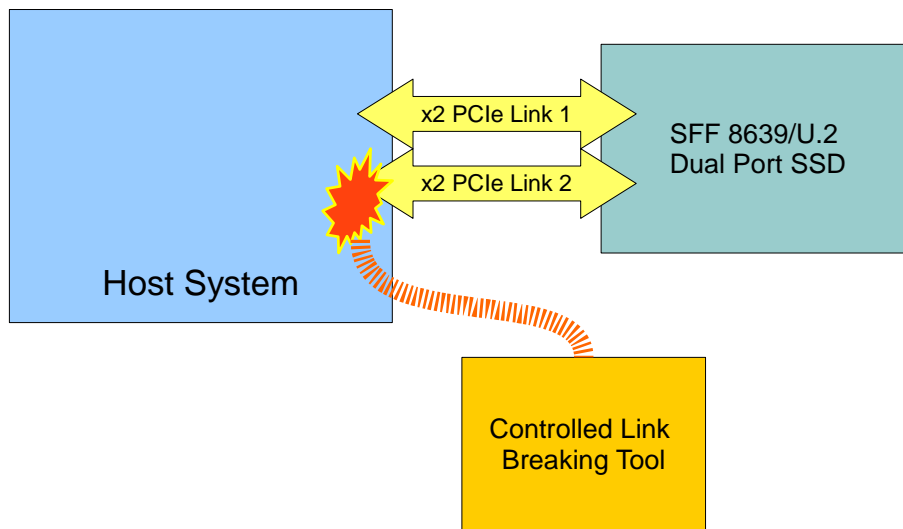
[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.

**Last Modification:** March 1, 2016

**Discussion:** Dual ported NVMe Devices provide redundancy in the face of link breakages or port failures. This test only applies to SFF-8639/U.2 devices that support dual ports. A controlled method must be used to break and repair links in this test, as shown in the diagram below.



**Test Setup:** A single NVMe dual ported device is physically attached to the NVMe Host Platform, has been formatted by the Host Platform, and is visible to the user through the operating system (OS).

**Test Procedure:**

1. Power on the NVMe Host and allow the OS to boot and all necessary drivers to be loaded.
2. Ensure that the OS has not loaded from the NVMe Device Under Test, but has loaded from another source.
3. Using the appropriate OS utilities, determine if the dual port NVMe Device appears twice in the OS, or, if some method of MPIO is enabled, the NVMe Device appears only once.
4. Using a controlled method, break one link to the NVMe Device while the other link remains connected and active.
5. Using a controlled method, reconnect the broken link to the NVMe Device while the other link remains connected and active.

**Observable Results:**

1. After step 4: Using the appropriate OS utilities, determine the following:
  - a. If the dual port NVMe Device appeared twice in the OS in step 3, now only 1 instance of the NVMe Device is visible in the Host OS.
  - b. If some method of MPIO is enabled, the NVMe device still appears in the Host OS.

2. After step 5: Using the appropriate OS utilities determine the following:
  - a. If the dual port NVMe Device appeared twice in the OS in step 3, there are now again 2 instances of the NVMe Device visible in the Host OS.
  - b. If some method of MPIO is enabled, the NVMe device still appears in the Host OS.

**Possible Problems:** Methods for disabling the NVMe Device may vary across different operating systems.



### Test 1.7 – Dual Port Device with Multiple Namespaces (FYI)

**Purpose:** To verify that an NVMe Host properly handles a NVMe Storage Device with Dual Ports which also supports multiple namespaces

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.

**Last Modification:** June 16, 2016

**Discussion:** Dual ported NVMe Devices provide redundancy in the face of link breakages or port failures. This test only applies to SFF-8639/U.2 devices that support dual ports.

**Test Setup:** One NVMe dual ported device is physically attached to the NVMe Host Platform by both ports – NVMe controller IDs 0 and 1. This can be accomplished through the use of cables & retimers, passive PCIe splitter board, or PCIe switches as available/required.

**Test Procedure:**

1. Power on the NVMe Host and allow the OS to boot and all necessary drivers to be loaded.
2. Ensure that the OS has *not* loaded from the NVMe Device Under Test, but has loaded from another source.
3. Check for PCIe devices (PCIe Link) from both controller ID 0 and 1.
  - a. Linux: `lspci | grep Non`
  - b. Windows: Device Manager > Storage Controllers > Mass Storage Device
4. Check for NVMe devices (NVMe Driver Loaded) and namespaces within those devices
  - a. Linux: `ls -al /dev/nvm*` or `lsblk`
  - b. Windows: Device Manager > Disk Drives
5. Detach/Delete Namespaces. The following steps describe how to clear the SSD of all existing namespaces.
6. Identify existing Namespaces.
  - a. In Linux using `nvme-cli` tools: `nvme list-ns /dev/nvmeX` - typically `/dev/nvme0` and `/dev/nvme1`
  - b. In Windows: `issdcm` or other tool > syntax TBD.
7. Detach Existing Namespaces
  - a. In Linux, using `nvme-cli` tools:
    - i. `nvme detach-ns /dev/nvmeX -n X` - repeat for additional namespaces
  - b. In Windows: `issdcm` or other tool > syntax TB
8. Delete Existing Namespaces
  - a. In Linux using `nvme-cli` tools:
    - i. `nvme delete-ns /dev/nvmeX -n X` - repeat for additional namespaces
  - b. In Windows: `issdcm` or other tool > syntax TB
9. Issue NVMe subsystem reset (NSSR) or Controller Reset, and Rescan or Reboot Host
  - a. In Linux: Prior to 4.4 kernel, reboot Host Platform
  - b. In Linux: 4.4+ kernel
    - i. `modprobe -r nvme, sudo sh -c "echo 1 > /sys/bus/pci/rescan", modprobe nvme`
  - c. In Windows: Reboot or disable/enable devices in Device Manager
10. Verify Namespaces Deleted
  - a. In Linux, using `nvme-cli` tools:
    - i. `nvme list-ns /dev/nvmeX` (typically `/dev/nvme0` and `/dev/nvme1`)
    - ii. Can also be accomplished with: `ls -l /dev/nvm*` or `lsblk` – look for absence of `/dev/nvmeXnY`, only controllers present
  - b. In Windows: `issdcm` or other tool > syntax TBD
11. Create/Attach Namespaces
  - a. For IOL multi-namespace testing the SSD should have a two namespaces, one attached to controller 0 and one attached to controller 1

12. Create Namespaces – Note: FLBA Size (logical block size) will vary from device to device, check available formats
  - a. First Namespace: In Linux, using nvme-cli tools execute the following. Note that any values provided regarding sector size and namespace size are provided as examples only. These values can be adjusted according to device and driver support:
    - i. `nvme create-ns /dev/nvmeX -s <sizeInBlocks> -c <capInBlocks> -f <FLBA Size> -d <dataProtection> -m 1 <sharing on>`
    - ii. i.e. `nvme create-ns /dev/nvme0 -s 52428800 -c 52428800 -f 2 -d 0 -m 1` creates a 200GB namespace on controller 0 based on 4k sectors -f 2 (4096b in this controller), no data protection information -d 0, and sharing enabled so that both controllers can attach to this namespace -m 1
    - iii. Similarly `nvme create-ns /dev/nvme0 -s 419430400 -c 419430400 -f 0 -d 0 -m 1` creates a similar 200GB namespace based on 512b sectors -f 0
  - b. Second Namespace: In Linux using nvme-cli tools:
    - i. `nvme create-ns /dev/nvmeX -s <sizeInBlocks> -c <capInBlocks> -f <FLBA Size> -d <dataProtection> -m 1 <sharing on>`
    - ii. This creates a 2nd namespace, preferably of the same capacity as the 1st with the same -f <num> FLBA Size.
13. Attach Namespaces
  - a. In Linux using nvme-cli tools: `nvme attach-ns -n <namespace> /dev/nvmeX`
    - i. i.e. `nvme attach-ns /dev/nvme0 -n1 -c0` attaches ns 1 to nvme controller /dev/nvme0
    - ii. i.e. `nvme attach-ns /dev/nvme1 -n2 -c1` attaches ns 2 to nvme controller /dev/nvme1
  - b. In Windows: `issdcm` or other tool > syntax TB
14. Issue NVMe subsystem reset device and Rescan or Reboot
  - a. In Linux: Prior to 4.4 kernel, reboot Host Platform
  - b. In Linux: 4.4+ kernel
    - i. `modprobe -r nvme, sudo sh -c "echo 1 > /sys/bus/pci/rescan", modprobe nvme`
  - c. In Windows: Reboot or disable/enable devices in Device Manager
15. Verify Namespaces Attached
  - a. In Linux, using nvme-cli tools:
    - i. `nvme list-ns /dev/nvmeX` (typically /dev/nvme0 and /dev/nvme1)
    - ii. Can also be accomplished with: `ls -l /dev/nvm*` or `lsblk` – look for presence of /dev/nvmeXnY, both controllers present and namespaces should be present
  - b. In Windows: `issdcm` or other tool > syntax TBD
16. Using appropriate OS utilities, mount and format both instances of the NVMe device.
  - a. In Linux: Format the drives with extX/xfs/other file system and assign a mount point.
  - b. In Windows: Format the drives with NTFS and assign a drive letter.
17. Verify Namespaces are independent of each other
  - a. Copy any unique file to the mount point of /dev/nvme0n1 (text file w/dev-nvme0n1)
  - b. List the directory contents of the mount of /dev/nvme1n2 – verify the copied file is not there
  - c. Copy any unique file to the mount point of /dev/nvme1n2 (text file w/dev-nvme0n2)
  - d. List the directory contents of the mount of /dev/nvme0n1 – verify the copied file is not there
18. Using appropriate OS utilities, cease all IO to the drive, and unmount.
  - a. In Linux: Unmount the devices from their mount points.
  - b. In Windows: Remove the assigned drive letters.
19. Detach Existing Namespaces
  - a. In Linux, using nvme-cli tools:
    - i. `nvme detach-ns /dev/nvmeX -n X` - repeat for additional namespaces
  - b. In Windows: `issdcm` or other tool > syntax TB
20. Attach Namespaces To Alternate Ports
  - a. In Linux using nvme-cli tools: `nvme attach-ns -n <namespace> /dev/nvmeX`
    - i. i.e. `nvme attach-ns /dev/nvme0 -n2 -c0` attaches ns 2 to nvme controller /dev/nvme0
    - ii. i.e. `nvme attach-ns /dev/nvme1 -n1 -c1` attaches ns 1 to nvme controller /dev/nvme1
21. Issue NVMe subsystem reset device and Rescan or Reboot
  - a. In Linux: Prior to 4.4 kernel, reboot Host Platform

- b. In Linux: 4.4+ kernel
    - i. `modprobe -r nvme, echo 1 > /sys/bus/pci/rescan, modprobe nvme`
  - c. In Windows: Reboot or disable/enable devices in Device Manager
22. Verify Namespaces Attached
- a. In Linux, using `nvme-cli` tools:
    - i. `nvme list-ns /dev/nvmeX` (typically `/dev/nvme0` and `/dev/nvme1`)
    - ii. Can also be accomplished with: `ls -l /dev/nvm*` or `lsblk` – look for presence of `/dev/nvmeXnY`, both controllers present and namespaces should be present
  - b. In Windows: `issdcm` or other tool > syntax TBD
23. Using appropriate OS utilities, mount both instances of the NVMe device.
- a. In Linux: Assign a mount point.
  - b. In Windows: Assign a drive letter.
24. Using the appropriate OS utility, check the filesystem for errors and verify file written in step 17 can be read
- a. In Linux: `fsck` both namespaces and `cat` the written files
  - b. In Windows: `chkdsk` both the namespaces and open the written files
  - c. File system check should return no errors and files should be readable
25. Unmount devices, shutdown hosts.

**Note:** ‘Test 1.3 Write Read Compare’ can be accomplished between step 15-16 and 24-25 in this procedure with ½ of the IO going to `/dev/nvme0n1` and the other ½ going to `/dev/nvme1n2`. Performing the test in this fashion can save potentially save test time.

**Observable Results:**

1. After initial power on, verify the NVMe device appears as two distinct NVMe controllers in the appropriate OS utility.
2. Verify ability to detach & delete namespaces – Steps 7-10.
3. Verify ability to create & attach namespaces – Steps 11-15.
4. Verify namespaces can be formatted & mounted – Step 16.
5. Verify namespaces are independent of each other – Step 17.
6. Verify namespaces can be unmounted – Step 18.
7. Verify namespaces can be attached to alternate ports/controllers – Step 19-21.
8. Verify filesystem and files remain intact on controller change – Step 22-24.

**Possible Problems:** Methods for disabling the NVMe Device may vary across different OSs and Linux kernel versions. Open source tools for namespace manipulation may be unavailable in Windows OSs. NVMe subsystem-reset will be unavailable in Linux prior to 4.4 kernel and will be dependent on OEM Windows driver in Server 2012 r2 and Windows 8.1 OSs. In addition, the NVMe Linux driver writers made a change to the behavior of namespace enumeration between 4.4 and 4.5 kernels.

4.4 and previous kernels:

`/dev/nvme0n1` - Controller 0 – Namespace 1  
`/dev/nvme1n2` – Controller 1 – Namespace 2

4.5 and after kernels:

`/dev/nvme0n1` - Controller 0 – Namespace unknown (use `nvme id-ns` command to identify)  
`/dev/nvme1n1` – Controller 1 – Namespace unknown (use `nvme id-ns` command to identify)



## Test 1.8 – Dual Port Device with Single Namespaces (FYI)

**Purpose:** To verify that an NVMe Host properly handles an NVMe Storage Device with Dual Ports which supports only a single namespace

### References:

[1] NVMe Specification

### Resource Requirements:

NVMe Host Platform and Device.

**Last Modification:** June 16, 2016

**Discussion:** Dual ported NVMe Devices provide redundancy in the face of link breakages or port failures. This test only applies to SFF-8639/U.2 devices that support dual ports.

**Test Setup:** One NVMe dual ported device is physically attached to the NVMe Host Platform by both ports – NVMe controller IDs 0 and 1. This can be accomplished through the use of cables & retimers, passive PCIe splitter board, or PCIe switches as available/required

### Test Procedure:

1. Power on the NVMe Host and allow the OS to boot and all necessary drivers to be loaded.
2. Ensure that the OS has *not* loaded from the NVMe Device Under Test, but has loaded from another source.
3. Check for PCIe devices (PCIe Link) from both controller ID 0 and 1
  - a. Linux: `lspci | grep Non`
  - b. Windows: Device Manager > Storage Controllers > Mass Storage Device
4. Check for NVMe devices (NVMe Driver Loaded) and single namespace within those devices
  - a. Linux: `ls -al /dev/nvm*` or `lsblk`
  - b. Windows: Device Manager > Disk Drives
5. Identify Existing Namespace (Single Namespace Device should have only 1)
  - a. In Linux, using `nvme-cli` tools:
    - i. `nvme list-ns /dev/nvmeX` - typically `/dev/nvme0` and `/dev/nvme1`
    - ii. `nvme list`
  - b. In Windows: `issdcm` or other tool > syntax TBD
6. Using appropriate OS utilities, Format and Mount *ONE* of the *TWO* NVMe devices.
  - a. In Linux: Format the drive `/dev/nvme0n1` with `extX/xfs/other` file system and assign a mount point.
  - b. In Windows: Format the drives with NTFS and assign a drive letter.
7. Verify Namespaces are self-similar (same LBA range)
  - a. Copy any unique file to the mount point of `/dev/nvme0n1` (text file w/`dev-nvme0n1`)
  - b. List the directory contents of the mount of `/dev/nvme0n1` – verify the copied file is there and readable
  - c. Unmount `/dev/nvme0n1` and then Mount `/dev/nvme1n1` to same mount point
  - d. Using the appropriate OS utility, check the filesystem for errors and verify file written in step 7a can be read
    - i. In Linux: `fsck /dev/nvme1n1` and `cat` the written files
    - ii. In Windows: `chkdsk` both the namespaces and open the *written* files
    - iii. File system check should return no errors and files should be readable
  - e. List the directory contents of the mount of `/dev/nvme1n1` – verify the copied file is there and readable
  - f. Modify the copied file in `/dev/nvme1n1` – change text in file & save
  - g. List the directory contents of the mount of `/dev/nvme1n1` – verify the copied file is there and readable
  - h. Unmount `/dev/nvme1n1` and then Mount `/dev/nvme0n1` to same mount point

- i. Using the appropriate OS utility, check the filesystem for errors and verify file written in step 7a can be read and change in file is present
    - i. In Linux: *fsck* both namespaces and *cat* the written files
    - ii. In Windows: *chkdsk* both the namespaces and open the *written* files
    - iii. File system check should return no errors and files should be readable
  - j. List the directory contents of the mount of /dev/nvme0n1 – verify the copied file is there & changes to file in step 7e are present
8. Unmount devices, shutdown hosts, end of test...

**Note:** ‘Test 1.3 Write Read Compare’ can be accomplished after step 8 in this procedure with ½ of the IO going to /dev/nvme0n1 and the other ½ going to /dev/nvme1n2. Performing the test in this fashion can save potentially save test time.

**Observable Results:**

1. After initial power on, verify the NVMe device appears as two distinct NVMe controllers in the appropriate OS utility.
2. Verify single namespace attached to both controllers
3. Verify ability to format & mount namespace from controller 0 – Step 6
4. Verify ability to write & read file – Step 6a-6b
5. Verify ability to unmount and remount namespace using alternate controller – Step 6C
6. Verify filesystem not corrupted and written file present & intact – Step 6d-6f
7. Verify ability to unmount and remount namespace using original controller – Step 6h
8. Verify filesystem not corrupted and written file present & intact – Step 6i-6j
9. End of Test

**Possible Problems:** Methods for disabling the NVMe Device may vary across different OSs and Linux kernel versions. The NVMe Linux driver writers made a change to the behavior of namespace enumeration between 4.4 and 4.5 kernels.

4.4 and previous kernels:

/dev/nvme0n1 - Controller 0 – Namespace 1  
/dev/nvme1n2 – Controller 1 – Namespace 2

4.5 and after kernels:

/dev/nvme0n1 - Controller 0 – Namespace unknown (use *nvme id-ns* command to identify)  
/dev/nvme1n1 – Controller 1 – Namespace unknown (use *nvme id-ns* command to identify)

## Appendix A - Write/Read/Compare Utility for Windows and Linux

**Purpose:** To define a means of connecting prototype NVMe products for performing the tests outlined in this document.

### References:

[1] NVMe Specification

### Resource Requirements:

NVMe Host and Device, PCIe conformance channel

**Last Modification:** March 1, 2016

**Discussion:** To perform interoperability testing between an NVMe Host Platform and an NVMe Device, a utility capable of writing, reading, and comparing data written to an NVMe Device is necessary. This Appendix will describe the use of the VDBENCH utility to perform this testing. Being a Java based tool, VDBENCH has the benefit of working across multiple operating systems (OS). Other utilities may be used instead of VDBENCH, but their behavior should be known to be similar to that implemented with VDBENCH.

Stressing patterns can be used in the test by using a configuration file with contents that will produce a pattern on the PCIe bus when the file transfer occurs.

**Test Setup:** The NVMe device is physically attached to the NVMe Host Platform. The NVMe device has not been formatted by the Host Platform, all partitions have been deleted, and is visible to the user.

### Test Procedure:

1. Install vdbench 504, following the instructions at <http://www.oracle.com/technetwork/server-storage/vdbench-downloads-1901681.html>
2. Download the VDBENCH parameter file for your OS:
  - a. [https://www.iol.unh.edu/sites/default/files/testsuites/nvme/unh\\_interop\\_6\\_0\\_linux.txt](https://www.iol.unh.edu/sites/default/files/testsuites/nvme/unh_interop_6_0_linux.txt)
  - b. [https://www.iol.unh.edu/sites/default/files/testsuites/nvme/unh\\_interop\\_6\\_0\\_windows.txt](https://www.iol.unh.edu/sites/default/files/testsuites/nvme/unh_interop_6_0_windows.txt)
3. Verify that the attached NVMe Device is visible in the OS (disk management in Windows or lspci in Linux)
4. Follow the steps in Appendix B to mount the drive. If you are only performing the Write/Read/Verify test it is not necessary to format the drive. If using a Windows operating system be sure to delete any formatted partition on the drive before performing the Write/Read/Verify test.
5. Edit the provided VDBENCH parameter file so that the 'lun' parameter points to the SSD under test. If using a Windows operating system this should be "PhysicalDrive1" or similar. The number will change depending on how many drives are installed in the host platform. Use Disk Management found under Computer Management to get the Disk number for the drive intended for testing.
6. Run the provided VDBENCH parameter file using the following command:  
`vdbench -f <parameter file filename> -vr.`
  - a. If using a Windows operating system, be sure to run vdbench from an Administrator command prompt (Right click cmd icon, select 'Run as Administrator').
  - b. If using a Linux operating system, be sure to give vdbench the correct permissions using 'chmod 777 vdbench'. This is only necessary when running vdbench for the first time on a new system.
7. Verify that the vdbench test run completed with no errors in the /vdbench504/output/summary.html file.

### Observable Results:

1. Verify that the write/read/compare operations complete without error to the NVMe Device.

### Possible Problems:

1. Methods for performing this test may vary across different operating systems and drivers.
2. On some Linux operating systems it may be necessary to install Java and csh. This can be the source of a ‘bad interpreter’ error message when attempting to run vdbench.
3. If VDBENCH indicates a Java error, it may be necessary to update the path to Java in the vdbench.bat file (Windows). To do this, use the following steps:
  - a. Open vdbench.bat for editing.
  - b. Find where the Java path is in the bat file; by default it says: set java=java.
  - c. Find where Java is on the system. For example, in Windows 7 64 bit and Windows Server 2012 systems the default path is C:\Program Files (x86)\Java\jre7\bin\java.exe.
  - d. Copy the new path into the vdbench.bat file: set java="C:\Program Files (x86)\Java\jre7\bin\java.exe". Be sure to use the double quotes.
4. If the DUT has a storage capacity less than 512MB, it will be necessary to modify the threads parameter in the vdbench parameter file. Change the ‘threads=128’ to ‘threads=16’. If this adjustment is not made, the test may not complete, and vdbench may indicate a ‘busy’ or ‘timeout’ error.

## Appendix B - Formatting Drive

**Purpose:** To define the procedure for formatting a drive in Windows and Linux operating systems (OS).

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host and Device, PCIe conformance channel

**Last Modification:** March 1, 2016

**Discussion:** NVMe Hosts should be able to format all attached NVMe Devices and display these in the operating system. This Appendix will describe how to perform this operation using Windows and Linux based operating systems.

**Test Setup:** The NVMe Device is physically attached to the NVMe Host Platform and is visible to the user.

**Windows Procedure:**

1. Navigate to Control Panel, Administrative Tools, Computer Management, Disk Management.
2. If prompted to Initialize the disk select 'Ok'.
3. Right click on the volume to be formatted. Follow the prompts for a 'Quick Format'.

**Linux Procedure:**

1. `sudo fdisk /dev/nvme0n1`, then press 'w'
2. `sudo mkfs.ext2 /dev/nvme0n1` (the filename may vary depending on the device)
3. `sudo mkdir /mnt/nvme` (this step only necessary when setting up a system for the first time).
4. `sudo mount /dev/nvme0n1 /mnt/nvme`
5. Run vdbench tests as described in Appendix A to ensure that read and write operations can be performed correctly.

Note that steps 1 and 2 are only necessary when initially installing the drive. After reboot, only step 4 is necessary

**Observable Results:**

1. Verify that the write/read/compare operations complete without error to the NVMe Device.

**Possible Problems:**

Methods for performing this test may vary across different operating systems and drivers.

## Appendix C - Write/Read/Compare Utility for UEFI

**Purpose:** To define a means of connecting prototype NVMe products for performing the tests outlined in this document when using UEFI.

**Last Modification:** April 3, 2014

**Discussion:** An NVMe Driver exists for the UEFI boot shell. Below is a procedure for performing a Write/Read/Compare operation from a UEFI Host to an NVMe Device to match the requirements of Test 1.3 in this document.

**Test Setup:** The NVMe device is physically attached to the NVMe Host Platform.

**Test Procedure:** The Write/Read/Compare tests over UEFI can be run off of a USB stick in a system that can boot from USB. To prep a USB stick for testing, simply extract the IHV-SCT\_Binary (this is used for device testing) and the UEFI-SCT\_Binary (this is used for platform testing) as well as the compatible shell version.

Download the X64 version of the UEFI shell by going to [http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=UEFI\\_Shell](http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=UEFI_Shell)  
Navigate by clicking the links to ShellBinPkg -> UefiShell/ X64 -> Shell.efi

Later releases of the SCT test software may support newer versions of the shell. To check on versions of the SCT test software, visit <http://www.uefi.org/specs/access> which has a list of all releases. Follow the instructions on the site to get access to the new releases.

### **UEFI Test Guide for IHV-SCT Testing** (Note: the steps below may vary for different host platforms)

1. Load a USB drive with the correct folder of SCT testing and shell
2. Start the system and enter boot manger when prompted
3. Select:
  - a. “UEFI Boot Menu”
  - b. “Boot From File”
  - c. “Cruzer Glide” USB option
  - d. “Shellv1.efi”
4. Stop automatic startup
5. EFI might start in an incorrect drive
  - a. Type “cd ..” until at the root of the drive
  - b. Type “ls” and you should see the following:
    - 1.\*\*\*\*\*SUBJECT TO CHANGE\*\*\*\*\*
    - 2.UEFI-SCT\_Binary
    - 3.Shellv1.efi
    - 4.Other
    - 5.SCT
    - 6.Startup.nsh
  - c. If these sections are not seen:
    - 1.Type map -b and try each of the fsX: locations
    - 2.Change locations by typing “fsx:”
6. Now that you are in the correct location navigate to “\IHV-SCT\_Binary\IHVSct\SctPackageX64\X64”
7. Type “sct -u”
8. Enter “Test Device Configuration”
  - a. Type “s1”
  - b. It should be obvious which drive is NVMe by looking at the drivers, it doesn’t specifically say NVMe but through process of elimination only one should remain
  - c. To add the device to the list, type “I XX” where XX is the reference handle (example: C6)
  - d. Choose “P” for PCIe device
  - e. Choose “Y” for all for default configuration

9. Quit device configuration with “q”
10. Enter “Test Case Management”
  - a. Enter the IHV folder and activate the following;
    - 1.“DevicePathProtocolTest”
    - 2.“DriverModelTest”
    - 3.“BootableImageSupportTest”
  - b. Hit F9 to begin testing
11. Once complete choose the “Test Report Generator” and save the results
  - a. These can later be found in \IHV-SCT\_Binary\IHVSct\SctPackageX64\X64\Report

Notes: This method takes around 7 hours to run though the USB. A faster method involves transferring all test files to a hard drive, which is advised as this method only takes ~30min to run.

**Observable Results:**

1. Verify that the SCT tests completed without error.



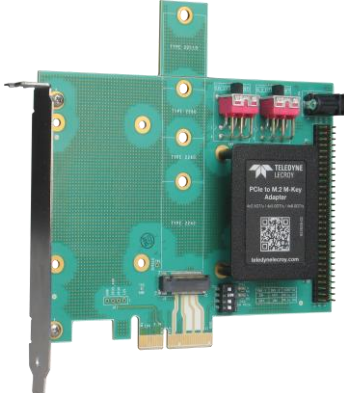
**Possible Problems:**

## Appendix D – Using Non-CEM Form Factors

**Purpose:** To define a means of connecting prototype NVMe products for performing the tests outlined in this document when using non-CEM Form Factors.

**Last Modification:** April 7, 2015

**Discussion:** An NVMe Drive may be implemented in a variety of form factors, including CEM, SFF-8639, and M.2. If a Host system supporting the needed drive form factor is not available, an adapter may be used to connect the drive and host for the purpose of protocol interoperability testing. Examples of such adapters are provided below for reference only. Adapters with similar functionality and specifications are acceptable.

|  |  |
|--|--|
| <p>Teledyne-LeCroy SFF-8639 Port A Adapter</p>  A green printed circuit board (PCB) adapter. It features a large black heat sink in the center, a white SATA data connector on the right, and a gold-plated SFF-8639 connector at the bottom. The PCB has various electronic components and a small white label with the number '5114-0001'.   | <p>Serial Cables PCI-AD-X439-01 Adapter</p>  A red PCB adapter with a silver metal mounting bracket. It has a gold-plated PCI edge connector at the bottom and a SATA data connector on the right. The board is populated with several electronic components. |
| <p>Teledyne-LeCroy M.2 M-Key Adapter</p>  A green PCB adapter with a black metal mounting bracket. It features a gold-plated M.2 M-Key connector at the bottom and a SATA data connector on the right. A black M.2 drive is mounted on the board. The PCB has several gold-plated test points and a small white label with a QR code and the text 'TELEDYNE LECROY PCI-to-M.2 M-Key Adapter'. |  |



## Appendix E - Non Integrated UEFI Boot from NVMe Device

**Purpose:** To verify that an NVMe Host can boot from an attached NVMe Storage Device.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.  
If using a non-CEM form factor, see Appendix D

**Last Modification:** November 19, 2015

**Discussion:** NVMe Hosts should be able to boot from an attached NVMe Device without error.

**Test Setup:** The NVMe Device is physically attached to the NVMe Host Platform. The Host Platform will use the UEFI NVMe Driver to boot from the NVMe device.

**Test Procedure:** The procedure below describes installing Windows Server 2012 R2 on an NVMe host platform with UEFI support. The procedure will vary for different operating systems. The system used was a **Dell PowerEdge R720 with BIOS Revision 2.2.2**. Other host platforms are acceptable.

1. Prepare a USB key to store the UEFI shell by formatting the key as FAT32 and then copy the Shell.efi to the root directory on the key. Shell.efi can be obtained from UEFI.org. Also copy NvmExpressDxe.x64.efi onto root of the thumb drive.
2. Power off system
3. Attach NVMe Device to the Host Platform.
4. Insert UEFI shell thumb drive in a USB port
5. Power on system
6. Insert Windows installation disc in optical drive.
7. Press function key to start UEFI boot manager (for example on Dell systems this is usually the 'F11' key)
8. Select "UEFI Boot Menu"
9. Select "Boot From File"
10. Select "[UEFI] Disk connected to front USB..."
11. Select "Shell.efi"
12. Press 'ESC' within 5 seconds when prompted.
13. At the command prompt, select the thumb drive by typing "fs0:"
14. Type: "NvmExpressDxe.x64.efi" (you can use the tab key to complete the filename) and press enter
15. Type: "connect" and press enter, the system should respond with several "Connect – Handle [xx] Result Success" messages
16. Restart the shell by typing "exit" to return to the Boot Manager menu.
17. Select "UEFI Boot Menu" then "Boot From File" then "[UEFI] Disk connected to front USB."
18. Select "Shell.efi"
19. Press 'ESC' within 5 seconds when prompted.
20. Type "fs1:" to select the optical drive
21. Type "cd efi" to Change Directory to the EFI directory on the installer disc
22. Type "cd boot" to Change Directory to the boot directory
23. Type "bootx64.efi" to start the installation process, be ready to press any key when prompted to boot from the optical disc
24. Follow the Windows installer prompts to format the NVMe device and complete the install
  - a. Be sure to select "Custom Install (Advanced)" to select the installation drive.
25. Once the install completes the system reboots
26. Press function key to start UEFI boot manager as in step 7

- a. You may see a message indicating that all disk drives from the previous configuration are gone. Press any key to continue.
27. Select “UEFI Boot Menu”
28. Select the NVMe device, which may appear as an unknown device.
29. OS will boot

**Observable Results:**

1. Verify that the operating system booted as expected, the NVMe Device was identified properly by the operating system for each power down or reboot cycle.

**Possible Problems:** Methods for performing this test may vary across different operating systems and drivers. Not all operating systems may have boot support for NVMe. Additionally different hardware platforms may behave differently, with slightly different steps necessary.

## **Appendix F – UNH-IOL Interop Test Bed**

**Purpose:** To provide guidance on products that may be potential interop partners.

**References:**

[1] NVMe Specification

**Resource Requirements:**

NVMe Host Platform and Device.

**Last Modification:** May 24, 2016

**Discussion:** NVMe Interoperability should be performed across a variety of host system hardware and operating systems. UNH-IOL maintains a test bed of potential interop partners for use during private in-lab testing and plugfest events. Currently products available in the interop test bed can be seen here:  
<https://www.iol.unh.edu/testing/storage/nvme/equipment>

## Appendix G – NVMe Integrators List Requirements

**Purpose:** To provide guidance on what tests are required for NVMe Integrators List Qualification

**References:**

[1] NVMe Integrators List Policy Document

**Resource Requirements:**

NVMe Host Platform and Device.

**Last Modification:** July 14, 2016

**Discussion:** The interop test requirements are indicated in the Test name, showing which items are Mandatory, Mandatory if Supported, and FYI, as well as how many interop partners a test must be performed with. If the product under test is a device, Among the passing NVMe Host Platforms, there must be at least one Linux-based OS and at least one Windows-based OS represented and at least 2 of the following drivers represented:

NVMe Linux Driver (since Kernel Version 3.3.0)

Windows OFA Reference Driver 1.3 or 1.4

Microsoft StorNVMe.sys “In Box” Driver

If the product under test is a host, it is only necessary to use one operating system. The operating system used can be any OS chosen by the host.