SANBlaZe

White Paper

Python Support for Verifying NVMe SSDs

Controlling the SBExpress Interface

Using Python for NVMe SSD Testing

Author:

Haiyan Lin, Sr. Software Engineer, SANBlaze

Table of Contents

Introduction

Many customers today are using the Python scripting language to achieve automation of their NVMe test systems. SANBlaze now offers one Python package (sanblaze_apis) that does not require any modifications to your local python configuration for integration or operation. The Python package provides a group of classes to control all functions of the SBExpress system including functions such as Power Control, Power Measurement, Hot Plug, Surprise Removal, Graceful Removal, PCIe Speed Training, Slot Information, NVMe command pass-through, namespace management, MI management, and more.

The SANBlaze Python package "sanblaze_apis" is compatible with Python2 and Python3 and is available for all SANBlaze SBExpress NVMe SSD test systems. Use of the SANBlaze "sanblaze_apis" Python package is documented in full in a user guide available from SANBlaze entitled *Controlling SANBlaze SBExpress NVMe Tester with Python*.

This white paper provides an overview of setup and usage of the Python package available for use with the SANBlaze SBExpress NVMe SSD test system.

Running the SANBlaze APIs from the SANBlaze system

By default, the "sanblaze-apis" are included starting with the V8.1 release or later. To access the "sanblaze_apis" you just need to log into the system via an SSH session and switch your working directory to "`/virtualun/apis/`" and import the class and instantiate objects with an optional slot number or port/target/LUN number.

You can instantiate as many objects as required for the specific slot or target that you need to communicate.

For example, following is the import and instantiate process in Python; it instantiates one object "s" with slot 4 and default system number 1.

```
[root@vlun-x10-100-111-IPMI-127 apis]# python
Python 2.7.15 (default, Oct  1 2019, 10:50:31)
[GCC 4.6.3 20120306 (Red Hat 4.6.3-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from sanblaze_apis import SB_i2c
>>> s = SB_i2c(4)
>>> dir(s)
['__class__', '__delattr__', '__dict__', '__doc__', '__format__',
'__getattribute__', '__hash__', '__init__', '__module__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', 'bump_insertion_count',
'button_press', 'clear_present', 'clock_glitch', 'clock_off', 'clock_on',
'clock_status', 'get_back_panel_switch_configurations',
'get_current_link_rate', 'get_deviceID', 'get_eeprom', 'get_info',
'get_max_link_rate', 'get_present', 'get_target_link_rate', 'get_vendorID',
'ident_off', 'ident_on', 'password', 'perst_glitch', 'perst_off', 'perst_on',
'perst_status', 'power_calibrate', 'power_glitch', 'power_margin',
'power_measure', 'power_measure_whole_tester', 'power_off', 'power_on',
'power_status', 'sb_sdb', 'set_insertion_count', 'set_present', 'slot',
'system', 'tester_IP', 'train_link_speed', 'username']
```

Running the SANBlaze APIs from your PC

You also have the option to run the sanblaze_apis directly from your local laptop/PC as well if that matches your python test infrastructure.

1) Copy the directory "`/virtualun/apis/`" (scp/ftp) from the SANBlaze system to your laptop. For example: C:/SANBlaze/Api
2) Install the Python "paramiko" package on your laptop with instructions as follows.

For Python 2.7.x:

1) Install Python 2.7.x - Select the correct MSI for your architecture (32-bit or 64-bit).
2) Download **get-pip.py** (use Firefox/Chrome) or use the following command "curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py." Refer to information in the webpage https://pip.pypa.io/en/stable/installing/ for details.) Put the installed package in directory "c:\Python27\".
3) Open an Administrator command prompt and run **c:\Python27\python.exe get-pip.py**.
4) From the same admin prompt, run **c:\Python27\Scripts\pip install paramiko**

For Python 3.8.x:

1) Install Python 3.8.x - Select the correct MSI for your architecture (32-bit or 64-bit).
2) Download **get-pip.py** (use Firefox/Chrome) or use the following command "curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py." Refer to information in webpage https://pip.pypa.io/en/stable/installing/ for details) and put it in directory "c:\Users\YourName\AppData\Local\Programs\Python\ Python38\".
3) Open an Administrator command prompt and run " c:\Users\hlin\AppData\Local\Programs \Python\Python38\python.exe get-pip.py".
4) From the same admin prompt, run **c:\Users\hlin\AppData\Local\Programs\Python\Python38\ Scripts\pip install paramiko**.

After the Python "paramiko" package installation is complete, you can import this class and instantiate objects with an optional slot number, optional system number, remote tester IP, access username and password.

For example, following is the import and instantiate process from a laptop to the SANBlaze system remotely controlled in the lab. It instantiates one object "s" with slot 4, default system number 1, tester IP = "192.168.100.111" with user name = "root" and password = "SANBlaze".

```
>>> from sanblaze_apis import SB_i2c
>>> s = SB_i2c(4, 1, "192.168.100.111", "root", "SANBlaze")
>>> dir(s)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', 'bump_insertion_count',
'button_press', 'clear_present', 'clock_glitch', 'clock_off', 'clock_on',
'clock_status', 'get_back_panel_switch_configurations',
'get_current_link_rate', 'get_deviceID', 'get_eeprom', 'get_info',
'get_max_link_rate', 'get_present', 'get_target_link_rate', 'get_vendorID',
'password', 'perst_glitch', 'perst_off', 'perst_on', 'perst_status',
```

```
'power_calibrate', 'power_glitch', 'power_margin', 'power_measure',
'power_measure_whole_tester', 'power_off', 'power_on', 'power_status',
'sb_sdb', 'set_insertion_count', 'set_present', 'slot', 'ssh', 'system',
'tester_IP', 'train_link_speed', 'username']
```

With this object "s" above then you have full control of slot 4. You can find help information for each API as follows:

```
>>> help(s.get_info)
Help on method get_info in module sanblaze_apis:
get_info(self) method of sanblaze_apis.SB_i2c instance
    Get slot information such as drive bay, tester SN/revision, i2c
addresses, link width/speed, power/LEDs status, PEX port/bus number,MRL
(Mechanical Retention Latch) sensor state, DUT on the bus or not and so on.
>>> help(s.power_glitch)
Help on method power_glitch in module sanblaze_apis:
power_glitch(self, glitch_hold_us=100000) method of sanblaze_apis.SB_i2c
instance
    Glitch the power for specific time on the slot
```

SANBlaze sb_i2c Python Module

The SANBlaze sb_i2c Python module defines one SB_i2c class which has the following 34 APIs so far for power control/measurement/glitch, hot plug, PCIe speed training, clock/PERST glitch and so on:

- `button_press(self)`
- `power_measure(self)`
- `get_info(self)`
- `get_eeprom(self)`
- `power_status(self)`
- `power_on(self)`
- `power_off(self)`
- `power_glitch(self, glitch_hold_us=100000)`
- `power_margin(self, voltage_margin_mv=11000)`
- `clock_status(self)`
- `clock_on(self)`
- `clock_off(self)`
- `clock_glitch(self, glitch_hold_us=100000)`
- `perst_status(self)`
- `perst_on(self)`
- `perst_off(self)`
- `perst_glitch(self, glitch_hold_us=100000)`
- `train_link_speed(self, link_speed)`
- `get_max_link_rate(self)`
- `get_target_link_rate(self)`
- `get_current_link_rate(self)`
- `power_calibrate(self)`
- `power_measure_whole_tester(self)`
- `get_deviceID(self)`
- `get_vendorID(self)`
- `get_present(self)`

- clear_present(self)
- set_present(self)
- bump_insertion_count(self)
- set_insertion_count(self, init_count)
- get_back_panel_switch_configurations(self)
- sb_sdb(self, register_address, writing_data=None, single_byte=False, binary_display=False, show_Atlas_temperature=False, show_link_errors=False, reset_link_errors=False, verbose=False, apply_all_slots=False)
- ident_on(self)
- ident_off(self)

The constructor of this SB_i2c class is "__init__(self, slot=0, system=1, tester_IP=None, username='root', password='SANBLaze')".

For example, you can get slot information such as drive bay, appliance SN and revision, i2c addresses, link width and speed, power and LEDs status, PEX port and bus number, MRL (Mechanical Retention Latch) sensor state, DUT on the bus or not, and so on with the API "get_info". This API call will retrieve slot 4 information from the SBExpress system 1 (default is 1 if not specified when you instantiate the object) as follows with the object "s" instantiated above.

```
>>> s.get_info()
INFO: System 1 SBExpress-RM  SN=920A8110006 Rev=R03 i2c=/dev/i2c-4
MI_i2c=/dev/i2c-5 VLUN_Port=0
  INFO: driveslot=04
  INFO: DriveBay=04
  INFO: PEXi2cAddr=0x5d
  INFO: PEXStation=0x02
  INFO: PEXPort=0x09
  INFO: PRIMARY_BUS=0x05
  INFO: SECONDARY_BUS=0x0f
  INFO: LNK_CUR_SPEED=0x03
  INFO: LNK_CUR_WIDTH=0x04
  INFO: PRESENCE=0x01
  INFO: MRL_SENSOR_STATE=0x00
  INFO: ATTN_LED=0x01
  INFO: POWER_LED=0x00
  INFO: POWER=0x01
  INFO: In waitForStatus Submitting -n 1 -d 4 -i -G 125823 to queue at
/tmp/NVMe/i2c_in
```

SANBlaze XML_API Python Module

The SANBlaze XML_API Python class defined 158 APIs so far for all NVMe commands for pass- through, decoding, and for use with the VLUN system management. The XML_API APIs run in the exact same manner either locally on the test system or remotely from a PC/laptop. Here is an example of how you can instantiate one object, find help for each API, and run the API:

```
>>> t=XML_API(tester_IP="192.168.100.111", port=0, target=107, lun=1, raw_return=2)
>>> help(t.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 (we recommend using a hex number like 0x104 to reset all zones.) Zone send actions are as follows:

```
    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
```

```
>>> temp = t1.zone_management_receive_for_zone(10)
The start LBA of zone 10 is: 0x3c000
Command ZoneManagementReceive passed on port 0 target 107 in tester
192.168.100.111
Command output is decoded as follows:
                                    Num_Zones = 0x000000000000A276
                  Zone_Descriptor:
                                    Zone_Type = 0x02
                                   Zone_State = 0xE0
                  Zone_Attributes:
                   Zone_Finished_by_Controller = 0x00
                       Zone_Finish_Recommended = 0x00
                        Reset_Zone_Recommended = 0x00
                                    ZRWA_Valid = 0x00
               Zone_Descriptor_Extension_Valid = 0x00

                                 Zone_Capacity = 0x0000000000006000
                                Zone_Start_LBA = 0x000000000003C000
                                 Write_Pointer = 0x000000000003C000

Command Completion Queue Status is decoded as follows:
                               CommandSpecific = 0x00000000
                                     Reserved0 = 0x00000000
                               SQ_Head_Pointer = 0x0363
                                 SQ_Identifier = 0x0001
                            Command_Identifier = 0x04B2
                     Status_Field:
                                      PhaseBit = 0x00
                                    StatusCode = 0x0000
                                 StatusCodeType = 0x00
                                       Reserved = 0x00
                               MoreInformation = 0x00
```

```
                                        DoNotRetry = 0x00
>>> temp
(0, {'Num_Zones': 41590, 'Zone_Descriptor': {'Zone_Type': 2, 'Zone_State':
224, 'Zone_Attributes': {'Zone_Finished_by_Controller': 0,
'Zone_Finish_Recommended': 0, 'Reset_Zone_Recommended': 0, 'ZRWA_Valid': 0,
'Zone_Descriptor_Extension_Valid': 0}, 'Zone_Capacity': 24576,
'Zone_Start_LBA': 245760, 'Write_Pointer': 245760}})
```

SANBlaze MI Python Module

The SANBlaze MI Python class defined 50 APIs so far for all current MI commands passing through/ decoding with SMBus/I2C and VDM/PCIe (if supported). The MI APIs run in the exact same manner either locally on the test system or remotely from a PC/laptop. Here is an example of how you can instantiate one object, find help information, and run the API:

```
>>> from sanblaze_apis import MI
>>> m1 = MI(1, 1, mi_transport="smbus", tester_IP="192.168.100.111", raw_return=1, decoding=True)
>>> dir(m1)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'basic_identification',
'basic_status', 'basic_vendor_specific', 'configuration_get', 'configuration_set',
'control_primitive_abort', 'control_primitive_get_state', 'control_primitive_pause',
'control_primitive_replay', 'control_primitive_resume', 'controller_health', 'controller_info',
'controller_list', 'curr_file_path', 'decoding', 'device_selftest', 'firmware_commit',
'firmware_download', 'format', 'get_feature', 'get_logpage', 'hPrint', 'identify',
'identify_IO_command_set', 'identify_NS_attached_controller_list',
'identify_ZNS_allocated_namespace', 'identify_ZNS_allocated_namespace_IDs',
'identify_ZNS_controller', 'identify_ZNS_namespace', 'identify_ZNS_namespace_IDs',
'identify_active_namespace_IDs', 'identify_allocated_namespace',
'identify_allocated_namespace_IDs', 'identify_controller', 'identify_controller_list',
'identify_namespace', 'identify_namespace_ID_descripts', 'log_file',
'me_buffer_commands_supported', 'mi_transport', 'nvm_subsystem_health', 'nvm_subsystem_reset',
'nvme_xml', 'optional_commands_supported', 'pack_to_binary_string', 'parse_stdout_output',
'password', 'pcie_configuration_read', 'pcie_configuration_write', 'pcie_io_read',
'pcie_io_write', 'pcie_memory_read', 'pcie_memory_write', 'port_info', 'ports_version',
'raw_return', 'sanitize', 'set_feature', 'show_field_values', 'slot', 'ssh', 'system',
'tester_IP', 'timeout', 'username', 'verbose', 'vpd_read', 'vpd_write', 'xmlstruct',
'xmlstruct_path', 'xmltable', 'xmltable_path']
```

With this object “m1” above then you have the full MI control of slot/drive 1. User can find help information for each API as follows:

```
>>> help(m1.basic_identification)
Help on method basic_identification in module sanblaze_apis:
basic_identification() method of sanblaze_apis.MI instance
    Issue MI type "NVMe-Basic" command with opcode "Basic Identification" to read back vendor ID,
drive SN and PEC (Packet Error Code).

>>> temp = m1.controller_info(controller_id=0)
Read NVMe-MI Data Structure
command is 'sb_i2c -n 1 -d 1 -z -k 0 -w 84 08 00 00 00 00 00 00 00 00 00 03 00 00 00 00 16 b0 35
4f'

Response Data Length: 32
Controller Information Data Structure:
  Port Identifier: 0
  PCIe Routing ID Valid: 0
  PCIe Routing ID:
    PCI Bus Number: 0Dh
    PCI Device Number: 00h
    PCI Function Number: 0h
  PCI Vendor ID: 1B96h
```

```
  PCI Device ID: 2600h
  PCI Subsystem Vendor ID: 1B96h
  PCI Subsystem Device ID: 2600h
>>> temp
{'Response Data Length': '32', 'Controller Information Data Structure': {'Port Identifier': '0',
'PCIe Routing ID Valid': '0', 'PCIe Routing ID': {'PCI Bus Number': '0Dh', 'PCI Device Number':
'00h', 'PCI Function Number': '0h'}, 'PCI Vendor ID': '1B96h', 'PCI Device ID': '2600h', 'PCI
Subsystem Vendor ID': '1B96h', 'PCI Subsystem Device ID': '2600h'}}
```

For a comprehensive list of API commands, please contact SANBlaze.

Summary

SANBlaze now provides – in addition to an easy-to-use GUI interface and its extensive SBExpress automated test scripts – a way to programmatically control these interfaces and low-level commands through a common Python API interface for ease of integration into your existing Python test infrastructure. With the ability to run alongside your current test infrastructure, this will allow easy migration to the SANBlaze test platform and allow you to greatly enhance your test coverage and results.