White Paper

Verification of ZNS

Zoned Namespaces

Verification for SSD Drives

Authors

Vince Asbridge, Founder & President, SANBlaze

Haiyan Lin, Sr. Software Engineer, SANBlaze

Table of Contents

List of Figures

1. Introduction

SANBlaze has announced the availability of ZNS (Zoned Namespace) verification that allows you to quickly and effectively test and validate the ZNS implementation of your solid state drives (SSDs). This white paper introduces ZNS and describes how to verify that your SSDs have implemented all ZNS features correctly using the SANBlaze SBExpress test and validation system.

2. Understanding ZNS

NVMe™ Zoned Namespace (ZNS) is a technical proposal under standardization by the NVM Express™ organization. It divides the logical address space of a namespace into zones. Each zone provides a Logical Block Address (LBA) range that must be written sequentially and if written again must be explicitly reset. This operation principle allows created namespaces that expose the natural boundaries of the device and provides offload management of internal mapping tables to the host.

2.1 Why ZNS for SSDs?

SSDs are intrinsically zoned devices due to flash characteristics. A page is the smallest area of the NAND flash memory that supports a write operation and consists of all the memory cells on the same WordLine. An erase block is the smallest area of the flash memory that can be erased in a single operation. Page and block sizes differ per manufacturer and flash generation. For example, 19nm 64Gb MLC flash contains 16KB page size and 4MB block size. 16KB page size corresponds to 16,384 bytes that are dedicated for data and 1,280 bytes that are available for control and Error Correction Code (ECC) information.

NAND flash technology has evolved from SLC (Single-Level Cell, one bit per cell) to MLC (Multi-Level Cell, 2 bits per cell), then to TLC (3 bits per cell) and the current QLC (4 bits per cell). SLC NAND provides faster write speed and longer write endurance (around 30,000 – 50,000 Program/Erase Cycles) but is more expensive. MLC NAND offers a larger capacity, twice the density of SLC but with less endurance (around 3,000 Program/ Erase Cycles). TLC and QLC increase capacity significantly but at the cost of much less endurance (maybe around 300 Program/Erase Cycles), lower performance, and the need for more DRAM to map the higher capacity. DRAM is the highest cost after NAND in a typical SSD.

ZNS introduces a new type of NVMe drive that provides several benefits over traditional SSDs. It divides one namespace into multiple zones and only allows sequential write in each zone.

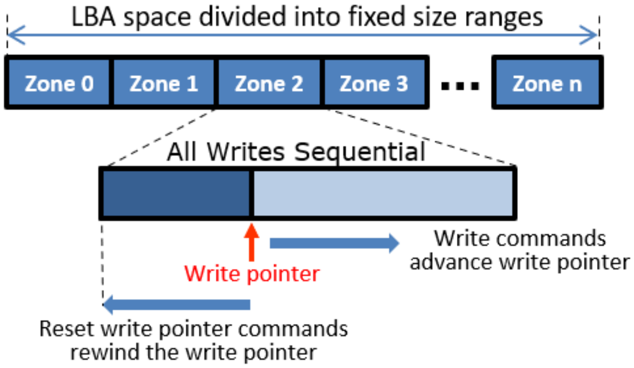

LBA space divided into fixed size ranges
Zone 0
Zone 1
Zone 2
Zone 3
...
Zone n
All Writes Sequential
Write pointer
Write commands advance write pointer
Reset write pointer commands rewind the write pointer

Figure 1: Zoned Namespace

SSDs cooperate using distributed FTL for the sequential access and eliminate multiple layers of indirection. No complex topology provisioning is needed because zones are logical. ZNS reduces write amplification, improves internal data movement, improves wear reduction, improves latency outliers and throughput, reduces DRAM in SSD (smaller L2P) and reduces the need for media over-provisioning. With the zones aligned to the internal physical properties of the NAND flash, several inefficiencies in the placement of data can be eliminated. In particular, the problem commonly known as the log-on-log challenge is naturally solved.

2.2 ZNS Model and State Machine

The ZNS model is similar to ZBC (Zoned Block Commands) and ZAC (Zoned ATA Commands) for SMR HDDs, but the interface is optimized for SSDs to align with media characteristics (i.e., aligned fixed zone size to NAND block sizes, and aligned variable zone capacity to physical media sizes). There are 7 states defined for ZNS as well: Empty, Full, Implicit Open, Explicit Open, Closed, Read Only and Offline. Valid transitions between each state can be changed by the NVMe Write, Zone Management Command (Open, Close, Finish, Reset) and Device Resets as shown in the zone state machine below.

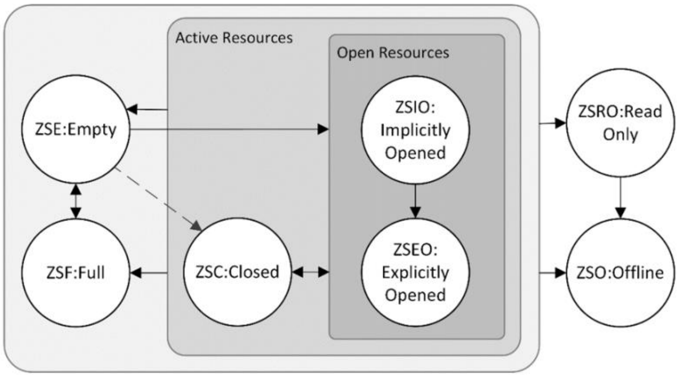


Figure 2: Zone State Machine

2.3 ZNS Commands

ZNS commands include Zoned Admin Command Sets and Zoned I/O Commands.

2.3.1 Zoned Admin Command Sets

The *NVMe – TP 4053 Zoned Namespaces 2020.03.19 – Final* specification provides specific additions to the ZNS Admin Command Set as follows:

- Identify Namespace Data Structure (TBD – specification not complete)
- Identify Controller Data Structure (TBD – specification not complete)
- Asynchronous Events Information
- Log page 0xBF
- Set Feature (Asynchronous Event Configuration)
- Sanitize
- Controller Architecture (Administrative Controller)

2.3.2 Zoned I/O Commands

The *NVMe – TP 4053 Zoned Namespaces 2020.03.19 – Final* specification provides specific commands for the Zoned Namespaces Command Set as follows:

- Flush
- Write
- Read
- Write Uncorrectable
- Compare
- Write Zeroes
- Dataset Management
- Verify
- Reservation Register
- Reservation Report
- Reservation Acquire
- Reservation Release
- Copy
- Zone Management Send
- Zone Management Receive
- Zone Append

Most commands are defined in the *NVMe specification v1.4* except the “Zone Management Send,” “Zone Management Receive” and “Zone Append” which are new.

Each zone is allowed to sequentially write only. If a sequential write in one zone in an SSD has a Queue Depth > 1 then it means multiple writes per zone, and it will involve significant lock contention and affect write performance. The Benchmark below shows multiple writes to a zone has low scalability, and one write per zone generates good performance. But write performance is improved by writing to multiple zones. Using the “Zone Append” command that appends data to a zone with an implicit write pointer (without defining the offset) improves performance significantly. The SSD returns an LBA where data was written in the zone and it will allow a higher Queue Depth (no host serialization).

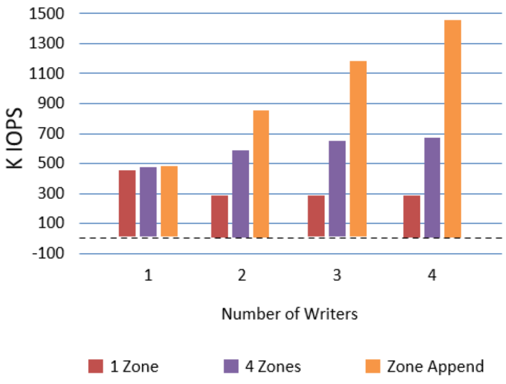


Figure 3: Scalability with Multiple Writers

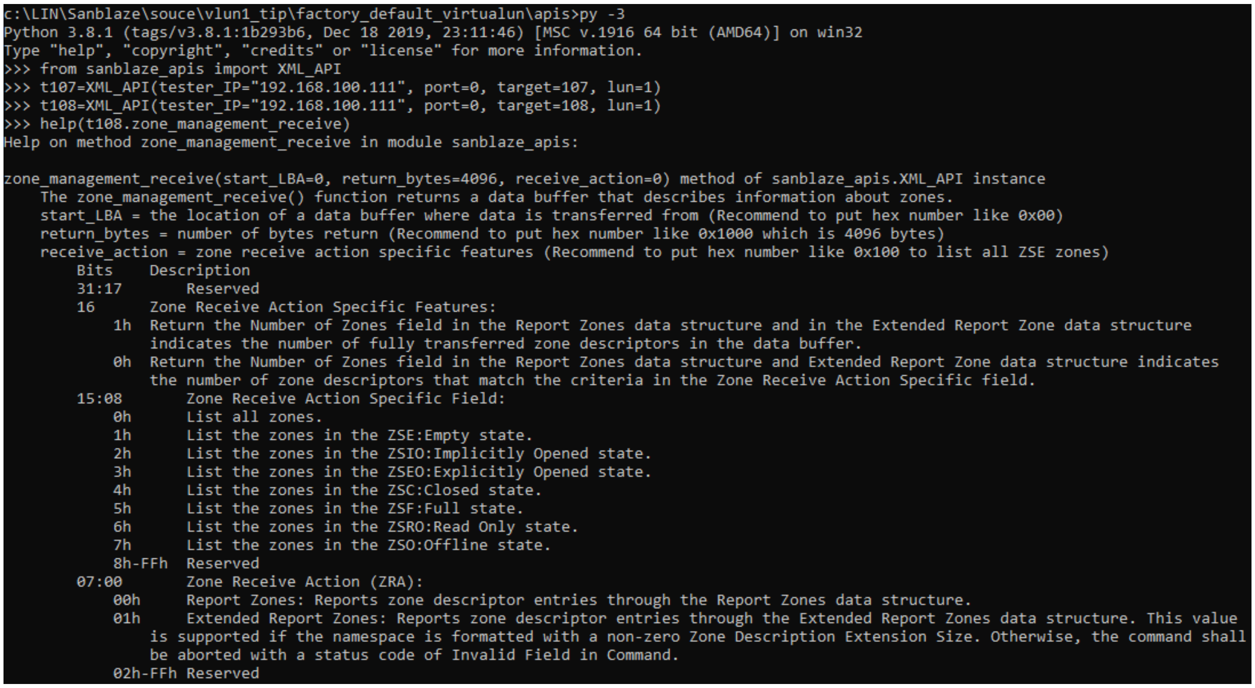
3. ZNS Verification by SANBlaze

The SANBlaze engineering team has incorporated ZNS testing into its SBExpress platform, and we are proud to be the industry's first to provide ZNS testing and validation to our customers. SANBlaze Application Support for ZNS includes Certified by SANBlaze pre-developed test cases that allow users to start validating ZNS support and capability right out of the box. Test cases support the following functionality:

- Support all Zoned Admin Command Sets and Zoned I/O Command Sets defined in the *NVMe – TP 4053 Zoned Namespaces 2020.03.19 – Final* specification in our SBExpress GUI, command line interface, XML API interface, and Python wrapped API interface for test automation.
- Customized Linux driver to handle ZNS state machine transition and sequential write requirement in each zone.
- Support multiple threads I/O running in the zones of ZNS in parallel with high throughputs. Each zone can be tested using write, read, compare, and append as needed. Each zone will be reset at the start, and then later when finished at the end.
- Namespace management for ZNS.
- Negative testing through scripts to test all ZNS features.

3.1 Zone Management/Append Examples with the SANBlaze Platform

3.1.1 Zone Management Receive



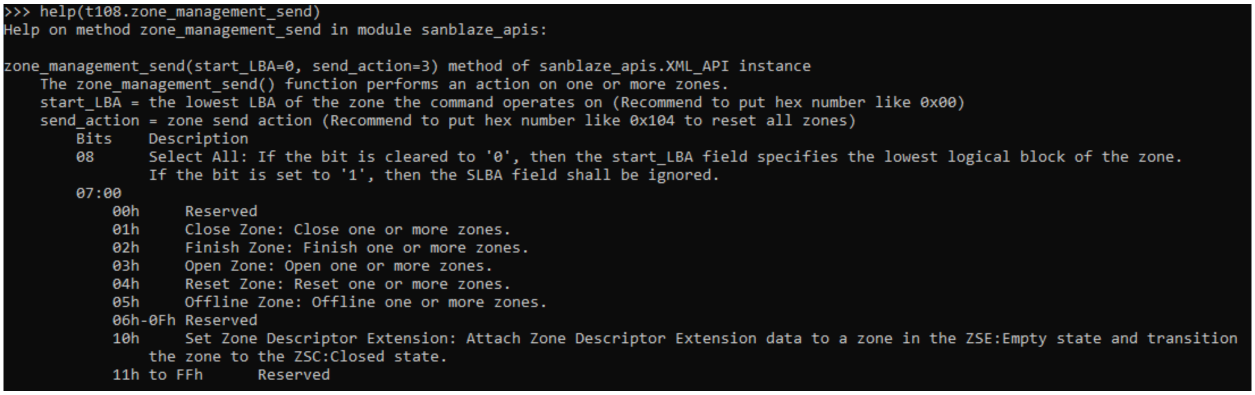
```
c:\LIN\Sanblaze\souce\vlun1_tip\factory_default_virtualun\apis>py -3
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from sanblaze_apis import XML_API
>>> t107=XML_API(tester_IP="192.168.100.111", port=0, target=107, lun=1)
>>> t108=XML_API(tester_IP="192.168.100.111", port=0, target=108, lun=1)
>>> help(t108.zone_management_receive)
Help on method zone_management_receive in module sanblaze_apis:

zone_management_receive(start_LBA=0, return_bytes=4096, receive_action=0) method of sanblaze_apis.XML_API instance
    The zone_management_receive() function returns a data buffer that describes information about zones.
    start_LBA = the location of a data buffer where data is transferred from (Recommend to put hex number like 0x00)
    return_bytes = number of bytes return (Recommend to put hex number like 0x1000 which is 4096 bytes)
    receive_action = zone receive action specific features (Recommend to put hex number like 0x100 to list all ZSE zones)
        Bits    Description
        31:17       Reserved
        16      Zone Receive Action Specific Features:
            1h  Return the Number of Zones field in the Report Zones data structure and in the Extended Report Zone data structure
                indicates the number of fully transferred zone descriptors in the data buffer.
            0h  Return the Number of Zones field in the Report Zones data structure and Extended Report Zone data structure indicates
                the number of zone descriptors that match the criteria in the Zone Receive Action Specific field.
        15:08       Zone Receive Action Specific Field:
            0h      List all zones.
            1h      List the zones in the ZSE:Empty state.
            2h      List the zones in the ZSIO:Implicitly Opened state.
            3h      List the zones in the ZSEO:Explicitly Opened state.
            4h      List the zones in the ZSC:Closed state.
            5h      List the zones in the ZSF:Full state.
            6h      List the zones in the ZSRO:Read Only state.
            7h      List the zones in the ZSO:Offline state.
            8h-FFh  Reserved
        07:00       Zone Receive Action (ZRA):
            00h     Report Zones: Reports zone descriptor entries through the Report Zones data structure.
            01h     Extended Report Zones: Reports zone descriptor entries through the Extended Report Zones data structure. This value
                is supported if the namespace is formatted with a non-zero Zone Description Extension Size. Otherwise, the command shall
                be aborted with a status code of Invalid Field in Command.
            02h-FFh Reserved
```

Figure 4: Help Info for Zone Management Receive

```
>>> t108.zone_management_receive()    # List all zones with default input arguments
Command ZoneManagementReceive passed on port 0 target 108 in tester 192.168.100.111. Output is
decoded as follows:
                                                Num_Zones = 0x0000000000003B98
                    Zone_Descriptor_0:
                                                Zone_Type = 0x02
```

```
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000000000
                                Write_Pointer = 0x0000000000000000
               Zone_Descriptor_1:
                                    Zone_Type = 0x02
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000004000
                                Write_Pointer = 0x0000000000004000
. . .

>>> t108.zone_management_receive(receive_action=0x10000) # Report zone structure in data buffer
Command ZoneManagementReceive passed on port 0 target 108 in tester 192.168.100.111. Output is
decoded as follows:
                                    Num_Zones = 0x000000000000003F
               Zone_Descriptor_0:
                                    Zone_Type = 0x02
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000000000
                                Write_Pointer = 0x0000000000000000
. . .
```

3.1.2 Zone Management Send



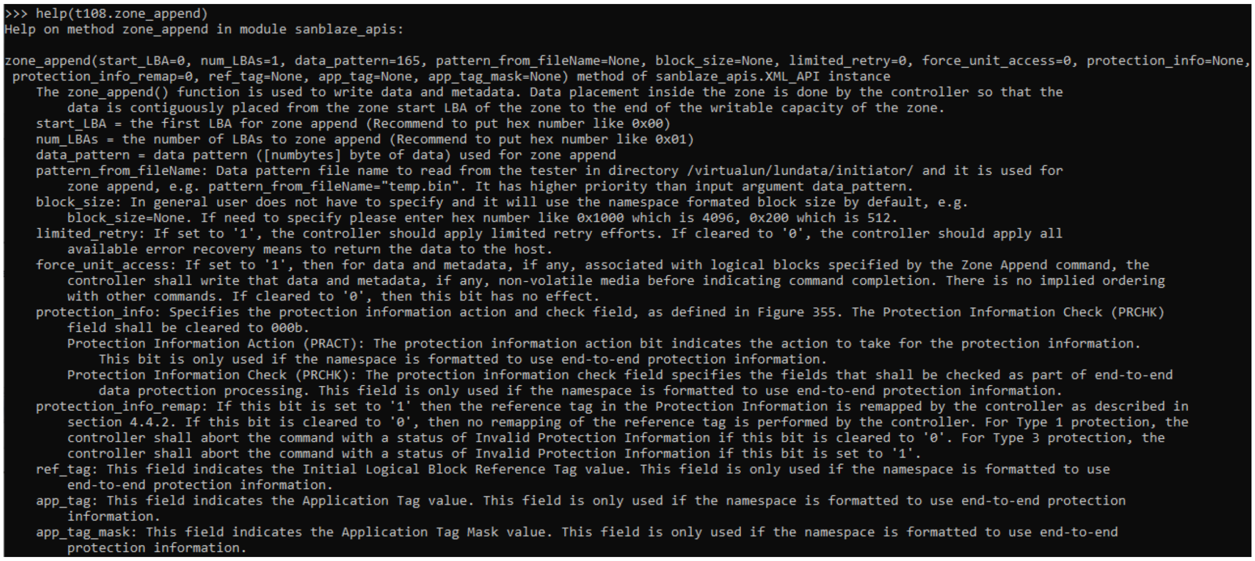
```
>>> help(t108.zone_management_send)
Help on method zone_management_send in module sanblaze_apis:

zone_management_send(start_LBA=0, send_action=3) method of sanblaze_apis.XML_API instance
    The zone_management_send() function performs an action on one or more zones.
    start_LBA = the lowest LBA of the zone the command operates on (Recommend to put hex number like 0x00)
    send_action = zone send action (Recommend to put hex number like 0x104 to reset all zones)
        Bits    Description
        08      Select All: If the bit is cleared to '0', then the start_LBA field specifies the lowest logical block of the zone.
                If the bit is set to '1', then the SLBA field shall be ignored.
        07:00
            00h     Reserved
            01h     Close Zone: Close one or more zones.
            02h     Finish Zone: Finish one or more zones.
            03h     Open Zone: Open one or more zones.
            04h     Reset Zone: Reset one or more zones.
            05h     Offline Zone: Offline one or more zones.
            06h-0Fh Reserved
            10h     Set Zone Descriptor Extension: Attach Zone Descriptor Extension data to a zone in the ZSE:Empty state and transition
                the zone to the ZSC:Closed state.
            11h to FFh      Reserved
```

Figure 5: Help Info for Zone Management Send

```
>>> t108.zone_management_send()       # open zone 0 with default input arguments
Command ZoneManagementSend passed on port 0 target 108 in tester 192.168.100.111. Output is
decoded as follows:

Command Completion Queue Status is decoded as follows:
                              CommandSpecific = 0x00000000
                                    Reserved0 = 0x00000000
                              SQ_Head_Pointer = 0x0004
                                SQ_Identifier = 0x0001
                           Command_Identifier = 0x07CB
                    Status_Field:
                                     PhaseBit = 0x01
                                   StatusCode = 0x0000
                               StatusCodeType = 0x00
                                     Reserved = 0x00
                              MoreInformation = 0x00
                                   DoNotRetry = 0x00

>>> t108.zone_management_receive()  # List all zones with default input arguments
Command ZoneManagementReceive passed on port 0 target 108 in tester 192.168.100.111. Output is
decoded as follows:
                                    Num_Zones = 0x0000000000003B98
```

```
               Zone_Descriptor_0:
                                    Zone_Type = 0x02
                                   Zone_State = 0x30
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000000000
                                Write_Pointer = 0x0000000000000000
               Zone_Descriptor_1:
                                    Zone_Type = 0x02
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000004000
                                Write_Pointer = 0x0000000000004000
. . .
```

3.1.3 Zone Append

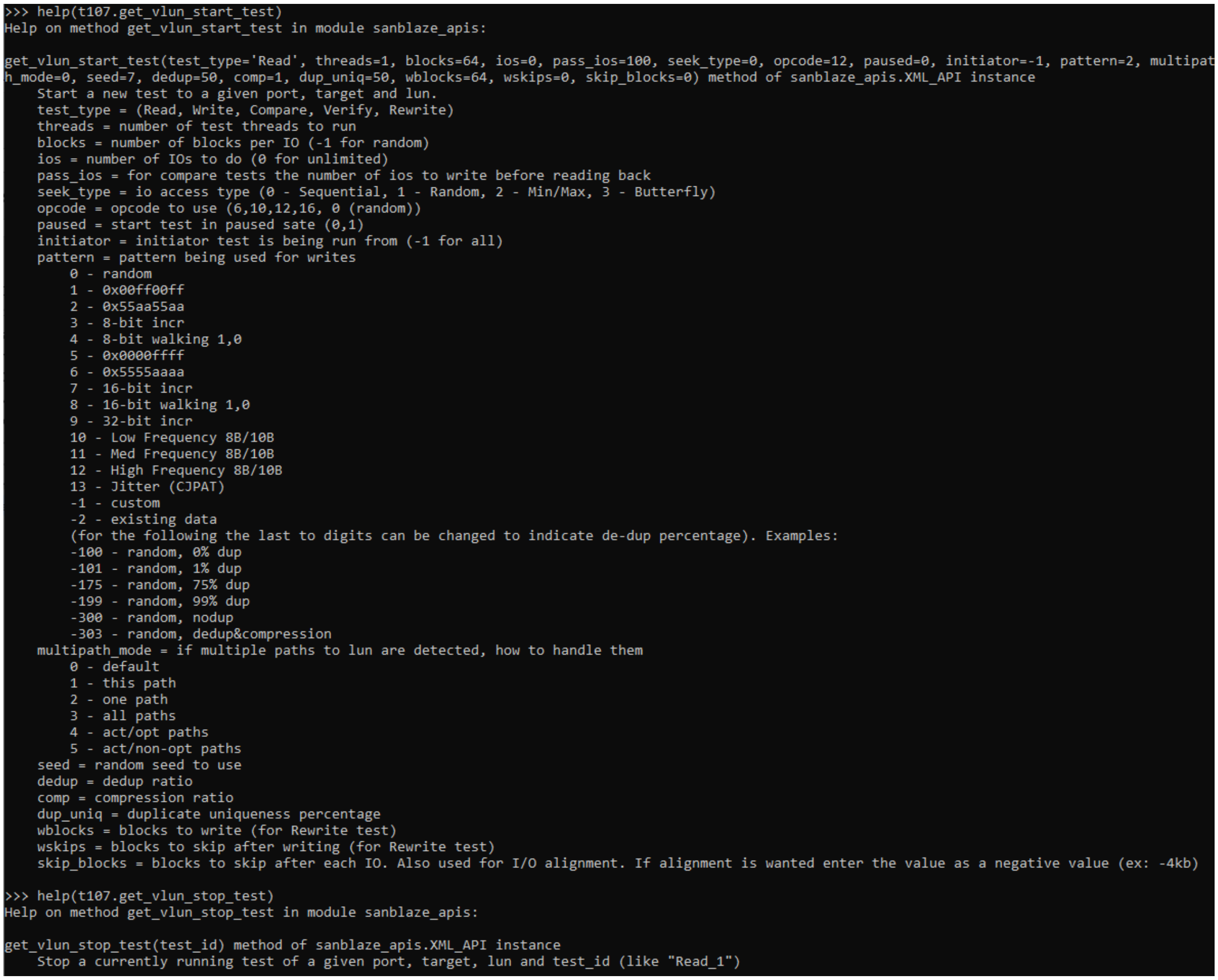


```
>>> help(t108.zone_append)
Help on method zone_append in module sanblaze_apis:

zone_append(start_LBA=0, num_LBAs=1, data_pattern=165, pattern_from_fileName=None, block_size=None, limited_retry=0, force_unit_access=0, protection_info=None,
 protection_info_remap=0, ref_tag=None, app_tag=None, app_tag_mask=None) method of sanblaze_apis.XML_API instance
    The zone_append() function is used to write data and metadata. Data placement inside the zone is done by the controller so that the
        data is contiguously placed from the zone start LBA of the zone to the end of the writable capacity of the zone.
    start_LBA = the first LBA for zone append (Recommend to put hex number like 0x00)
    num_LBAs = the number of LBAs to zone append (Recommend to put hex number like 0x01)
    data_pattern = data pattern ([numbytes] byte of data) used for zone append
    pattern_from_fileName: Data pattern file name to read from the tester in directory /virtualun/lundata/initiator/ and it is used for
        zone append, e.g. pattern_from_fileName="temp.bin". It has higher priority than input argument data_pattern.
    block_size: In general user does not have to specify and it will use the namespace formated block size by default, e.g.
        block_size=None. If need to specify please enter hex number like 0x1000 which is 4096, 0x200 which is 512.
    limited_retry: If set to '1', the controller should apply limited retry efforts. If cleared to '0', the controller should apply all
        available error recovery means to return the data to the host.
    force_unit_access: If set to '1', then for data and metadata, if any, associated with logical blocks specified by the Zone Append command, the
        controller shall write that data and metadata, if any, non-volatile media before indicating command completion. There is no implied ordering
        with other commands. If cleared to '0', then this bit has no effect.
    protection_info: Specifies the protection information action and check field, as defined in Figure 355. The Protection Information Check (PRCHK)
        field shall be cleared to 000b.
        Protection Information Action (PRACT): The protection information action bit indicates the action to take for the protection information.
            This bit is only used if the namespace is formatted to use end-to-end protection information.
        Protection Information Check (PRCHK): The protection information check field specifies the fields that shall be checked as part of end-to-end
            data protection processing. This field is only used if the namespace is formatted to use end-to-end protection information.
    protection_info_remap: If this bit is set to '1' then the reference tag in the Protection Information is remapped by the controller as described in
        section 4.4.2. If this bit is cleared to '0', then no remapping of the reference tag is performed by the controller. For Type 1 protection, the
        controller shall abort the command with a status of Invalid Protection Information if this bit is cleared to '0'. For Type 3 protection, the
        controller shall abort the command with a status of Invalid Protection Information if this bit is set to '1'.
    ref_tag: This field indicates the Initial Logical Block Reference Tag value. This field is only used if the namespace is formatted to use
        end-to-end protection information.
    app_tag: This field indicates the Application Tag value. This field is only used if the namespace is formatted to use end-to-end protection
        information.
    app_tag_mask: This field indicates the Application Tag Mask value. This field is only used if the namespace is formatted to use end-to-end
        protection information.
```

Figure 6: Help Info for Zone Append

```
>>> t108.zone_append()        # zone append LBA 0 in zone 0 with default input arguments
Zone append data pattern 0xa5 to starting LBA 0x0 with 0x1 LBAs
0000    A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5 A5         ¥¥¥¥¥¥¥¥¥¥¥¥¥¥¥¥
. . .
Command Completion Queue Status is decoded as follows:
                              CommandSpecific = 0x00000000
                                    Reserved0 = 0x00000000
                              SQ_Head_Pointer = 0x0006
                                SQ_Identifier = 0x0001
                           Command_Identifier = 0x0704
                    Status_Field:
                                     PhaseBit = 0x01
                                   StatusCode = 0x0000
                               StatusCodeType = 0x00
                                     Reserved = 0x00
                              MoreInformation = 0x00
                                   DoNotRetry = 0x00

>>> t108.zone_management_receive()  # List all zones with default input arguments
Command ZoneManagementReceive passed on port 0 target 108 in tester 192.168.100.111. Output is
decoded as follows:
                                    Num_Zones = 0x0000000000003B98
               Zone_Descriptor_0:
                                    Zone_Type = 0x02
                                   Zone_State = 0x30
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000000000
                                Write_Pointer = 0x0000000000000001
               Zone_Descriptor_1:
                                    Zone_Type = 0x02
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000003000
                               Zone_Start_LBA = 0x0000000000004000
                                Write_Pointer = 0x0000000000004000
. . .
```

3.2 Multiple Threads I/O in ZNS Examples with SANBlaze Platform



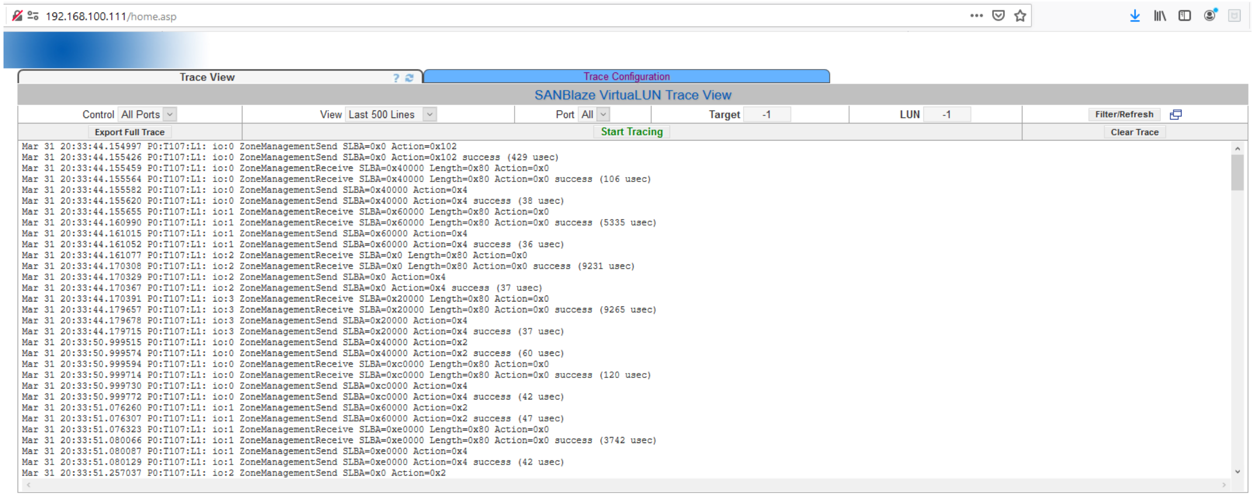
```
>>> help(t107.get_vlun_start_test)
Help on method get_vlun_start_test in module sanblaze_apis:

get_vlun_start_test(test_type='Read', threads=1, blocks=64, ios=0, pass_ios=100, seek_type=0, opcode=12, paused=0, initiator=-1, pattern=2, multipat
h_mode=0, seed=7, dedup=50, comp=1, dup_uniq=50, wblocks=64, wskips=0, skip_blocks=0) method of sanblaze_apis.XML_API instance
    Start a new test to a given port, target and lun.
    test_type = (Read, Write, Compare, Verify, Rewrite)
    threads = number of test threads to run
    blocks = number of blocks per IO (-1 for random)
    ios = number of IOs to do (0 for unlimited)
    pass_ios = for compare tests the number of ios to write before reading back
    seek_type = io access type (0 - Sequential, 1 - Random, 2 - Min/Max, 3 - Butterfly)
    opcode = opcode to use (6,10,12,16, 0 (random))
    paused = start test in paused sate (0,1)
    initiator = initiator test is being run from (-1 for all)
    pattern = pattern being used for writes
        0 - random
        1 - 0x00ff00ff
        2 - 0x55aa55aa
        3 - 8-bit incr
        4 - 8-bit walking 1,0
        5 - 0x0000ffff
        6 - 0x5555aaaa
        7 - 16-bit incr
        8 - 16-bit walking 1,0
        9 - 32-bit incr
        10 - Low Frequency 8B/10B
        11 - Med Frequency 8B/10B
        12 - High Frequency 8B/10B
        13 - Jitter (CJPAT)
        -1 - custom
        -2 - existing data
        (for the following the last to digits can be changed to indicate de-dup percentage). Examples:
        -100 - random, 0% dup
        -101 - random, 1% dup
        -175 - random, 75% dup
        -199 - random, 99% dup
        -300 - random, nodup
        -303 - random, dedup&compression
    multipath_mode = if multiple paths to lun are detected, how to handle them
        0 - default
        1 - this path
        2 - one path
        3 - all paths
        4 - act/opt paths
        5 - act/non-opt paths
    seed = random seed to use
    dedup = dedup ratio
    comp = compression ratio
    dup_uniq = duplicate uniqueness percentage
    wblocks = blocks to write (for Rewrite test)
    wskips = blocks to skip after writing (for Rewrite test)
    skip_blocks = blocks to skip after each IO. Also used for I/O alignment. If alignment is wanted enter the value as a negative value (ex: -4kb)

>>> help(t107.get_vlun_stop_test)
Help on method get_vlun_stop_test in module sanblaze_apis:

get_vlun_stop_test(test_id) method of sanblaze_apis.XML_API instance
    Stop a currently running test of a given port, target, lun and test_id (like "Read_1")
```

Figure 7: Help Info for Start Test and Stop Test

```
>>> t107.zone_management_receive(receive_action=0x10000) # Report zone structure in data buffer
Command ZoneManagementReceive passed on port 0 target 107 in tester 192.168.100.111. Output is
decoded as follows:
                                 Num_Zones = 0x000000000000003F
               Zone_Descriptor_0:
                                 Zone_Type = 0x02
                                Zone_State = 0x10
                           Zone_Attributes = 0x00
                             Zone_Capacity = 0x0000000000018000
                            Zone_Start_LBA = 0x0000000000000000
```

```
                                Write_Pointer = 0x0000000000000000
               Zone_Descriptor_1:
                                    Zone_Type = 0x02
                                   Zone_State = 0x10
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000018000
                               Zone_Start_LBA = 0x0000000000020000
                                Write_Pointer = 0x0000000000020000
. . .

>>>t107.get_vlun_start_test(test_type='Compare',threads=4,blocks=64)#Start 4 threads Compare test
<result>
  <test>
    <status>0</status>
    <test_id>Compare_2</test_id>
    <index>1</index>
  </test>
</result>

>>> t107.zone_management_receive(receive_action=0x10000)  # Report zone structure in data buffer
Command ZoneManagementReceive passed on port 0 target 107 in tester 192.168.100.111. Output is
decoded as follows:
                                    Num_Zones = 0x000000000000003F
               Zone_Descriptor_0:
                                    Zone_Type = 0x02
                                   Zone_State = 0xE0
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000018000
                               Zone_Start_LBA = 0x0000000000000000
                                Write_Pointer = 0x00000000FFFFFFFF
               Zone_Descriptor_1:
                                    Zone_Type = 0x02
                                   Zone_State = 0xE0
                              Zone_Attributes = 0x00
                                Zone_Capacity = 0x0000000000018000
                               Zone_Start_LBA = 0x0000000000020000
                                Write_Pointer = 0x00000000FFFFFFFF
. . .

>>> t107.get_vlun_stop_test(test_id='Compare_2')  # Stop the 4 threads Compare test above
<result>
  <status>0</status>
</result>
```

The trace from the SANBlaze platform shows that the 4 threads are running as follows: Each thread is running in one zone, so 4 threads are running in 4 zones. Once complete the first 4 zones begin running on the next 4 zones until they are stopped by user or complete the full test.



```
Mar 31 20:33:44.154997 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x0 Action=0x102
Mar 31 20:33:44.155426 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x0 Action=0x102 success (429 usec)
Mar 31 20:33:44.155459 P0:T107:L1: io:0 ZoneManagementReceive SLBA=0x40000 Length=0x80 Action=0x0
Mar 31 20:33:44.155564 P0:T107:L1: io:0 ZoneManagementReceive SLBA=0x40000 Length=0x80 Action=0x0 success (106 usec)
Mar 31 20:33:44.155582 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x40000 Action=0x4
Mar 31 20:33:44.155620 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x40000 Action=0x4 success (38 usec)
Mar 31 20:33:44.155655 P0:T107:L1: io:1 ZoneManagementReceive SLBA=0x60000 Length=0x80 Action=0x0
Mar 31 20:33:44.160990 P0:T107:L1: io:1 ZoneManagementReceive SLBA=0x60000 Length=0x80 Action=0x0 success (5335 usec)
Mar 31 20:33:44.161015 P0:T107:L1: io:1 ZoneManagementSend SLBA=0x60000 Action=0x4
Mar 31 20:33:44.161052 P0:T107:L1: io:1 ZoneManagementSend SLBA=0x60000 Action=0x4 success (36 usec)
Mar 31 20:33:44.161077 P0:T107:L1: io:2 ZoneManagementReceive SLBA=0x0 Length=0x80 Action=0x0
Mar 31 20:33:44.170308 P0:T107:L1: io:2 ZoneManagementReceive SLBA=0x0 Length=0x80 Action=0x0 success (9231 usec)
Mar 31 20:33:44.170329 P0:T107:L1: io:2 ZoneManagementSend SLBA=0x0 Action=0x4
Mar 31 20:33:44.170367 P0:T107:L1: io:2 ZoneManagementSend SLBA=0x0 Action=0x4 success (37 usec)
Mar 31 20:33:44.170391 P0:T107:L1: io:3 ZoneManagementReceive SLBA=0x20000 Length=0x80 Action=0x0
Mar 31 20:33:44.179657 P0:T107:L1: io:3 ZoneManagementReceive SLBA=0x20000 Length=0x80 Action=0x0 success (9265 usec)
Mar 31 20:33:44.179678 P0:T107:L1: io:3 ZoneManagementSend SLBA=0x20000 Action=0x4
Mar 31 20:33:44.179715 P0:T107:L1: io:3 ZoneManagementSend SLBA=0x20000 Action=0x4 success (37 usec)
Mar 31 20:33:50.999515 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x40000 Action=0x2
Mar 31 20:33:50.999574 P0:T107:L1: io:0 ZoneManagementSend SLBA=0x40000 Action=0x2 success (60 usec)
Mar 31 20:33:50.999594 P0:T107:L1: io:0 ZoneManagementReceive SLBA=0xc0000 Length=0x80 Action=0x0
Mar 31 20:33:50.999714 P0:T107:L1: io:0 ZoneManagementReceive SLBA=0xc0000 Length=0x80 Action=0x0 success (120 usec)
Mar 31 20:33:50.999730 P0:T107:L1: io:0 ZoneManagementSend SLBA=0xc0000 Action=0x4
Mar 31 20:33:50.999772 P0:T107:L1: io:0 ZoneManagementSend SLBA=0xc0000 Action=0x4 success (42 usec)
Mar 31 20:33:51.076260 P0:T107:L1: io:1 ZoneManagementSend SLBA=0x60000 Action=0x2
Mar 31 20:33:51.076307 P0:T107:L1: io:1 ZoneManagementSend SLBA=0x60000 Action=0x2 success (47 usec)
Mar 31 20:33:51.076323 P0:T107:L1: io:1 ZoneManagementReceive SLBA=0xe0000 Length=0x80 Action=0x0
Mar 31 20:33:51.080066 P0:T107:L1: io:1 ZoneManagementReceive SLBA=0xe0000 Length=0x80 Action=0x0 success (3742 usec)
Mar 31 20:33:51.080087 P0:T107:L1: io:1 ZoneManagementSend SLBA=0xe0000 Action=0x4
Mar 31 20:33:51.080129 P0:T107:L1: io:1 ZoneManagementSend SLBA=0xe0000 Action=0x4 success (42 usec)
Mar 31 20:33:51.257037 P0:T107:L1: io:2 ZoneManagementSend SLBA=0x0 Action=0x2
```

Figure 8: Trace of Multiple Threads I/O in zoned namespace

4. SANBlaze ZNS Qualification Suite

There are 3 major technical proposals for ZNS so far as follows:

- TP4053
- TP4056
- TP4076

The current SANBlaze ZNS qualification suite includes 175 scripts to cover all sections in these 3 technical proposals, and has added some I/O tests to check data integrity of all zones as well as I/O performance measurement.

4.1 TP4053 Qualification

TP4053 defines Zoned Namespaces and the associated Zoned Namespace Command Set. SANBlaze has developed 118 scripts to cover the TP4053 qualification.

Script Name	Test Description
ZNS_01.01.01_IdentifyZNSNamespace.sh	Zoned Namespace Command Set Identify Namespace Data Structure (CNS 05h) 1. Verify structure is returned and is of correct size 2. Verify reserved bytes are reserved 3. Report details of data structure
ZNS_01.01.02_IdentifyZNSController.sh	Zoned Namespace Command Set Identify Controller Data Structure (CNS 06h) 1. Verify structure is returned and is of correct size 2. Verify reserved bytes are reserved
ZNS_01.01.03_IdentifyController.sh	Identify Controller Data Structure, I/O Command Set Independent (CNS 01h) 1. Check bit value and report its status
ZNS_01.05.02_AsyncEvents-NoAck.sh	Case 2: 1. Issue async event 2. Verify each described action doesn't cause event to be acknowledged
ZNS_01.04.01_GetLogPage.sh	1. Issue every applicable log page on ZNS and verify it's successful
ZNS_01.06.01_Sanitize-0b-XXb-Yb-Zb.sh	Case 1: Fig. 18 row 1
ZNS_01.06.02_Sanitize-1b-00b-0b-Zb.sh	Case 2: Fig. 18 row 2
ZNS_01.06.03_Sanitize-1b-00b-1b-0b.sh	Case 3: Fig. 18 row 3
ZNS_01.06.04_Sanitize-1b-00b-1b-1b.sh	Case 4: Fig. 18 row 4
ZNS_01.06.05_Sanitize-1b-01b-0b-Zb.sh	Case 5: Fig. 18 row 5
ZNS_01.06.06_Sanitize-1b-01b-1b-1b.sh	Case 6: Fig. 18 row 6
ZNS_01.06.07_Sanitize-1b-01b-1b-0b.sh	Case 7: Fig. 18 row 7
ZNS_01.06.08_Sanitize-1b-10b-0b-Zb.sh	Case 8: Fig. 18 row 8
ZNS_01.06.09_Sanitize-1b-10b-1b-1b.sh	Case 9: Fig. 18 row 9
ZNS_01.06.10_Sanitize-1b-10b-1b-0b.sh	Case 10: Fig. 18 row 10
ZNS_01.06.11_Sanitize-1b-11b-Yb-Zb.sh	Case 11: Fig. 18 row 11
ZNS_02.01.01_Flush.sh	Case 1: Verify successful command with NSID=X
ZNS_02.07.01_Dataset_Deallocate.sh	Case 1: Verify successful command with NSID=X
ZNS_02.07.02_Dataset_Read_Hint.sh	Case 1: Verify successful command with NSID=X
ZNS_02.07.03_Dataset_Write_Hint.sh	Case 1: Verify successful command with NSID=X
ZNS_02.09.01_ReservationRegister.sh	Case 1: Verify successful command with NSID=X

ZNS_02.10.01_ReservationReport.sh	Case 1: Verify successful command with NSID=X
ZNS_02.11.01_ReservationAcquire.sh	Case 1: Verify successful command with NSID=X
ZNS_02.12.01_ReservationRelease.sh	Case 1: Verify successful command with NSID=X
ZNS_02.02.01_Write.sh	Case 1: Verify successful command
ZNS_02.02.02_Write-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_02.02.03_Write-ZoneFull.sh	Case 3: Zone Is Full - verify this error code can be returned
ZNS_02.02.06_Write-ZoneInvalidWrite.sh	Case 6: Zone Invalid Write - verify this error code can be returned
ZNS_02.02.07_Write-TooManyActiveZones.sh	Case 7: Too Many Active Zones - verify this error code can be returned
ZNS_02.02.08_Write-TooManyOpenZones.sh	Case 8: Too Many Open Zones - verify this error code can be returned
ZNS_02.03.01_Read.sh	Case 1: Verify successful command
ZNS_02.03.02_Read-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_02.04.01_WriteUncorrectable.sh	Case 1: Verify successful command
ZNS_02.04.02_WriteUncorrectable-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_02.04.03_WriteUncorrectable-ZoneFull.sh	Case 3: Zone Is Full - verify this error code can be returned
ZNS_02.04.06_WriteUncorrectable-ZoneInvalidWrite.sh	Case 6: Zone Invalid Write - verify this error code can be returned
ZNS_02.04.07_WriteUncorrectable-TooManyActiveZones.sh	Case 7: Too Many Active Zones - verify this error code can be returned
ZNS_02.04.08_WriteUncorrectable-TooManyOpenZones.sh	Case 8: Too Many Open Zones - verify this error code can be returned
ZNS_02.05.01_Compare.sh	Case 1: Verify successful command
ZNS_02.05.02_Compare-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_02.06.01_WriteZeroes.sh	Case 1: Verify successful command
ZNS_02.06.02_WriteZeroes-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_02.06.03_WriteZeroes-ZoneFull.sh	Case 3: Zone Is Full - verify this error code can be returned
ZNS_02.06.06_WriteZeroes-ZoneInvalidWrite.sh	Case 6: Zone Invalid Write - verify this error code can be returned
ZNS_02.06.07_WriteZeroes-TooManyActiveZones.sh	Case 7: Too Many Active Zones - verify this error code can be returned
ZNS_02.06.08_WriteZeroes-TooManyOpenZones.sh	Case 8: Too Many Open Zones - verify this error code can be returned
ZNS_02.08.01_Verify.sh	Case 1: Verify successful command
ZNS_02.08.02_Verify-ZoneBoundaryError.sh	Case 2: Zone Boundary Error - verify this error code can be returned
ZNS_05.01.01_ZoneMgmtSend-Rsvd.sh	Case 1: Set ZSA to each reserved field value and verify error code is returned
ZNS_05.01.02_ZoneMgmtSend-InvalidField-1.sh	Case 2: If the command SLBA field does not specify the starting logical block for a zone in the specified zoned namespace and the Select All bit is cleared to ‘0’, then the command shall be aborted with a status code of Invalid Field in Command
ZNS_05.01.03_ZoneMgmtSend-InvalidField-2.sh	Case 3: If the Zone Send Action field specifies Set Zone Descriptor Extension, and the Zone Descriptor Extension Size field value in the Identify Namespace data structure is cleared to 0h, then the command shall be aborted with a status code of Invalid Field in Command
ZNS_05.01.04_ZoneMgmtSend-WriteProtect.sh	Case 4: If the zoned namespace containing the specified zone is in the write protection state (refer to Namespace Write Protection section in the NVMe Base specification), then the command shall be aborted with a status code of Namespace is Write Protected

ZNS_05.01.05_ZoneMgmtSend-Abort.sh	Case 5: The command may be aborted according to the available Active Resources and available Open Resources as defined in section 2.5
ZNS_05.01.06_ZoneMgmtSend_ZoneMgmtRcv-PRPs_SGLs.sh	Case 6: Check if zone management send and receive commands working with both PRPs and SGLs.
ZNS_05.02.01_ZoneMgmtSend-CloseZone.sh	Case 1: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSIO:Implicitly Opened state or the ZSEO:Explicitly Opened state, the zone shall be is transitioned to the ZSC:Closed state
ZNS_05.02.02_ZoneMgmtSend-CloseZone-NoChange.sh	Case 2: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSC:Closed state, no change shall be made to the zone state
ZNS_05.02.03_ZoneMgmtSend-CloseZone-InvalidTransition.sh	Case 3: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSE:Empty state, the ZSF:Full state, the ZSRO:Read Only state or the ZSO:Offline state, the command shall be aborted with a status code of Invalid Zone State Transition
ZNS_05.02.04_ZoneMgmtSend-CloseZone-ClosedState.sh	Case 4: If the Select All bit is set to '1', then the SLBA field shall be ignored, and all zones that are in the ZSIO:Implicitly Opened state or ZSEO:Explicitly Opened state shall be transitioned to the ZSC:Closed state
ZNS_05.03.01_ZoneMgmtSend-FinishZone.sh	Case 1: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSE:Empty state, the ZSIO:Implicitly Opened state, the ZSEO:Explicitly Opened state, or the ZSC:Closed state, the zone shall be transitioned to the ZSF:Full state
ZNS_05.03.02_ZoneMgmtSend-FinishZone-NoChange.sh	Case 2: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSF:Full state, no change shall be made to the zone state
ZNS_05.03.04_ZoneMgmtSend-FinishZone-IgnoreSLBA.sh	Case 4: If the Select All bit is set to '1', then the SLBA field shall be ignored and all zones that are in the ZSIO:Implicitly Opened state, the ZSEO:Explicitly Opened state, or the ZSC:Closed state shall be transitioned to the ZSF:Full state, and the command completes successfully
ZNS_05.04.01_ZoneMgmtSend-OpenZone.sh	Case 1: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSE:Empty state, the ZSIO:Implicitly Opened state or the ZSC:Closed state, the zone should be transitioned to the ZSEO:Explicitly Opened state
ZNS_05.04.02_ZoneMgmtSend-OpenZone-NoChange.sh	Case 2: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSEO:Explicitly Opened state, no change shall be made to the zone state
ZNS_05.04.03_ZoneMgmtSend-OpenZone-InvalidTransition.sh	Case 3: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSF:Full state, the ZSRO:Read Only state or the ZSO:Offline state, the command shall be aborted with a status code of Invalid Zone State Transition
ZNS_05.04.04_ZoneMgmtSend-OpenZone-IgnoreSLBA.sh	Case 4: If the Select All bit is set to '1', then the SLBA field shall be ignored and all zones that are in the ZSC:Closed state should be transitioned to the ZSEO:Explicitly Opened state. If the operation causes the the number of Open Resources to exceed the value specified by the Maximum Open Resources field (refer to section 2.5), then the command shall be aborted with a status code of Too Many Open Zones, and no zone state transitions shall occur
ZNS_05.05.01_ZoneMgmtSend-ResetZone.sh	Case 1: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSIO:Implicitly Opened state, the ZSEO:Explicitly Opened state, the ZSC:Closed state, or the ZSF:Full state, the specified zone shall be transitioned to the ZSE:Empty state
ZNS_05.05.02_ZoneMgmtSend-ResetZone-NoChange.sh	Case 2: If the Select All bit is cleared to '0', and the zone specified by the SLBA field is in the ZSE:Empty state, no change shall be made to the zone state

ZNS_05.05.04_ZoneMgmtSend-ResetZone-IgnoreSLBA.sh	Case 4: If the Select All bit is set to ‘1’, then the SLBA field shall be ignored and each zone that is in the ZSIO:Implicitly Opened state, the ZSEO:Explicitly Opened state, the ZSC:Closed state, or the ZSF:Full state shall be transitioned to the ZSE:Empty state
ZNS_05.05.05_ZoneMgmtSend-ResetZone-ZoneDescriptor.sh	Case 5: 1. Create a zone 2. Issue Reset Zone command 3. Verify the following: If the command completes successfully, then the Zone Descriptor of each affected zone shall: a) set the Write Pointer zone attribute to the ZSLBA of the zone; and b) clear the following zone attribute bits to ‘0’: a) Zone Descriptor Extension Valid; b) Finish Zone Recommended; c) Reset Zone Recommended; and d) Zone Finished by Controller
ZNS_05.06.03_ZoneMgmtSend-OfflineZone-InvalidTransition.sh	Case 3: If the Select All bit is cleared to ‘0’, and the zone specified by the SLBA field is in the ZSE:Empty state, the ZSIO:Implicitly Opened state, the ZSEO:Explicitly Opened state, the ZSC:Closed state, or the ZSF:Full state, the command shall be aborted with a status code of Invalid Zone State Transition
ZNS_05.07.01_ZoneMgmtSend-SetZoneDescExt.sh	Case 1: If the Select All bit is cleared to ‘0’ and the zone specified by the SLBA field is in the ZSE:Empty state, the zone should be transitioned to the ZSC:Closed state
ZNS_05.07.02_ZoneMgmtSend-SetZoneDescExt-InvalidTransition.sh	Case 2: If the Select All bit is cleared to ‘0’ and the zone specified by the SLBA field is in any state other than the ZSE:Empty state, the command shall be aborted with a status code of Invalid Zone State Transition
ZNS_05.07.03_ZoneMgmtSend-SetZoneDescExt-InvalidField.sh	Case 3: If the Select All bit is set to ‘1’, then the command shall be aborted with a status Invalid Field in Command
ZNS_05.07.04_ZoneMgmtSend-SetZoneDescExt-DataBuffer.sh	Case 4: On successful command completion, the Zone Descriptor Extension of the zone shall be set to the data in the data buffer
ZNS_05.08.01_ZoneMgmtSend-ErrorCodes.sh	Case 1: Invalid Zone State Transition - verify this error code can be returned
ZNS_05.08.02_ZoneMgmtSend-ErrorCodes-ZoneCapChanged-Set.sh	Case 2: Zone Capacity Changed bit set to 1 - verify the zone capacity has changed due to this command. The host should read the Zone Descriptor data structure for the zone specified by the SLBA field.
ZNS_05.08.03_ZoneMgmtSend-ErrorCodes-ZoneCapChanged-NotSet.sh	Case 3: Zone Capacity Changed bit set to 0 - verify the zone capacity has not changed due to this command
ZNS_06.01.01_ZoneMgmtRcv-ReportZonesPartial-ZRA-0h.sh	Case 1: ZRA Specific field = 0h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.02_ZoneMgmtRcv-ReportZonesPartial-ZRA-1h.sh	Case 2: ZRA Specific field = 1h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.03_ZoneMgmtRcv-ReportZonesPartial-ZRA-2h.sh	Case 3: ZRA Specific field = 2h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.04_ZoneMgmtRcv-ReportZonesPartial-ZRA-3h.sh	Case 4: ZRA Specific field = 3h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.05_ZoneMgmtRcv-ReportZonesPartial-ZRA-4h.sh	Case 5: ZRA Specific field = 4h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.06_ZoneMgmtRcv-ReportZonesPartial-ZRA-5h.sh	Case 6: ZRA Specific field = 5h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.01.09_ZoneMgmtRcv-ReportZonesPartial-ZRA-Rsvd.sh	Case 9: ZRA Specific field = 8h to FFh - verify command fails, likely with INVALID_FIELD
ZNS_06.02.01_ZoneMgmtRcv-ReportZonesFull-ZRA-0h.sh	Case 1: ZRA Specific field = 0h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.02.02_ZoneMgmtRcv-ReportZonesFull-ZRA-1h.sh	Case 2: ZRA Specific field = 1h - verify all zones are listed in ascending order based on ZSLBA value

ZNS_06.02.03_ZoneMgmtRcv-ReportZonesFull-ZRA-2h.sh	Case 3: ZRA Specific field = 2h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.02.04_ZoneMgmtRcv-ReportZonesFull-ZRA-3h.sh	Case 4: ZRA Specific field = 3h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.02.05_ZoneMgmtRcv-ReportZonesFull-ZRA-4h.sh	Case 5: ZRA Specific field = 4h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.02.06_ZoneMgmtRcv-ReportZonesFull-ZRA-5h.sh	Case 6: ZRA Specific field = 5h - verify all zones are listed in ascending order based on ZSLBA value
ZNS_06.02.09_ZoneMgmtRcv-ReportZonesFull-ZRA-Rsvd.sh	Case 9: ZRA Specific field = 8h to FFh - verify command fails, likely with INVALID_FIELD
ZNS_06.05.01_ZoneMgmtRcv-ReportZones-After-NSSR.sh	Zone states change or not after NSSR
ZNS_06.05.02_ZoneMgmtRcv-ReportZones-After-FLR.sh	Zone states change or not after FLR
ZNS_06.05.03_ZoneMgmtRcv-ReportZones-After-Controller-Reset.sh	Zone states change or not after controller reset
ZNS_06.05.04_ZoneMgmtRcv-ReportZones-After-Conventional-Reset.sh	Zone states change or not after conventional reset
ZNS_06.05.05_ZoneMgmtRcv-ReportZones-After-Power-Cycle.sh	Zone states change or not after power cycle
ZNS_06.05.06_ZoneMgmtRcv-ReportZones-After-Link-Cycle.sh	Zone states change or not after link up/down
ZNS_06.03.01_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-0h.sh	Case 1: ZRA Specific field = 0h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.02_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-1h.sh	Case 2: ZRA Specific field = 1h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.03_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-2h.sh	Case 3: ZRA Specific field = 2h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.04_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-3h.sh	Case 4: ZRA Specific field = 3h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.05_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-4h.sh	Case 5: ZRA Specific field = 4h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.06_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-5h.sh	Case 6: ZRA Specific field = 5h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.03.09_ZoneMgmtRcv-ExtReportZonesPartial-ZRA-Rsvd.sh	Case 9: ZRA Specific field = 8h to FFh - verify command fails, likely with INVALID_FIELD
ZNS_06.04.01_ZoneMgmtRcv-ExtReportZonesFull-ZRA-0h.sh	Case 1: ZRA Specific field = 0h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.02_ZoneMgmtRcv-ExtReportZonesFull-ZRA-1h.sh	Case 2: ZRA Specific field = 1h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.03_ZoneMgmtRcv-ExtReportZonesFull-ZRA-2h.sh	Case 3: ZRA Specific field = 2h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.04_ZoneMgmtRcv-ExtReportZonesFull-ZRA-3h.sh	Case 4: ZRA Specific field = 3h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.05_ZoneMgmtRcv-ExtReportZonesFull-ZRA-4h.sh	Case 5: ZRA Specific field = 4h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.06_ZoneMgmtRcv-ExtReportZonesFull-ZRA-5h.sh	Case 6: ZRA Specific field = 5h - verify all zones are listed in ascending order based on ZSLBA value, or INVALID_FIELD is returned
ZNS_06.04.09_ZoneMgmtRcv-ExtReportZonesFull-ZRA-Rsvd.sh	Case 9: ZRA Specific field = 8h to FFh - verify command fails, likely with INVALID_FIELD
ZNS_07.01.01_ZoneAppend.sh	Case 1: Verify command is successful
ZNS_07.01.02_ZoneAppend-InvalidField-1.sh	Case 2: If the zone which the Zone Append command specifies is not of zone type Sequential Write Required, then the command shall be aborted with a status code of Invalid Field in Command

ZNS_07.01.03_ZoneAppend-InvalidField-2.sh	Case 3: If the ZSLBA field in the Zone Append command does To Be Donet specify the lowest logical block for a zone, then the command shall be aborted with a status code of Invalid Field in Command
ZNS_07.03.01_ZoneAppend-ZoneBoundaryError.sh	Case 1: Zone Boundary Error - verify this error code can be returned
ZNS_07.03.02_ZoneAppend-ZoneFull.sh	Case 2: Zone Is Full - verify this error code can be returned
ZNS_07.03.05_ZoneAppend-TooManyActiveZones.sh	Case 5: Too Many Active Zones - verify this error code can be returned
ZNS_07.03.06_ZoneAppend-TooManyOpenZones.sh	Case 6: Too Many Open Zones - verify this error code can be returned

4.2 TP4056 Qualification

TP4056 adds support for namespace types. SANBlaze has developed 22 scripts to cover the TP4056 qualification.

Script Name	Test Description
ZNS_03.01.01_RegCAP-CSS_RegCC-CSS.sh	Offset 0h: CAP – Controller Capabilities Offset 14h: CC – Controller Configuration 1. Get bits 37 & 43 of CAP.CSS, report values to user 2. Get bits 06:04 of CC.CSS, report to user. Based on bits 37 & 43 of CAP.CSS, CC.CSS value should make sense.
ZNS_01.02.01_GetFeatures.sh	1. Do Get Features for FID=19h and verify it's successful
ZNS_01.03.01_SetFeatures.sh	1. Do Set Features for FID=19h and verify it's successful
ZNS_01.01.04_IdentifyNamespace.sh ZNS_01.01.05_IdentifyNamespaceIDs.sh ZNS_01.01.06_IdentifyNSIdentDescript.sh ZNS_01.01.07_IdentifyNVMSetList.sh ZNS_01.01.08_IdentifyZNSNamespaceIDs.sh ZNS_01.01.09_IdentifyAllocatedNSIDs.sh ZNS_01.01.10_IdentifyAllocatedNS.sh ZNS_01.01.11_IdentifyAttachContIDs.sh ZNS_01.01.12_IdentifyControllerIDs.sh ZNS_01.01.13_IdentifyPrimaryContCap.sh ZNS_01.01.14_IdentifySecondaryContList.sh ZNS_01.01.15_IdentifyNSGranularityList.sh ZNS_01.01.16_IdentifyUUIDList.sh ZNS_01.01.17_IdentifyDomainList.sh ZNS_01.01.18_IdentifyEnduranceGroupList.sh ZNS_01.01.19_IdentifyZNSAllocatedNSIDs.sh ZNS_01.01.1A_IdentifyZNSAllocatedNS.sh ZNS_01.01.1B_IdentifyIOCommandSet.sh	1. Do Identify for each supported CNS with CSI=2h (zoned NS) and verify it's successful
ZNS_04.01.80_ErrorCodes-GenericCmd-80h.sh	Generic Command Status Definition 1. Issue a command in order to have the specific error code returned 2. Repeat for each applicable error code

4.3 TP4076 Qualification

TP4076 defines Zoned Random Write Area (ZRWA) and the associated commit operations. SANBlaze developed 26 scripts to cover the TP4076 qualification.

Script Name	Test Description
ZNS_11.01.01_IdentifyZNSNamespace.sh	Identify Namespace Data Structure (CNS 05h). Report the values for the ZRWA fields
ZNS_11.01.02_IdentifyZNSController.sh	Identify Controller Data Structure (CNS 06h). Report the values for the ZRWA fields
ZNS_11.02.01_CommitZoneImplicit.sh	Case 1: 1. Create zone with ZRWA 2. Get write pointer value 3. Do writes and implicit commit 4. Verify write pointer value changed appropriately
ZNS_11.02.02_CommitZoneImplicit-NotMultipleZRWACG.sh	Case 2: The number of logical blocks to be committed doesn't have to be a multiple of Random Write Area Commit Granularity (ZRWACG). If the amount of data to be committed is not an integral multiple of ZRWACG but meet MDTS and write alignment requirement, then the controller shall pass in this command and WP move as expected.
ZNS_11.02.05_CommitZoneImplicit-NotInRange.sh	Case 5: The controller shall abort write operations that specify a Starting LBA field that is not within the ZRWA or the ICR range with a status code of Zone Invalid Write
ZNS_11.02.06_CommitZoneImplicit-NoWPChange.sh	Case 6: 1. Create zone with ZRWA 2. Get write pointer value 3. Do writes but no implicit commit 4. Transition zone to ZSF and verify write pointer value didn't change when number of LBAs written is not multiple of ZRWACG but that data was written (the WP may/shall? change if the number of LBAs written is multiple of ZRWACG although the WP may be undefined value when in ZSF state).
ZNS_11.03.01_CommitZoneExplicit.sh	Case 1: 1. Verify explicit commit is supported 2. Create zone with ZRWA 3. Get write pointer value 4. Do writes and explicit commit 5. Verify write pointer value changed appropriately
ZNS_11.03.02_CommitZoneExplicit-NotMultipleZRWACG.sh	Case 2: The number of logical blocks to be committed shall be a multiple of Random Write Area Commit Granularity (ZRWACG). if the amount of data to be committed is not an integral multiple of ZRWACG, then the controller may <Ed Note: shall?> abort the command with a status of Invalid Field in Command
ZNS_11.03.03_CommitZoneExplicit-NotOpened-NoZRWA.sh	Case 3: When a Commit Zone Send Zone Action is requested, if the specified zone is not: a) in the ZSEO:Explicitly Opened or the ZSIO:Implicitly Opened state; and b) associated with a ZWRA, then the controller shall fail the command with an error status of Invalid Zone Operation Request
ZNS_11.03.04_CommitZoneExplicit-SelectAll-1.sh	Case 4: If the Select All bit is set to ‘1’, the command shall abort the command with status Invalid Field in Command
ZNS_11.03.05_CommitZoneExplicit-CrossZRWABoundary.sh	Case 5: The range to be committed shall not cross the ZRWA boundary. If the Commit Zone Send Zone Action operation attempts to cross the ZRWA boundary, then the command shall be aborted with a status code of Zone Boundary Error
ZNS_11.03.06_CommitZoneExplicit-NoWPChange.sh	Case 6: 1. Create zone with ZRWA 2. Get write pointer value 3. Do writes but no explicit commit 4. Transition zone to ZSF and verify write pointer value didn't change but that data was written

ZNS_11.04.01_ZoneMgmtSend-ZRWAA-1.sh	Case 1: If the Zone Send Action (ZSA) field specifies Open Zone and no ZRWA is currently associated with this zone: • If this bit is set to '1' and If a ZRWA resource is available, then a ZRWA shall be allocated to this zone upon transitioning to the ZSEO:Explicitly Opened state if: o The zone is in the ZSE:Empty state; or o The write pointer is on a Zone Random Write Area Commit Granularity boundary and the zone is in the ZSIO:Implicitly Opened state
ZNS_11.04.02_ZoneMgmtSend-ZRWAA-0.sh	Case 2: If the Zone Send Action (ZSA) field specifies Open Zone and no ZRWA is currently associated with this zone: • If this bit is cleared to '0', then no ZRWA shall be allocated to this zone upon transitioning to the ZSEO:Explicitly Opened state.
ZNS_11.04.03_ZoneMgmtSend-ZRWA-Remove-Full.sh	Case 3: 1. Allocate ZRWA 2. Transition zone to ZSF:Full 3. Verify ZRWA is removed
ZNS_11.04.04_ZoneMgmtSend-ZRWA-Remove-Empty.sh	Case 4: 1. Allocate ZRWA 2. Transition zone to ZSE:Empty 3. Verify ZRWA is removed
ZNS_11.04.07_ZoneMgmtSend-ZRWAA-Ignore.sh	Case 7: If the Zone Send Action (ZSA) field specifies Open Zone, the ZRWAA bit is set to 1, and a ZRWA is currently associated with this zone, the request to allocate a ZRWA resource shall be ignored
ZNS_11.04.08_ZoneMgmtSend-ZRWAA-1-InvalidZoneStateTransition.sh	Case 8: If the Zone Send Action (ZSA) field specifies Open Zone, the ZRWAA bit is set to 1, no ZRWA is currently associated with this zone, and the zone is in the ZSEO:Explicitly Opened state , then the controller shall abort the command with a status of Invalid Zone State Transition
ZNS_11.04.09_ZoneMgmtSend-ZRWAA-0-InvalidZoneStateTransition.sh	Case 9: If the Zone Send Action (ZSA) field specifies Open Zone, the ZRWAA bit is set to 0, and a ZRWA is currently associated with this zone, then the controller shall abort the command with a status of Invalid Zone State Transition
ZNS_11.04.10_ZoneMgmtSend-ZRWA-NotCreated.sh	Case 10: Set ZSA to anything but Open Zone and verify ZRWA isn't created
ZNS_11.05.01_ZoneMgmtRcv-ZRWAA-0.sh	Case 1: 1. Create zone 2. Do Zone Management Receive 3. Verify ZRWAA is 0
ZNS_11.05.02_ZoneMgmtRcv-ZRWAA-1.sh	Case 2: 1. Create zone and setup ZRWA 2. Do Zone Management Receive 3. Verify ZRWAA is 1
ZNS_11.06.02_ZRWA_Associate_More_Than_One_Zone-Fails.sh	Case 2: 1. Create one ZRWA almost the end of current zone and try to associate with next zone. 2. Verify it fails
ZNS_11.06.03_ZRWA-VerifyWrites.sh	Case 3: 1. Create one ZRWA and associate with zone. 2. Write data to it with pattern 1 3. Verify pattern 1 was written 4. Write data to it with pattern 2 5. Verify pattern 2 was written
ZNS_11.06.05_ZRWA-StartLBALessThanWP.sh	Case 5: If a write operation specifies a Starting LBA that is less than the write pointer, then the controller shall abort the command with a status code of Zone Invalid Write
ZNS_11.07.02_ZRWA-WritePointerMisalign.sh	Case 2: If allocation of a ZRWA fails due to write pointer misalignment (refer to Figure 30, ZRWAA field definition), then the command shall fail with status Zone Random Write Area Allocation Failed

4.4 I/O and Others

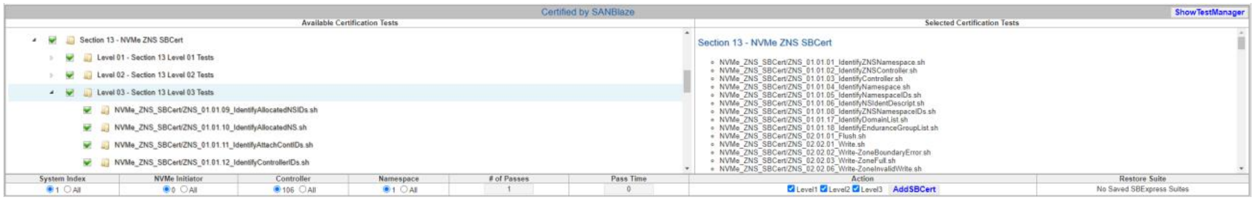
SANBlaze has developed 4 I/O scripts and 5 other scripts to run write/read/compare tests across all zones to ensure data integrity, as well as check I/O performance with a specific number of threads and transfer sizes.

Script Name	Test Description
ZNS_10.01.01_RunIO.sh	Runs I/O for a few seconds, reports data
ZNS_10.01.02_RunIO.sh	Runs I/O with MOR number of threads and cover all zones, reports data
ZNS_10.01.03_RunIO.sh	Runs Write and Read operation with different number of threads (<=MOR number of threads or max at 256 threads), walk through all different transfer size, reports data
ZNS_10.01.04_RunIO.sh	Runs I/O (read/write/compare) with MOR number of threads and weighted round robin I/O queues like auto, round robin, urgent, high priority, medium priority and low priority.
ZNS_08.01.01_MAR-MOR-Verify.sh	Do zone management send and receive and verify MOR <= MAR
ZNS_08.01.02_MAR-MOR-NumZonesZSEO.sh	Do Zone Management Send with ZSA=3h and get number of zones with state ZSEO
ZNS_08.01.03_MAR-MOR-TooManyActiveZones.sh	Test for ZONE_TOO_MANY_ACTIVE
ZNS_09.01.01_Transition-ZSE-ZSIO.sh	Transition from ZSE to ZSIO
ZNS_09.01.02_Transition-ZSC-ZSIO.sh	Transition from ZSC to ZSIO

5. ZNS Qualification with SANBlaze sb_cert

5.1 Choose ZNS Scripts to Run

You can run the ZNS suite from SANBlaze SBExpress Manager GUI under sb_cert as follows:



Available Certification Tests:

- Section 13 - NVMe ZNS SBCert
  - Level 01 - Section 13 Level 01 Tests
  - Level 02 - Section 13 Level 02 Tests
  - Level 03 - Section 13 Level 03 Tests
    - NVMe_ZNS_SBCert/ZNS_01.01.09_IdentifyAllocatedNSIDs.sh
    - NVMe_ZNS_SBCert/ZNS_01.01.10_IdentifyAllocatedNS.sh
    - NVMe_ZNS_SBCert/ZNS_01.01.11_IdentifyAttachContIDs.sh
    - NVMe_ZNS_SBCert/ZNS_01.01.12_IdentifyControllerIDs.sh

Section 13 - NVMe ZNS SBCert

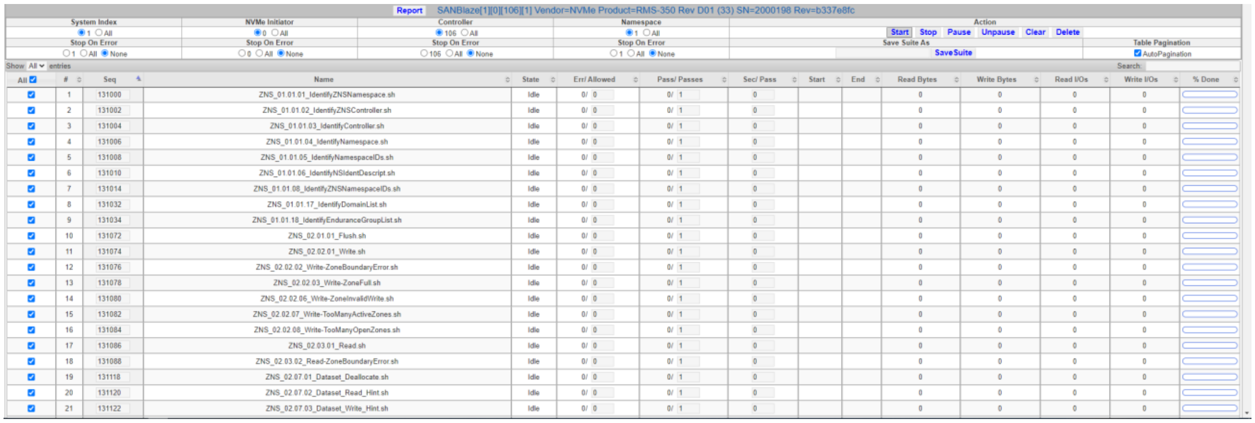
- NVMe_ZNS_SBCert/ZNS_01.01.01_IdentifyZNSNamespace.sh
- NVMe_ZNS_SBCert/ZNS_01.01.02_IdentifyZNSController.sh
- NVMe_ZNS_SBCert/ZNS_01.01.03_IdentifyController.sh
- NVMe_ZNS_SBCert/ZNS_01.01.04_IdentifyNamespace.sh
- NVMe_ZNS_SBCert/ZNS_01.01.05_IdentifyNamespaceIDs.sh
- NVMe_ZNS_SBCert/ZNS_01.01.06_IdentifyNSIdentDescript.sh
- NVMe_ZNS_SBCert/ZNS_01.01.08_IdentifyZNSNamespaceIDs.sh
- NVMe_ZNS_SBCert/ZNS_01.01.17_IdentifyDomainList.sh
- NVMe_ZNS_SBCert/ZNS_01.01.18_IdentifyEnduranceGroupList.sh
- NVMe_ZNS_SBCert/ZNS_02.01.01_Flush.sh
- NVMe_ZNS_SBCert/ZNS_02.02.01_Write.sh
- NVMe_ZNS_SBCert/ZNS_02.02.02_Write-ZoneBoundaryError.sh
- NVMe_ZNS_SBCert/ZNS_02.02.03_Write-ZoneFull.sh
- NVMe_ZNS_SBCert/ZNS_02.02.06_Write-ZoneInvalidWrite.sh

System Index	NVMe Initiator	Controller	Namespace	# of Passes	Pass Time	Action	Restore Suite
● 1 ○ All	● 0 ○ All	● 106 ○ All	● 1 ○ All	1	0	☑ Level1 ☑ Level2 ☑ Level3 AddSBCert	No Saved SBExpress Suites

Figure 1: ZNS Scripts Selection

We categorized all 175 ZNS scripts at 3 levels. With the Level 1 scripts, we expect all ZNS drives to pass. With Level 2 scripts, we expect consumer ZNS drives to pass some of them but enterprise ZNS drives to pass all. Level 3 scripts are the most challenging to pass and even enterprise ZNS drives may not pass all Level 3 scripts.

After choosing the ZNS scripts to run, click the **AddSBCert** button and the selected scripts will be added into the SBExpress test window ready to run:



SANBlaze[1][0][106][1] Vendor=NVMe Product=RMS-350 Rev D01 (33) SN=2000198 Rev=b337e8fc

System Index	NVMe Initiator	Controller	Namespace	Action	
● 1 ○ All	● 0 ○ All	● 106 ○ All	● 1 ○ All		
Stop On Error	Stop On Error	Stop On Error	Stop On Error	Save Suite As	Table Pagination
○ 1 ○ All ● None	○ 0 ○ All ● None	○ 106 ○ All ● None	○ 1 ○ All ● None		☑ AutoPagination

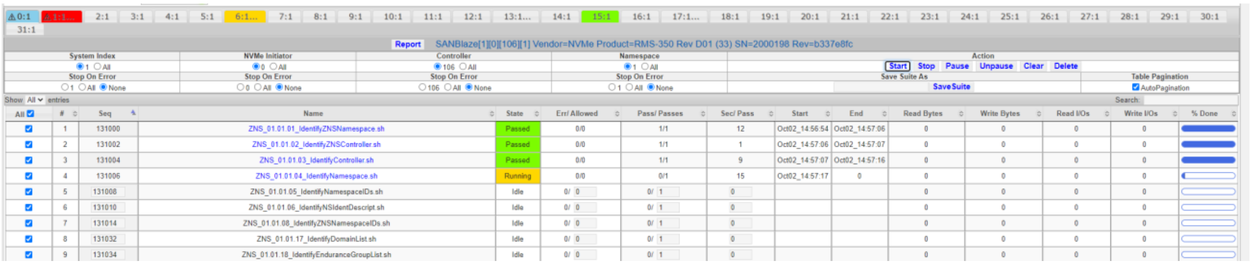
Show All entries

All ☑	#	Seq	Name	State	Err/ Allowed	Pass/ Passes	Sec/ Pass	Start	End	Read Bytes	Write Bytes	Read I/Os	Write I/Os	% Done
☑	1	131000	ZNS_01.01.01_IdentifyZNSNamespace.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	2	131002	ZNS_01.01.02_IdentifyZNSController.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	3	131004	ZNS_01.01.03_IdentifyController.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	4	131006	ZNS_01.01.04_IdentifyNamespace.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	5	131008	ZNS_01.01.05_IdentifyNamespaceIDs.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	6	131010	ZNS_01.01.06_IdentifyNSIdentDescript.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	7	131014	ZNS_01.01.08_IdentifyZNSNamespaceIDs.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	8	131032	ZNS_01.01.17_IdentifyDomainList.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	9	131034	ZNS_01.01.18_IdentifyEnduranceGroupList.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	10	131072	ZNS_02.01.01_Flush.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	11	131074	ZNS_02.02.01_Write.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	12	131076	ZNS_02.02.02_Write-ZoneBoundaryError.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	13	131078	ZNS_02.02.03_Write-ZoneFull.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	14	131080	ZNS_02.02.06_Write-ZoneInvalidWrite.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	15	131082	ZNS_02.02.07_Write-TooManyActiveZones.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	16	131084	ZNS_02.02.08_Write-TooManyOpenZones.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	17	131086	ZNS_02.03.01_Read.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	18	131088	ZNS_02.03.02_Read-ZoneBoundaryError.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	19	131118	ZNS_02.07.01_Dataset_Deallocate.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	20	131120	ZNS_02.07.02_Dataset_Read_Hint.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	21	131122	ZNS_02.07.03_Dataset_Write_Hint.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	

Figure 2: ZNS Scripts Added

5.2 Start Testing

Once all selected ZNS scripts have been added into SBExpress test window, click **Start** to begin testing. The GUI will show the progress of the running tests as follows:



SANBlaze[1][0][106][1] Vendor=NVMe Product=RMS-350 Rev D01 (33) SN=2000198 Rev=b337e8fc

System Index	NVMe Initiator	Controller	Namespace	Action	
● 1 ○ All	● 0 ○ All	● 106 ○ All	● 1 ○ All		
Stop On Error	Stop On Error	Stop On Error	Stop On Error	Save Suite As	Table Pagination
○ 1 ○ All ● None	○ 0 ○ All ● None	○ 106 ○ All ● None	○ 1 ○ All ● None		☑ AutoPagination

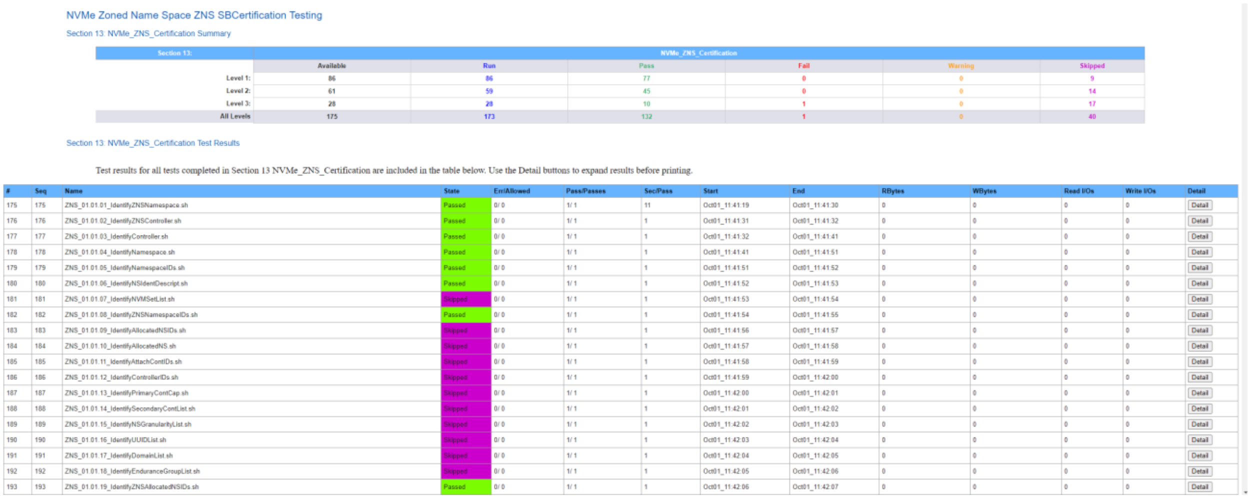
Show All entries

All ☑	#	Seq	Name	State	Err/ Allowed	Pass/ Passes	Sec/ Pass	Start	End	Read Bytes	Write Bytes	Read I/Os	Write I/Os	% Done
☑	1	131000	ZNS_01.01.01_IdentifyZNSNamespace.sh	Passed	0/0	1/1	12	Oct02_14:56:54	Oct02_14:57:06	0	0	0	0	
☑	2	131002	ZNS_01.01.02_IdentifyZNSController.sh	Passed	0/0	1/1	1	Oct02_14:57:06	Oct02_14:57:07	0	0	0	0	
☑	3	131004	ZNS_01.01.03_IdentifyController.sh	Passed	0/0	1/1	9	Oct02_14:57:07	Oct02_14:57:16	0	0	0	0	
☑	4	131006	ZNS_01.01.04_IdentifyNamespace.sh	Running	0/0	0/1	15	Oct02_14:57:17	0	0	0	0	0	
☑	5	131008	ZNS_01.01.05_IdentifyNamespaceIDs.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	6	131010	ZNS_01.01.06_IdentifyNSIdentDescript.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	7	131014	ZNS_01.01.08_IdentifyZNSNamespaceIDs.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	8	131032	ZNS_01.01.17_IdentifyDomainList.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	
☑	9	131034	ZNS_01.01.18_IdentifyEnduranceGroupList.sh	Idle	0/ 0	0/ 1	0			0	0	0	0	

Figure 3: ZNS scripts running

5.3 Test Results Review

You can view the test results “on the fly” by clicking the script name in the GUI above, or generate the results report by clicking the **Report** button, and then all checked script test results will show up in one HTML report file as follows:



NVMe Zoned Name Space ZNS SBCertification Testing

Section 13: NVMe_ZNS_Certification Summary

Section 13:	NVMe_ZNS_Certification					
	Available	Run	Pass	Fail	Warning	Skipped
Level 1:	86	86	77	0	0	9
Level 2:	61	59	45	0	0	14
Level 3:	28	28	10	1	0	17
All Levels	175	173	132	1	0	40

Section 13: NVMe_ZNS_Certification Test Results

Test results for all tests completed in Section 13 NVMe_ZNS_Certification are included in the table below. Use the Detail buttons to expand results before printing.

#	Seq	Name	State	Err/Allowed	Pass/Passes	Sec/Pass	Start	End	RBytes	WBytes	Read I/Os	Write I/Os	Detail
175	175	ZNS_01.01.01_IdentifyZNSNamespace.sh	Passed	0/ 0	1/ 1	11	Oct01_11:41:19	Oct01_11:41:30	0	0	0	0	Detail
176	176	ZNS_01.01.02_IdentifyZNSController.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:31	Oct01_11:41:32	0	0	0	0	Detail
177	177	ZNS_01.01.03_IdentifyController.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:32	Oct01_11:41:41	0	0	0	0	Detail
178	178	ZNS_01.01.04_IdentifyNamespace.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:41	Oct01_11:41:51	0	0	0	0	Detail
179	179	ZNS_01.01.05_IdentifyNamespaceIDs.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:51	Oct01_11:41:52	0	0	0	0	Detail
180	180	ZNS_01.01.06_IdentifyNSIdentDescript.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:52	Oct01_11:41:53	0	0	0	0	Detail
181	181	ZNS_01.01.07_IdentifyNVMSetList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:41:53	Oct01_11:41:54	0	0	0	0	Detail
182	182	ZNS_01.01.08_IdentifyZNSNamespaceIDs.sh	Passed	0/ 0	1/ 1	1	Oct01_11:41:54	Oct01_11:41:55	0	0	0	0	Detail
183	183	ZNS_01.01.09_IdentifyAllocatedNSIDs.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:41:56	Oct01_11:41:57	0	0	0	0	Detail
184	184	ZNS_01.01.10_IdentifyAllocatedNS.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:41:57	Oct01_11:41:58	0	0	0	0	Detail
185	185	ZNS_01.01.11_IdentifyAttachContIDs.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:41:58	Oct01_11:41:59	0	0	0	0	Detail
186	186	ZNS_01.01.12_IdentifyControllerIDs.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:41:59	Oct01_11:42:00	0	0	0	0	Detail
187	187	ZNS_01.01.13_IdentifyPrimaryContCap.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:00	Oct01_11:42:01	0	0	0	0	Detail
188	188	ZNS_01.01.14_IdentifySecondaryContList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:01	Oct01_11:42:02	0	0	0	0	Detail
189	189	ZNS_01.01.15_IdentifyNSGranularityList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:02	Oct01_11:42:03	0	0	0	0	Detail
190	190	ZNS_01.01.16_IdentifyUUIDList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:03	Oct01_11:42:04	0	0	0	0	Detail
191	191	ZNS_01.01.17_IdentifyDomainList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:04	Oct01_11:42:05	0	0	0	0	Detail
192	192	ZNS_01.01.18_IdentifyEnduranceGroupList.sh	Skipped	0/ 0	1/ 1	1	Oct01_11:42:05	Oct01_11:42:06	0	0	0	0	Detail
193	193	ZNS_01.01.19_IdentifyZNSAllocatedNSIDs.sh	Passed	0/ 0	1/ 1	1	Oct01_11:42:06	Oct01_11:42:07	0	0	0	0	Detail

Figure 4: ZNS Scripts Results Report

You can click the **Detail** button at the end of each row to display or hide the detailed test results.

Summary

In summary, SANBlaze supports all of the Zoned Admin Command Sets and Zoned I/O Command Sets as specified and defined in the latest spec (*TP 4053 2020.06.15, TP4056 2020.06.15, TP4076 2020.08.04).* SANBlaze provides written scripts that can be run right of the box in our SBExpress GUI, as well as run through our command line interface, XML API interface, and Python wrapped API interface for test automation. SANBlaze is proud to provide a high quality and simple way to test and qualify ZNS for your SSD drives.