SBExpress Environmental Control Program and Remote Control of SBExpress Test Page sb_i2c

REST API

Vince Asbridge
Version V0.8

October 12, 2020

Oct. 12, 2020

V0.1 - Original
V0.2 - Added tests control information (add, start, stop, clear, etc.)
V0.3 - Added documentation for adding multiple tests
V0.4 - Added information regarding test status in .csv format
V0.5 - Updated procedure for adding one test
V0.6 - Fixed typo in set_power example
V0.7 - Add SBExpress test suite commands
V0.8 - Add SBExpress saved report commands

Table of Contents

SBExpress NVMe Test Platform REST API

The SANBlaze SBExpress enclosure for testing NVMe drives can be monitored and controlled via a built-in REST API, allowing for the monitoring of system environmental conditions such as fan speed, enclosure temperature, PCIe status and voltage.

This guide documents the use of the REST API for all supported parameters.

REST API

Representational State Transfer (REST) is a method for reading and writing information to a remote system running html. The SANBlaze SBExpress uses a REST Application Programming Interface to read and write environmental parameters from the system. Parameters may be read only or read and write, as defined in each command section below.

Installing REST Client for Firefox

How to install REST Client on Firefox and some example of using it

HTTP Methods Get and Post

The SBExpress REST API interface is accessed using the HTTP GET and Post methods.

Curl is used in this document, but any language capable of issuing the GET and POST methods can be used to set and get parameters.

CURL

curl is an open source tool that can be used to interface from a command line interface (CLI) to a REST based interface. Using the SANBlaze SBExpress REST interface allows users to integrate SBExpress parameters into scripts or custom control programs.

curl examples for each supported parameter are provided below.

Installing CURL

how to install curl on redhat and debian linux and windows

Example GET command using curl

There are four fans in the SANBlaze-DT unit, and six fans in the SANBlaze-RM unit. Read the temperature at each of the fans using the following curl GET command:

```
curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/fan1temp
```

JSON

All responses to REST queries are returned in JavaScript Object Notation (JSON) format. For example, consider the response to a request for the Temperature at fan1.


```
[
    {
        "fan1temp": "31"
    },
    {
        "status": {
            "code": 200,
            "reason": "Success"
        }
    }
]
```


SBExpress REST API

The following settings can be controlled via the REST API. Note all parameters are case sensitive and must be issued exactly as they appear below.

Parameters that affect the entire system are read and written at location:

/rest/sanblazes/N/[parameter]

Where N is the system number (show them how to get it from the web and from REST)

Parameters that affect one drive slot are read and written at location:

/rest/sanblazes/N/ports/P/targets/T/[parameter]

System Parameters

The parameters in this section will affect all slots in the system.

set_load12V

Parameter	set_load12V
Purpose	Set the input voltage for the system in mV. Voltage will be set and measured on the motherboard and will affect all slots
Direction	POST
Data	Positive integer mV
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/set_load12V -d 13000

Return	[
	{
	"set_load12V": "13000"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	The status reflects successful write to the request file

status_set_load12V

Parameter	status_set_load12V
Purpose	Read the progress of the set_load12V command
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/status_set_load12V

Return	[
	{
	"status_set_load12V": " Pass: sb_i2c -n 1 -d -3 -v 12000"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Pass: Command has been successfully executed Fail: Command was attempted but exited with error Pending: Command has been acknowledged and queued to the SBExpress Running: Command is currently running

load12V

Parameter	load12V
Purpose	Read the current input Voltage for the system
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/load12V

Return	[
	{
	"load12V": "11980"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Current power supply voltage measured on the motherboard

fanNrpm

Parameter	fanNrpm Where N = 1 to 6
Purpose	Read the speed of each of the six enclosure fans
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/fan1rpm

Return	[
	{
	"fan1rpm": "5226"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	A reading under 100 indicates the fan is not spinning

fanNtemp

Parameter	fanNtemp Where N = 1 to 6
Purpose	Read the current temperature at each the six enclosure fans
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/fan1temp

Return	[
	{
	"fan1temp": "29"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Temperature in degrees C

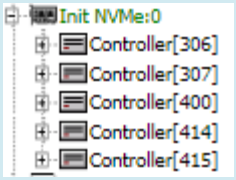
Slot Parameters

The parameters in this section affect only the targeted slot.

load12V

Parameter	load12V
Purpose	Read the current input Voltage in mV at the specified drive slot
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/load12V

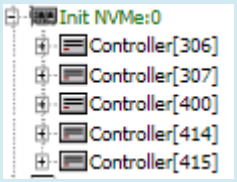
Return	[
	{
	"load12V": "11970"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	The index 400 above is the controller number of the drive under test. This can be read via REST or from the VLUN left hand menu [controller] 

load12I

Parameter	load12I
Purpose	Read the current input Curent in mA at the specified drive slot
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/load12I

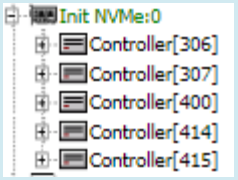
Return	[
	{
	"load12I": "124"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	The index 400 above is the controller number of the drive under test. This can be read via REST or from the VLUN left hand menu [controller] 

load12P

Parameter	load12P
Purpose	Read the Power in mW at the specified drive slot
Direction	GET
Data	N/A
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/load12P

Return	[
	{
	"load12P": "1483"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	The index 400 above is the controller number of the drive under test. This can be read via REST or from the VLUN left hand menu [controller] 

button

Parameter	button
Purpose	Press the Hot Plug Button
Direction	POST
Data	1
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/set_button -d 1

Return	[
	{
	"set_button": "1"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}

	]
Notes	Initiates a controlled hot swap operation by pressing the Hot Plug button on the drive slot

set_power

Parameter	set_power
Purpose	Turn the power to a slot on or off
Direction	POST
Data	-d 0,1
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/set_power -d 0

Return	[
	{
	"set_power": "0"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Removes power from the device without pressing the hot plug button

set_IOX_PORT0_100MS_RST

Parameter	set_IOX_PORT0_100MS_RST
Purpose	Toggle PERST for ~100mS, force hard PCIe reset to device
Direction	POST
Data	-d 0
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/101/set_IOX_PORT0_100MS_RST -d 0

Return	[
	{
	"set_IOX_PORT0_100MS_RST": "0"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Toggle the PERST signal to the device. Resets to 1, 0 to trigger

set_IOX_PORT0_PERST_L

Parameter	set_IOX_PORT0_PERST_L
Purpose	Hold device in PCIe reset
Direction	POST
Data	-d 0,1
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/101/set_IOX_PORT0_PERST_L -d 0

Return	[
	{
	"set_IOX_PORT0_PERST_L": "1"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Write -d 0 to hold device in reset, -d 1 to release reset

set_IOX_DIS_PRSNT0_L

Parameter	set_IOX_DIS_PRSNT0_L
Purpose	Disable the present signal at the device
Direction	POST
Data	-d 0,1
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/101/set_IOX_DIS_PRSNT0_L -d 0

Return	[
	{
	"set_IOX_DIS_PRSNT0_L": "0"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Write -d 0 to remove present from device -d 1 to restore normal operation

sbexpress

Parameter	sbexpress
Purpose	All parameters available from a SBExpress shelf
Direction	GET
Data	n/a
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/sbexpress
Return	[{ "sanblazes": [{ "index": "1", "i2caddress": "0", "load12V": "12432", "min12V": "9604", "max12V": "14484", "fan1fail": "0", "fan1rpm": "2738", "fan1temp": "27", "fan2fail": "0", "fan2rpm": "2998", "fan2temp": "27", "fan3fail": "0", "fan3rpm": "5454", "fan3temp": "32", "fan4fail": "0", "fan4rpm": "5305", "fan4temp": "37", "activenvmeport": "0", "license": "full", "ports": [{ "index": "0", "targets": [{ "index": "100", "PCIeDEV": "0a:00.0", "SECONDARY_BUS": "10", "PRESENCE": "0", "MRL_SENSOR_STATE": "1", "LNK_CUR_SPEED": "1", "LNK_CUR_WIDTH": "0", "POWER": "0", "POWER_LED": "1", "ATTN_LED": "0", "load12V": "16", "load12I": "0", "load12P": "0", "plugged": "1", "quarchSN": "2067-02-024", "ExpressSlot": "0",

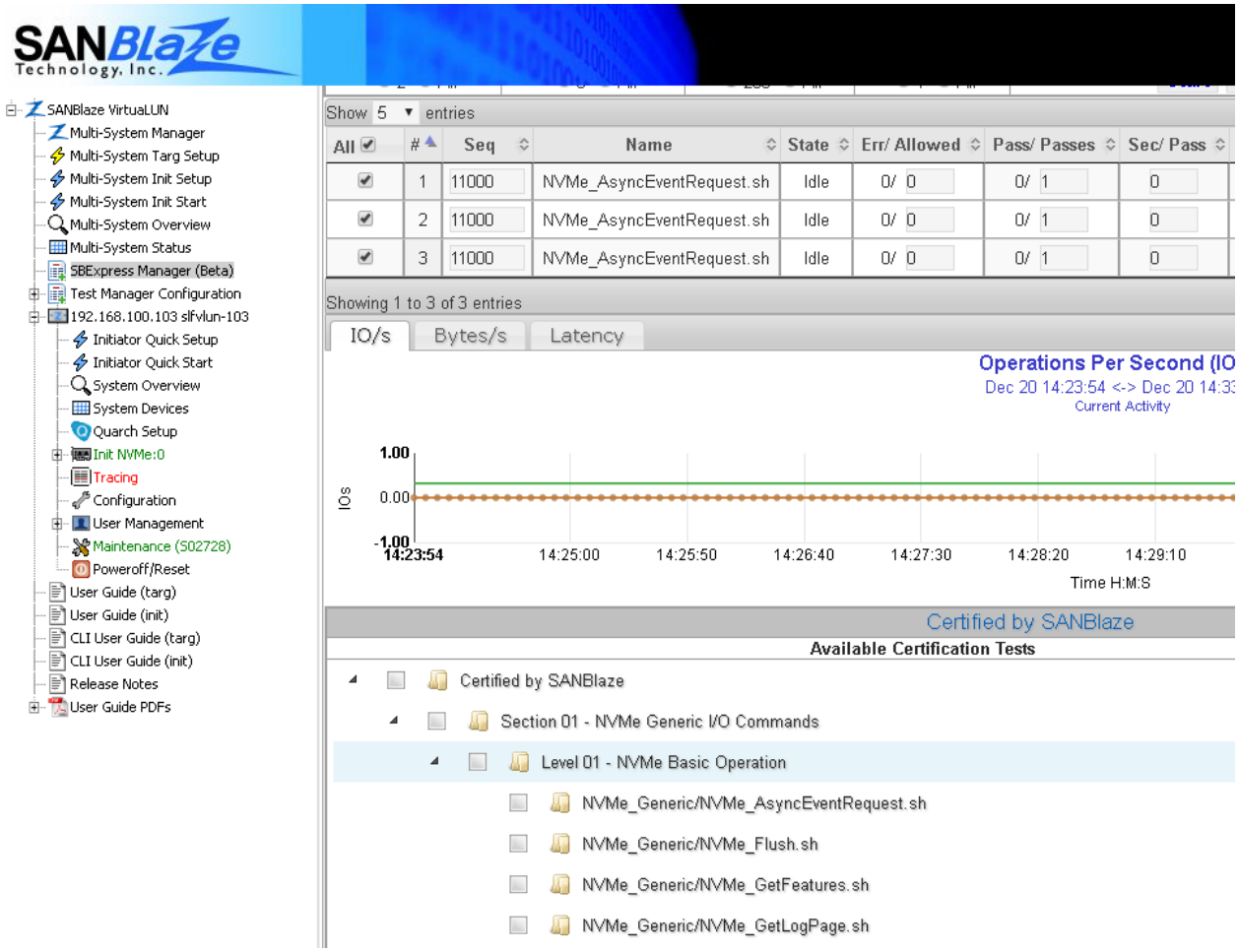
	"luns": [...
Notes	Can be used to query the layout and current status of the SBExpress system

Test Interface

SANBlaze tests can be configured via the REST interface. This section describes the API for the SANBlaze TestManager interface using REST syntax.

Assigning a Test to a Device

Tests are assigned from the test pool to devices in the following way. For a list of available tests, see the SANBlaze Certification section of the SBExpress page. You will see a page similar to this:



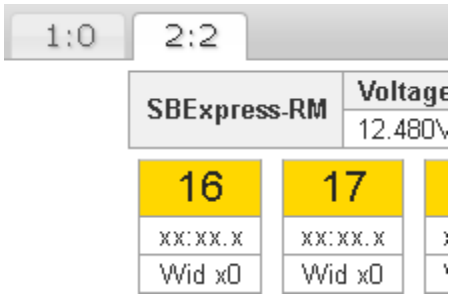
Determine the name of the test you will add, and issue the following HTTP POST to add the test to a selected device:

```
curl -X POST http://192.168.100.103/goform/TestManagerAdd -d
'express=1&ok=Add&system=1&port=0&target=102&lun=1&script="-I 11000 -S 0 -N 1
-f NVMe_Generic/NVMe_AsyncEventRequest.sh"'
```

Where the parameters are as follows:

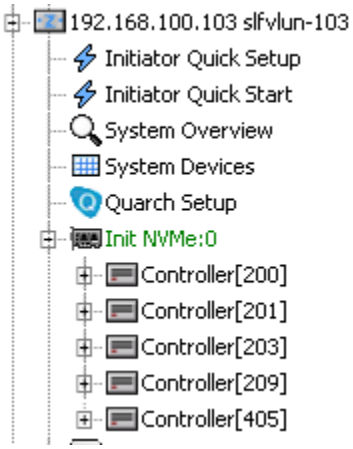
- express: System type, always 1 for SBExpress
- ok: The action to take, in this case Add
- system[1]: System number, see number before : on system tab
- port[2]: NVMe port number, see below
- target[2]: NVMe target number, controller number see below
- lun: NVMe Namespace, starting at 1
- index: Any number >=1. The order the tests will run if sequenced. Tests given the same index will run at the same time.
- passes: Number of times to run the test, default 1
- passtimer: Time to run each pass
- script: Test dir and name of script (see example above)

For SBExpress system number, see tabs on SBExpress GUI page:



[1]System number is the first number before the ":" on the SBExpress GUI page. In the picture above, the 1 and 2 before the ":"

[2]Port number, read from GUI page the number of the Init NVMe bus in the line below "Init NVMe:0"



TestManagerAdd Add Tests to a Device

The following command is used to Add Tests to a device which can then be acted upon (start, stop, etc.). The command will stage one or more tests to a Namespace. It is recommended that one command with a list of tests is issued rather than multiple calls to add single tests one at a time.

Parameter	TestManagerAdd
Purpose	Add one or multiple test scripts to a given port/target/lun
Direction	POST
Data	-d 'express=1&ok=Add&system=1&port=0&target=102&lun=1&script="-I 11000 -S 0 -N 1 -f NVMe_Generic/NVMe_AsyncEventRequest.sh -I 11018 -S 0 -N 1 -f NVMe_Generic/NVMe_Flush.sh -I 11032 -S 0 -N 1 -f NVMe_Generic/NVMe_GetFeatures.sh -I 11034 -S 0 -N 1 -f NVMe_Generic/NVMe_GetLogPage.sh -I 11036 -S 0 -N 1 -f NVMe_Generic/NVMe_IdentifyAllocatedNSIDs.sh"'
Example Adding One Test	curl -X POST http://192.168.100.103/goform/TestManagerAdd -d 'express=1&ok=Add&system=1&port=0&target=102&lun=1&script="-I 11000 -S 0 -N 1 -f NVMe_Generic/NVMe_AsyncEventRequest.sh"'
Example Adding Multiple Tests	curl -X POST http://192.168.100.103/goform/TestManagerAdd -d 'express=1&ok=Add&system=1&port=0&target=102&lun=1&script="-I 11000 -S 0 -N 1 -f NVMe_Generic/NVMe_AsyncEventRequest.sh -I 11018 -S 0 -N 1 -f NVMe_Generic/NVMe_Flush.sh -I 11032 -S 0 -N 1 -f NVMe_Generic/NVMe_GetFeatures.sh -I 11034 -S 0 -N 1 -f NVMe_Generic/NVMe_GetLogPage.sh -I 11036 -S 0 -N 1 -f NVMe_Generic/NVMe_IdentifyAllocatedNSIDs.sh"'

Return	[{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}]

Notes	A second return format can be requested using express=2, which is required for the response to be compatible with Jenkins. Return when express=2 is: <html> <status>200</status>

```
<reason>Success</reason>
</html>
```

Note that in the command above, any number of tests may be added with one call. The scripts are listed with their control parameters in a list after script=. The syntax must be as in the example:

script="-I 11000 -S 0 -N 1 -f NVMe_Generic/NVMe_AsyncEventRequest.sh -I 11018 -S 0 -N 1 -f NVMe_Generic/NVMe_Flush.sh"

Where:

- -I Index Number -1 = "next available, auto assign"
- -S Seconds per pass 0 = "default set by script"
- -N Number of passes 1 = "once"
- -f Filename = Script file name, example NVMe_Generic/NVMe_Flush.sh

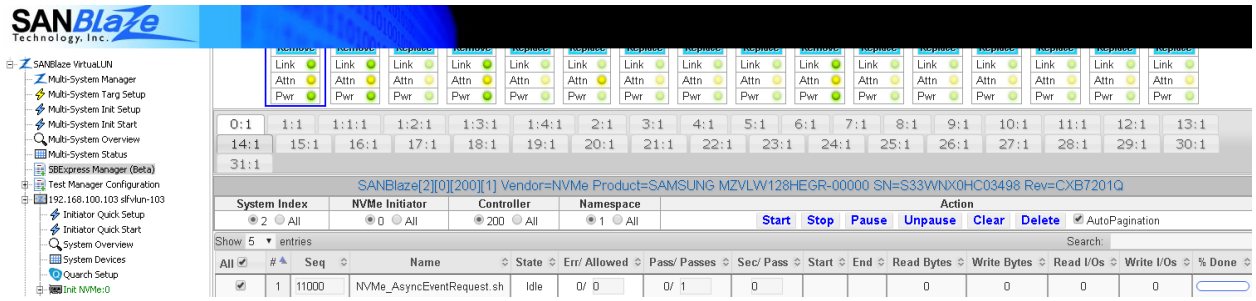
The script names can be found at:

/virtualun/rest/scripts/ under directories for each test category

- Generic_IO
- IO_Tests
- NVMe_DualPort
- NVMe_Dual_Server
- NVMe_Generic
- NVMe_MI
- NVMe_MI_Basic
- NVMe_Misc_Actions
- NVMe_Namespace
- NVMe_Quarch
- NVMe_Resets
- SBExpress
- SSD_Endurance
- V110_IOL_NVMe
- V110_IOL_NVMe_MI

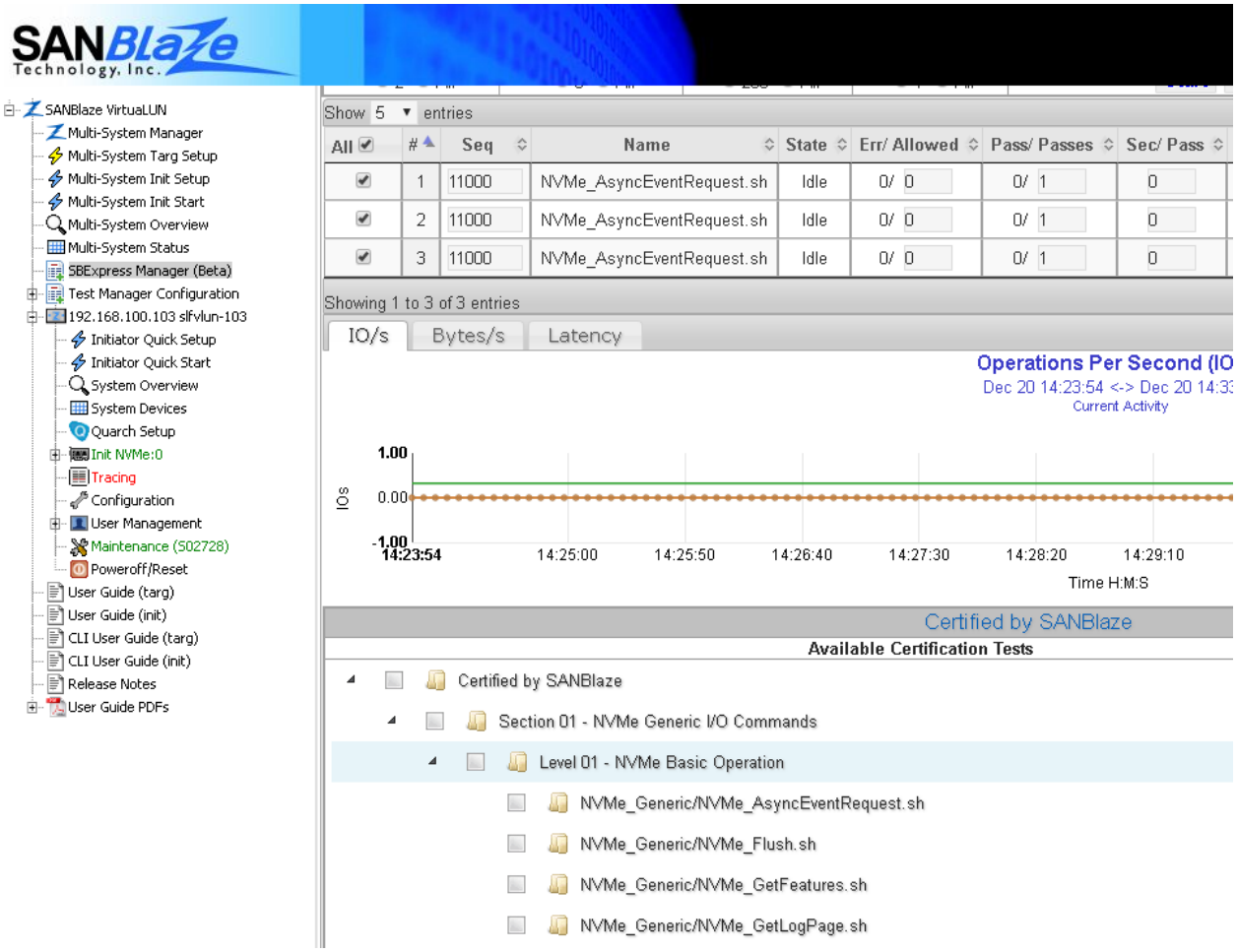
Verify Success for TestManagerAdd

You can verify the success of the above command by browsing to the SBExpress web page (make sure to refresh if you are on the page). You will see your test added to the test list for the NVMe device. See the figure below.



Selecting Tests

Once tests have been assigned to your NVMe device, actions can be taken on a single test or on a group of tests. The scope your actions will have will depend on if the tests are Selected on Unselected. The first column in the table below shows a check mark for each test that is selected. When tests are assigned, they are selected by default. On theSBExpress page, you will see all tests selected by default:



Any action targeted to the tests (see below) will affect all selected tests. To Select or Unselect a test, use the command below:

Select/Unselect All Tests on a Namespace

Parameter	selected
Purpose	Select all tests on a given target
Direction	POST
Data	-d "1" or "0"
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/2/ports/0/targets/200/luns/1/tests/selected -d "1"

Return	[
	{
	"selected": "1"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Selects or unselects all tests on lun1

Select/Unselect A Single Test on a Namespace

Parameter	selected
Purpose	Select all tests on a given target
Direction	POST
Data	-d "1" or "0"
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/2/ports/0/targets/200/luns/1/tests/2/selected -d "1"

Return	[
	{
	"selected": "1"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Selects or unselects a specific test (2 in the example) on lun1

Verify Success for Select/Unselect Test

You can verify the success of the above command by browsing to the SBExpress web page. The page will reflect the change without refreshing, but the change can take up to 20 seconds to be reflected on the page. You will see the check box set or clear for each test to select/unselect.

Performing Actions on Tests

The following actions can be taken on one or more tests that have been assigned to an NVMe device:

- Start: Starts the specified test first then all selected tests in sequence
- Stop: Stops the selected tests
- Pause: Pauses the selected tests
- Unpause: Unpauses the selected tests
- Clear: Deletes the test results of the selected tests (use with care, archive your test results first)
- Delete: Deletes the instance of the test from the NVMe device

Action All Tests on a Namespace

Parameter	requestedscriptstate
Purpose	Request an action on all tests on the given namespace
Direction	POST
Data	-d "start", "stop", "pause", "unpause", "clear" or "delete"
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/2/ports/0/targets/200/luns/1/tests/requestedscriptstate -d "start"

Return	[
	{
	"requestedscriptstate": "start"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Takes action on all selected tests on lun1 Tests will start in sequence order, lowest first Tests with the same sequence number will run together

Action One Test on a Namespace

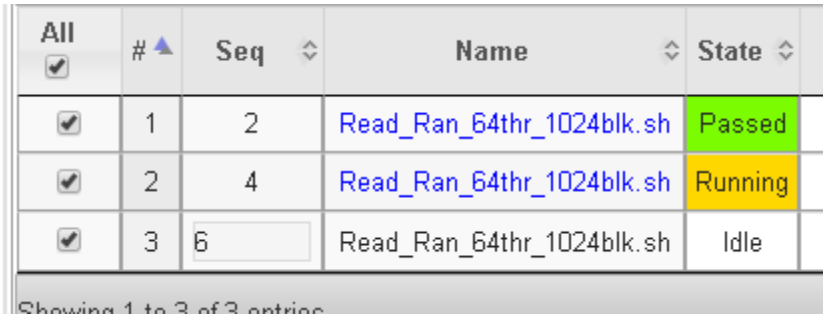
Parameter	requestedscriptstate
Purpose	Request an action on all tests on the given namespace
Direction	POST
Data	-d "start", "stop", "pause", "unpause", "clear" or "delete"
Example	curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/2/ports/0/targets/200/luns/1/tests/2/requestedscriptstate -d "start"

Return	[
	{
	"requestedscriptstate": "start"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Takes action on a single tests (in the example, test 2) on lun1 Specified test will run first, followed by all selected tests in sequence order To run a single test alone, unselect all tests, select the target test, issue start to the test

Verify Success for Actions

You can verify the success of the above command by browsing to the SBExpress web page. The page will reflect the state of the tests without refreshing, but the change can take up to 20 seconds to be reflected on the page. You will see the status of the test change as shown below:



Reading the Status of a Test

The current status of a test can be read as follows.

Read the Current State of a Test

Parameter	scriptstate
Purpose	Request the state of a test
Direction	GET
Data	n/a
Example	curl -X GET http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/2/ports/0/targets/200/luns/1/tests/2/scriptstate

Return	[
	{
	"scriptstate": "Passed"
	},
	{
	"status": {
	"code": 200,
	"reason": "Success"
	}
	}
	]

Notes	Reads the current state of a test

Test Results Data in CSV Format

Note: This support is added in the V8.0 Beta16 release

As SBExpress tests on the system complete, a summary of the test results are kept, and are available to the user.

Test results are available at the following levels:

- Individual Test Directory
- Namespace (LUN) Directory
- Controller (Target) Directory

CSV File at the Individual Test Directory

CSV files for each individual test are stored at the ../tests/# directory, for example:

/rest/sanblazes/1/ports/0/targets/100/luns/SN:S2J4NX0H902718/1/tests/1/Read_Ran_64thr_1024blk.sh.csv

Each file contains one or more runs for the single test.

Typical results:

```
time,hostname,ipaddr,system,port,target,lun,testnumber,sequence,name,status,errors,allowederrors,pass,passes,secsperpass,starttime,endtime,secrun,secspause,bytesread,byteswritten,readios,writeios
```

```
Aug15_14:37:48,VLUN103,192.168.100.103,1,0,100,1,5,21318,Read_Ran_64thr_1blk.sh,Passed,0,0,1,1,60,15658942
08,1565894268,60,0,6055907840,0,11827945,0
Aug15_14:37:48,VLUN103,192.168.100.103,1,0,100,1,5,21318,Read_Ran_64thr_1blk.sh,Passed,0,0,1,1,60,15658942
08,1565894268,60,0,6055907840,0,11827945,0
Aug15_15:44:30,VLUN103,192.168.100.103,1,0,100,1,5,21318,Read_Ran_64thr_1blk.sh,Passed,0,0,1,1,60,15658982
10,1565898270,60,0,1967468032,0,3842711,0
Aug15_15:44:30,VLUN103,192.168.100.103,1,0,100,1,5,21318,Read_Ran_64thr_1blk.sh,Passed,0,0,1,1,60,15658982
10,1565898270,60,0,1967468032,0,3842711,0
```

Where the columns are:

- time - The time this log entry was added to the CSV file
- hostname - Name of the host running the tests
- ipaddr - ip address of the host running the tests
- system - System / Rack number, starts at 1
- port - Port number for NVMe on this system (usually zero)
- target - Target / Controller number the test was run on
- lun - LUN / Namespace the test was run on
- testnumber - Directory number of the test ../tests/N
- sequence - Execution order (sequence number) of the test
- name - Script name for the test
- status - Final status (Pass/Fail/Warning/Skipped)
- errors - Number of errors during test
- allowederrors - Number of errors allowed during test
- pass - The number of the pass of passes (usually 1)
- passes - Number of passes to run for this test
- secsperpass - How many seconds to run each pass (some tests are "one shot" and ignore)
- starttime - Time in seconds since epoch at start of tests
- endtime - Time in seconds since epoch at end of tests
- secrun - Number of seconds the test was actually running (not paused or stopped)
- secspause - Number of seconds the test was paused
- bytesread - Total number of bytes read during the test
- byteswritten - Total number of bytes written during the test
- readios - Number of IOs read during the test
- writeios - Number of IOs written during the test

CSV File at the Namespace (LUN) Level

A cumulative CSV file for all tests run on a specific namespace is at the ../luns/# directory, for example:

/rest/sanblazes/1/ports/0/targets/100/luns/SN:S2J4NX0H902718/1/SANBlaze_1_0_100_1_NamespaceTests.csv

Where: SANBlaze_1_0_100_1_NamespaceTests = system1, port0, controller100, namespace1

Each file contains results from all tests run on the LUN.

CSV File at the Controller (Target) Level

A cumulative CSV file for all tests run on all namespaces is at the ../targets/# directory, for example:

/rest/sanblazes/1/ports/0/targets/100/SANBlaze_1_0_100_ControllerTests.csv

Where: SANBlaze_1_0_100_NamespaceTests = system1, port0, controller100

Each file contains results from all tests run on all Namespaces on the Controller.

Accessing the CSV Files Locally

You can access the csv test files locally or remotely. To access the files on the system running the tests, do the following:

Substitute your system's IPAddress, system (rack) number, usually 1, port usually 0, target, lun information in the commands below

- ssh 192.168.100.103
    - vlun
    - SANBlaze
- su
    - SANBlaze

```
cat /rest/sanblazes/1/ports/0/targets/100/luns/SN:S2J4NX0H902718/1/tests/1/Read_Ran_64thr_1024blk.sh.csv
cat /rest/sanblazes/1/ports/0/targets/100/luns/SN:S2J4NX0H902718/1/SANBlaze_1_0_100_1_NamespaceTests.csv
cat /rest/sanblazes/1/ports/0/targets/100/SANBlaze_1_0_100_ControllerTests.csv
```

Accessing the CSV Files Remotely

You can access the csv test files locally or remotely.

The curl program is open source, it will need to be on the system issuing the commands:

On any system with network access and CURL:

Remote Access the CSV File in Plain Text Format

To access the files remotely in plain TEXT, use the following command syntax.

```
curl -X GET
http://192.168.100.103/rest/sanblazes/1/ports/0/targets/100/SANBlaze_1_0_100_ControllerTests.csv
```

Remote Access the CSV File in JSON Format

```
curl -X GET
http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/100/SANBlaze_1_0_100_ControllerT
ests.csv
```

Accessing Saved SBExpress Suites from REST

This functionality is introduced in V8.2 Beta2

Suites are saved in a format containing all information needed to run the suite and to build the report. The JSON description of a test suite is also used by Jenkins. You can list the available test suites, if any, and get a description of a single suite of tests using the REST commands below.

*** developing currently on 100.130, SS is on 101 with the analyzer **

The sbexpress.cfg format is as follows:

```
cat /rest/sanblazes/saved_suites/3/sbexpress.cfg
#
#    sbexpress.cfg script suite
#
#    Created by sbexpress on Tue Sep  8 16:05:52 EDT 2020
#
#    Data in this .cfg defines tests for a given suite and is used
#    to load tests to SBExpress devices and Jenson Jobs
#
#    You can modify this file using the given format.  All comments
#    must start with # in the first column, comments within a line
#    are not valid
#
#    The CSV data is in the following format:
# section,level,sequence,passes,passtime,name
#
#     StopOn can be none,this,all
#
NAME=SuiteName
VENDOR=FastNVMeCo
COMPANY=FastNVMECo Inc.
ENGINEER=Vincent Asbridge
ADDRESS=1 Monarch Dr Littleton MA
PHONE=978-897-1888
STOPONSYS=none
STOPONPORT=none
STOPONTARGET=none
STOPONLUN=none
2,1,0,1,60,/virtualun/rest/scripts/IO_Tests/Read_Seq_64thr_1blk.sh
2,2,2,1,60,/virtualun/rest/scripts/IO_Tests/Write_Seq_64thr_1024blk.sh
2,3,4,1,60,/virtualun/rest/scripts/IO_Tests/Compare_Seq_64thr_1024blk.sh
3,2,0,1,0,/virtualun/rest/scripts/NVMe_Resets/NVMe_Controller_Reset.sh
2,1,6,1,60,/virtualun/rest/userscripts/IO_Tests/MyExampleScript.sh
```

New REST Commands for Test Suites

The following REST commands are used to get list of available test suites and info on a specific suite:

List All Available Suites

```
curl -X POST
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/saved_suites -d 0

    { "saved_suites": [
            {
               "index": "1",
               "suitename": "SuiteName"
            ]},
            {
               "index": "2",
               "suitename": "SuiteName"
            ]},
            {
               "index": "3",
               "suitename": "SuiteName"
            ]}
        ]
    }
```

List One Specific Suite

```
curl -X POST
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/saved_suites -d 1
    { "saved_suites": [
            {
               "index": "1",
               "suitename": "SuiteName",
               "vendor": "FastNVMeCo",
               "company": "FastNVMECo Inc.",
               "engineer": "Vincent Asbridge",
               "address": "1 Monarch Dr Littleton MA",
               "phone": "978-897-1888Littleton MA",
            "tests": [
               {
                   "section": "2",
                   "level": "1",
                   "sequence": "0",
                   "passes": "1",
                   "passtime": "60",
                   "name":
"/virtualun/rest/scripts/IO_Tests/Read_Seq_64thr_1blk.sh"               }
            ]}
        ]
```

```
    }
```

New REST Commands for Running Reports and Reading Results

Get List of Saved Results

```
curl -X POST
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/saved_results -d 1
[
    { "saved_results": [
            {
               "report": "20-10-
05_16_53_54.N1.T101.L1.R24.P23.F0.W1.S0.S550NE0M800315.html"
            }
        ]
    },
    {
        "status": {
            "code": 200,
            "reason": "Success"
        }
    }
]
```

Generate a New Report

```
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/report -d "-s 1 -p 0 -t
103 -l 1 -T all -v SANBlaze"
```

Set StopOnFirstError

New optional parameter introduced in V8.2 Beta3. StopOnFirstError will force tests to terminate after the first error is detected rather than waiting for the script to exit. Write a 1 to enable StopOnFirstError, 0 to disable.

```
curl -X GET
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/100/S
topOnFirstError
```

```
curl -X POST
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/100/S
topOnFirstError -d 1
```

Script Loops

Suites of scripts can now be run in a loop. You can define a number of scripts and set the loops parameter to set the number of times a group of scripts will run in a loop.

```
curl -X GET
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/100/luns/loops
```

```
curl -X POST
http://192.168.1.101/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/100/luns/loops -d 2
```

Appendix A: Programming Notes

The notes below are for SANBlaze internal use.

The method of operation for the REST interface is as follows:

A read or write is sent via HTTP (curl -X GET or -X POST for example).

vlun_web20.c receives the GET or POST and reads or writes to a file in the REST tree, for example:

```
curl -X POST
http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400
/set_load12V -d 12000
[
    {
        "set_load12V": "12000"
    },
    {
        "status": {
            "code": 200,
            "reason": "Success"
        }
    }
]
```

Writes the file /rest/sanblazes/1/ports/0/targets/400/set_load12V

```
cat /rest/sanblazes/1/ports/0/targets/400/set_load12V
12000
```

sb_testmgr watches files in /rest/sanblazes/N and /rest/sanblazes/N/ports/P/targets/t for writes

If a write is issued to a watched file at the CLI or via the REST API, sb_testmgr will get an interrupt and read the command from the file.

If the command is recognized, sb_testmgr will write to the input queue for sb_i2c.c and write to the command status file status_set_load12V for example:

```
cat /rest/sanblazes/1/ports/0/targets/400/status_set_load12V
Pending: echo "sb_i2c -n 1 -d 0 -v 12000" >> /tmp/NVMe/i2c_in
```

Writing to /tmp/NVMe/i2c_in will trigger sb_i2c to act on the command

When sb_i2c begins working on the command, the status is changed to Running: in the status file

When sb_i2c finishes, the status file will contain either Pass: <command> or Fail: <command>

If a command fails, in addition to having a Fail: status in the status_ file, sb_i2c also writes the Fail: status to /rest/sanblazes/N/weberror

When the web requests the sbexpress data from vlun_web20.c, if the weberror file exists, the error is sent to the web, and the file is deleted so that the user only sees the message once. The error is displayed in a pop up window so the user knows the command failed.

Example failure case:

curl -X POST http://192.168.100.103/goform/JsonApi?op=rest/sanblazes/1/ports/0/targets/400/set_load12V -d 16000 (too high!)

cat /rest/sanblazes/1/ports/0/targets/400/status_set_load12V

Fail: sb_i2c -n 1 -d -3 -v 16000

Web will see: