InfiniFlash™ System

November 30, 2016

IF150

RESTful API Guide

SanDisk
a Western Digital brand

Western Digital Technologies, Inc.
951 SanDisk Drive, Milpitas, CA 95035
Western Digital Technologies, Inc. is the seller of record and licensee
in the Americas of SanDisk® products

www.SanDisk.com

PN: 80-11-01943

2016 - 11 Rev. B1

SanDisk
a Western Digital brand

Contents

List of figures

About this guide

This guide describes the RESTful software interface for the InfiniFlash™ IF150. The information in this guide is intended for administrators responsible for servers and storage systems.

It is presumed they are familiar with basic Linux system administration. Later sections provide details about CLI functions, parameters, and return codes.

Typographical conventions

Table 1-1. Typographical conventions

Convention	Usage	Examples
 **NOTE:**	Important additional information or further explanation of a topic.	 **NOTE:** A weekly backup is recommended.
 **WARNING!**	The task or operation might have serious consequences if conducted incorrectly or without appropriate safeguards. If you are not a product expert, consult SanDisk for assistance.	 **WARNING!** **Do not change configuration parameters.**
Bold	A command or system input that you type, or text or a button displayed on a screen.	Click **HELP** for details on disaster recovery.
Italic	Italic font indicates any of the following: *A term with a specific meaning* in the context of this document. *Emphasis* on specific information. *Reference* to another document.	Detailed information on *disaster recovery* methods is available in the *Administrator Guide*.
Square [ ] Brackets	Syntax elements within square brackets are optional. Vertical bars separate choices from which none or only one can be selected.	`ifcli [help]`
Angle < > Brackets	Syntax elements within angle brackets are required and must be replaced with user-specified inputs. Vertical bars separate choices from which at least one must be selected.	`ifcli <cmd>`

Abbreviations

Table 1-2. Abbreviations

Abbreviation	Description
SE	SAS Expander
SAS	Serial Attached SCSI
HSE	Host SAS Expander
DSE	Drive SAS Expander
FIB	Fan Interface Board
SSD	Solid State Drive
SES	SCSI Enclosure Services
REST	Representational State Transfer
CLI	Command Line Interface
WWN	World Wide Name

1

InfiniFlash IF150 overview

The InfiniFlash™ IF150 is a 3U storage enclosure consisting of the following components:

- Drive plane board (DPB) that supports up to 64 board SSD modules (BSSDs)
- Four drive SAS expanders (DSEs)
- Two host SAS expanders (HSEs)
- Four fan interface boards (FIBs) each having two fans
- Two power supply units (PSUs)

The system enclosure is a storage enclosure mounted into a rack using full extension rack slides. The DSEs and the BSSD modules are serviced from the top after removing the cover.

The InfiniFlash IF150 enclosure has two HSEs, each with four mini-SAS connections for interfacing external servers through 12 Gb/s SAS links using cables up to three meters long.

The BSSD Module is an InfiniFlash drive card with a capacity of 4 or 8 TB.

Manageable elements

The following are the various types of elements inside the InfiniFlash IF150 that can be managed and monitored:

Enclosure

The enclosure provides the power, cooling, mechanical protection, and external electronic interfaces for attached InfiniFlash drive cards. It provides services that establish the mechanical environment, electrical environment, and external indicators and controls for the proper operation and maintenance of devices within the enclosure.

Device slot

A device slot holds an InfiniFlash BSSD. The slot provides appropriate power, signal, and control connections to the InfiniFlash drive card device. The position also provides mechanical protection, locking capability, automatic insertion, visual device status indicators, and other features to manage the InfiniFlash drive card in the enclosure.

Fan

An InfiniFlash IF150 can contain up to four fan modules to keep the enclosure and its elements at the desired temperature. Each fan module contains two fans.

Temperature sensors

The temperature sensors provide current temperature measurements at various installation points inside the enclosure.

Voltage sensors

The voltage sensors provide current voltage measurements at various installation points within the enclosure.

HSE

Each HSE provides an external interface to connect to one or more servers through four 12 Gb/s SAS links using SAS connectors. It also manages enclosure elements such as fans, drive slots, and the power supply unit. The InfiniFlash IF150 supports up to two HSEs for high availability.

DSE

DSEs manage the internal interfaces that connect to up to 32 InfiniFlash BSSDs. The InfiniFlash IF150 supports up to 4 DSEs. It uses two DSEs to connect to port A of all the InfiniFlash BSSDs (up to 64); the remaining two DSEs are used to connect to port B of the same set of InfiniFlash BSSD cards.

BSSD

Each BSSD is an InfiniFlash drive card. Up to 64 BSSDs can be installed.

Related documents

- *InfiniFlash IF150 Site Preparation Guide*
- *InfiniFlash IF150 Hardware Installation Guide*
- *InfiniFlash IF150 Hardware User Guide*
- *InfiniFlash IF150 Software Installation Guide*
- *InfiniFlash IF150 CLI User Guide*
- *InfiniFlash IF150 Troubleshooting Guide*
- *InfiniFlash IF150 Regulatory Information Flyer*

2

InfiniFlash tools overview

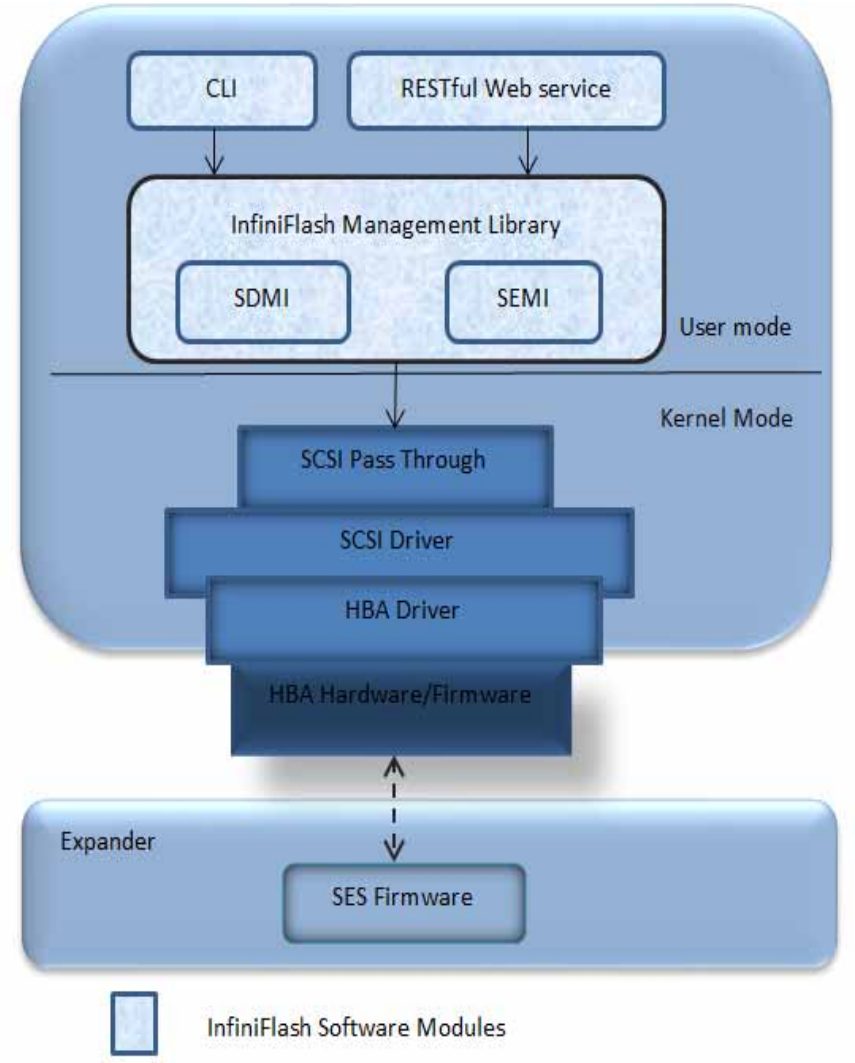
The InfiniFlash tools enable management and monitoring of the InfiniFlash IF150 and its elements, including the InfiniFlash drive cards. The tools allow you to manage all directly attached InfiniFlash devices on a host system, either through a command line terminal or REST interfaces across the web.

InfiniFlash IF150 firmware provides SES-compliant SCSI targets to the attached host through the connected HBA so that any OS process can configure its capabilities. The InfiniFlash tools interact with system firmware through SPC and SES commands and provide simple interfaces to manage the system elements. They also leverage proprietary features provided by the firmware to achieve some of the critical functions not available to an SES-standard-based third party utility.

Software tools architecture

A high level view of the software tools architecture is shown in Figure 2-1 on page 14.

Figure 2-1. InfiniFlash IF150 software modules



CLI

CLI is the presentation module that takes command line input, translates it to InfiniFlash management library interfaces, performs the requested operations, and displays results to the console.

RESTful web service

The representational state transfer (REST) web service runs as a daemon in the host OS and provides standardized web services to remote clients interacting with the InfiniFlash management library to manage the system and its InfiniFlash drive cards.

InfiniFlash management library

This C/C++ library module, used by the CLI and the RESTful web service, provides an API to manage the system and its elements.

SanDisk drive management interface

SanDisk drive management interface (SDMI) is a sub-module for drive management within the InfiniFlash Management Library. It provides a common interface for managing any SanDisk SSD, and provides an interface for getting various drive properties such as serial number and firmware version, and drive operations such as format, self-test, and firmware update.

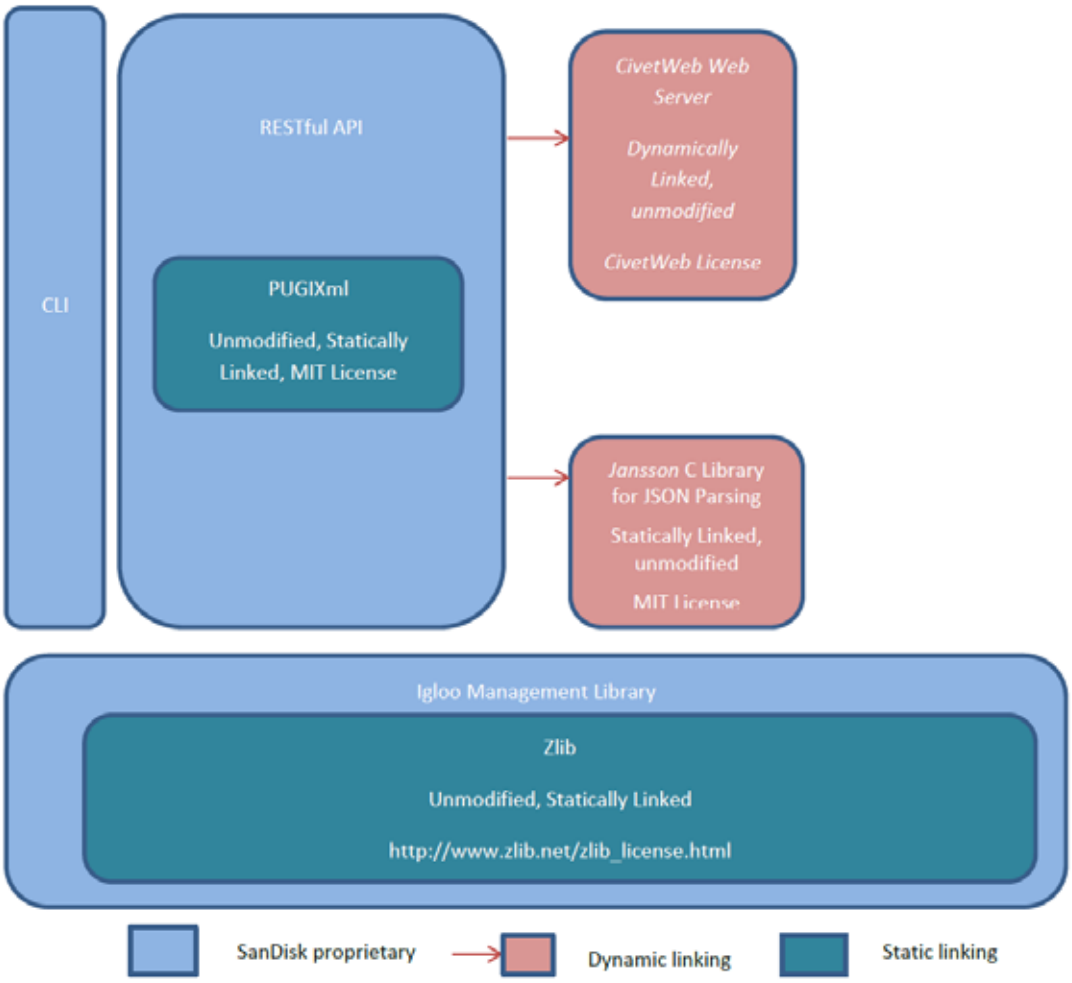
SanDisk enclosure management interface

The SanDisk enclosure management interface (SEMI) is a sub-module for enclosure management within the InfiniFlash Management Library. It lists the system devices connected to the host and provides a standard interface for managing them. It also provides interfaces for performing management operations on the enclosure and its elements, such as getting product numbers, serial numbers, and configuration pages. It also provides an interface for fetching log pages, firmware updates, self-tests, and more.

Open source architecture

InfiniFlash tools use open source components for its common infrastructure. The diagram in Figure 2-2 on page 16 represents the open source architecture.

Figure 2-2. Open source architecture

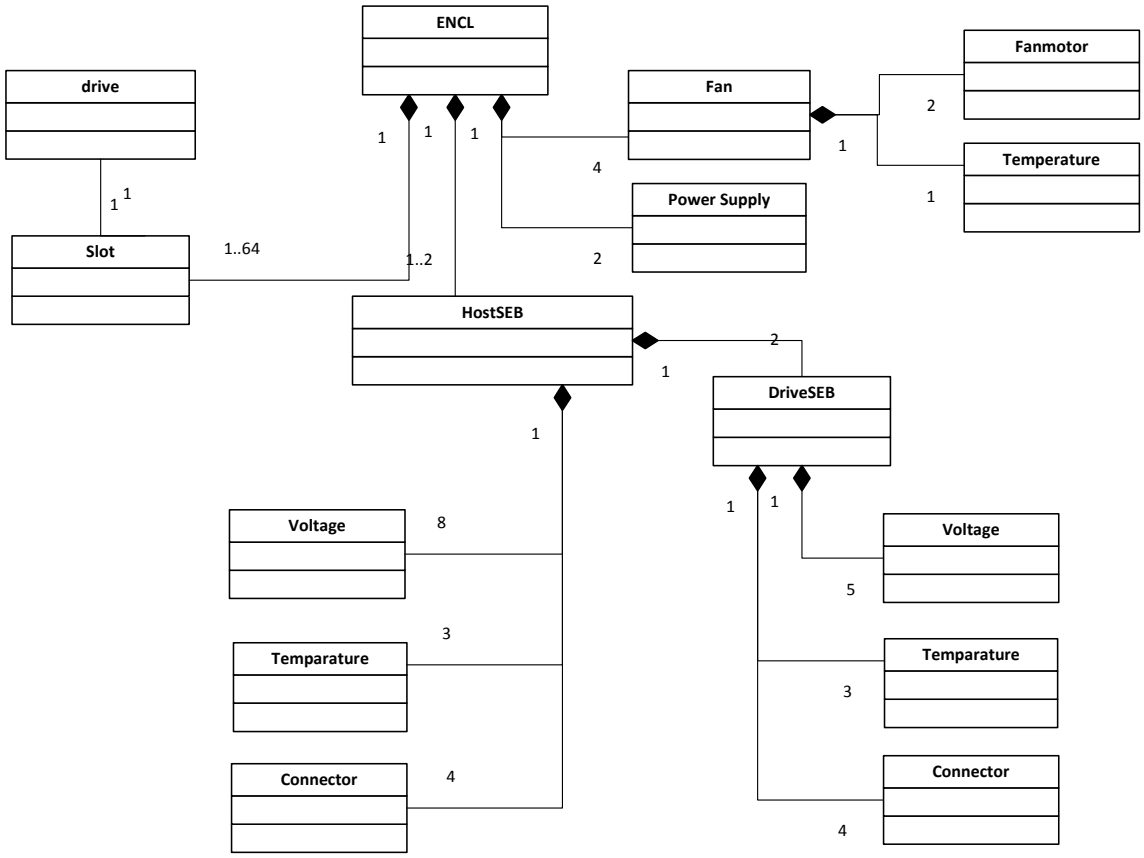


3

InfiniFlash management object model

The following diagram represents the relationship between the InfiniFlash IF150 and its elements. The tools provide mechanisms to manage the system and its elements using these relationships.

Figure 3-1. InfiniFlash IF150 object model



Object naming convention

The InfiniFlash tools provide support for managing the InfiniFlash IF150 and its elements. An object type is assigned for each of the elements. The system tools use the object type to form the object names for the elements. The list of elements and their object type is given in Table 3-1 on page 18.

Table 3-1. Object naming conventions

Object Type	Description
encl or wwn	InfiniFlash Enclosure
hse	Host SAS Expander
d	Drive SAS Expander
ts	Temperature Sensors
vs	Voltage Sensors
fan	Fan Module
fm	Fan Motor
slot	Slots
drive	Drive in a Slot
psu	Power Supply Unit
cn	SAS connector

InfiniFlash object name

Each of the manageable elements supported by the system tools is assigned an object name. The assigned object name is used to perform management operations on the manageable elements. The object names are formed based on the hierarchical InfiniFlash management object model on page 17 and the object identifiers assigned by the InfiniFlash tools.

The system tools assign an object identifier for each of the manageable elements. The object identifiers are zero-based. For example, the drive slots get object identifiers from 0 to 63, and they are identified as slot0, slot1 and so on. Objects of the same type at different levels in the hierarchy get the same object identifiers. For example, the first voltage sensor at HSE and DSE gets the object identifier of 0 (vs0).

In addition to the object identifier, an object path is also required to uniquely address an object. The object path is formed based on the relationship between the elements defined in InfiniFlash management object model on page 17.

The object names to address various enclosure elements are listed in Table 3-2 on page 19.

Table 3-2. Enclosure object names

Object Path	Description
encl<id> or wwn<logical-id>	Addresses a specific enclosure
encl<id>/fan<id> > or wwn<logical-id>/fan<id>	Addresses a specific fan module available in a specified enclosure
encl<id>/fan<id>/fm<id> or wwn<logical-id>/fan<id>/fm<id>	Addresses a specific fan motor available in a specified fan module
encl<id>/hse<id> > or wwn<logical-id>/hse<id>	Addresses a specific HSE in a specified enclosure
encl<id>/hse<id>/d<id> or wwn<logical-id>/hse<id>/d<id>	Addresses a specific DSE in a specified HSE
encl<id>/hse<id>/ts<id> or wwn<logical-id>/hse<id>/ts<id>	Addresses a specific temperature sensor available in a specified HSE
encl<id>/hse<id>/d<id>/ts<id> or wwn<logical-id>/hse<id>/d<id>/ts<id>	Addresses a specific temperature sensor available in a specified DSE
encl<id>/hse<id>/vs<id> or wwn<logical-id> hse<id>/vs<id>	Addresses a specific voltage sensor available in a specified HSE
encl<id>/hse<id>/d<id>/vs<id> or wwn<logical-id>/hse<id>/d<id>/vs<id>	Addresses a specific voltage sensor available in a specified DSE
encl<id>/slot<id> or wwn<logical-id>/slot<id>	Addresses a specific slot available in a specified enclosure
encl<id>/slot<id>/drive or wwn<logical-id>/slot<id>/drive	Addresses a specific InfiniFlash drive card available in a specified slot
encl<id>/hse<id>/cn<id> or wwn<logical-id>/hse<id>/cn<id>	Addresses a specific SAS connector available in the specified HSE

In addition to addressing the InfiniFlash drive cards through the object names shown above, InfiniFlash drive cards can also be managed through the device names assigned by the operating system through the conventions listed in Table 3-3 on page 19.

Table 3-3. Device paths

Object Path	Description
encl<id>/slot<id>/drive or wwn<logical-id>/slot<id>/drive	Enclosure path
dev/sdX	Operating system drive name (Linux), similar OS drive names can be derived for other supported OSs.

The InfiniFlash tools address all the elements of the same type at the same hierarchical level by special object paths, as listed in Table 3-2 on page 19.

4

InfiniFlash RESTful web service

The RESTful web service provides an interface that manages components of the InfiniFlash enclosure. It is designed with the following REST principles:

1. Resources (elements and jobs) are identified by URLs
2. Data is sent/received as JSON/XML over HTTP
3. Manipulations of resources are done by HTTP methods (GET, POST)

The interface is stateless and hypertext driven.

NOTE: You may get an empty response when a REST request is made during the uninstallation of the InfiniFlash tools.

API request/response format

InfiniFlash RESTful API management follows a well-defined URL convention to refer to various enclosure elements and the jobs running in the system. All APIs comply with the following URL syntax:

```
<http|https>://<hostname>:[portnumber]/<webroot>/<resourcename>
[?queryparameter]
```

Protocol support

The web service supports access to RESTful API through both HTTP and HTTPS. The port number the web service uses to listen for client requests can be configured during the installation of the web service.

Webroot

All the RESTful APIs are implemented under the webroot `sndk/infiniflash`.

Resource name

Identifies the name of the resource against which the operation is invoked. The resource name could either include InfiniFlash object names or refer to job objects.

Query parameter

The query parameter can be used to access a specific attribute or a method of an object.

Table 4-1. HTTP header

Name	Description
Content-Type	Where the API requires an HTTP body, the content should be provided and the content type should also be stated in the header. Below are the content types supported in a request: application/xml, application/json, multipart/form-data and application/octet-stream.
Accept	In the HTTP request, a client can state the type of data accepted as part of response. Supported return types are application/xml and application/json. By default, XML content type is returned in the response.

Table 4-2. HTTP request

Request Method	Description	Query Parameter	Request Body
GET	Mapped to getting properties of objects.	An optional attribute passed as a part of the URL. Example: asset, state.	N/A
POST	Mapped to enclosure management operation	An optional action/attribute passed as a part of the URL. Example: self-test, update.	Optional and part of http body in the form of XML or JSON. Default would be XML.

HTTP response

With a successful GET/POST operation, the HTTP response body contains the requested information in either JSON or XML format.

With a request failure, an additional status code and error description string may be returned as part of the response body.

Table 4-3. HTTP status code

Status	Description
200 – Success	Returned when the requested operation is completed successfully.
201 – Created	Returned whenever a new object is created for this operation.
400 – Bad request	Indicates that the requested operation is either invalid or the parameters passed in the request body are invalid. The response body contains an additional status code and error description string.

Table 4-3. HTTP status code (continued)

Status	Description
500 – Operation failure (Internal server error)	Returned if the requested operation fails in the web service. The response body contains a status code and error description string.

Error responses

When there is an error status code returned, the body contains an additional status code and error description string. The following sample HTTP body data shows the structure of error response body:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
    <ErrorCode>-9</ErrorCode>
    <ErrorDescription> Self-test operation failed </ErrorDescription>
</error>
```

The following table lists the error response elements.

Table 4-4. Error response elements

Name	Description
Error	Container for all error elements. Type: Container Ancestor: None
ErrorCode	The error code is an integer that identifies an error message. Type: Integer Ancestor: Error
ErrorDescription	Message contains the details description of error code. Type: String Ancestor: Error

Error codes

Requests return an error code when there is a logical or conditional failure in the RESTful web server software or as a result of a failure in any SCSI operation on the devices. In both cases, REST API reports an error code and a description of the error to aid in the debugging of the failure.

If any SCSI operation fails, that SCSI device returns an error in the form of a KCQ as indicated by the SCSI specification. The RESTful web server reports that error by encapsulating KCQ values in the following way:

- `Errorcode = ((Sense Key << 16)|(ASC << 8)|(ASCQ)`

If an error code value is greater than 0, sense key, ASC, and ASCQ values can be extracted as above and can be used for further debugging.

The error codes (with value less than 0) and their descriptions are listed in the following table:

Table 4-5. Error codes

Error Code	Description
-1	Failure
-2	Insufficient memory
-3	Invalid parameter, or failed to parse
-4	Uninitialized
-5	Unable to open file
-6	Unable to create lock
-7	Unable to acquire lock
-8	Admin privilege required
-9	Invalid command
-10	Unable to open device
-11	SCSI error
-12	Device type not supported
-13	Already initialized
-14	Invalid device
-15	Device not ready
-16	Not allowed in boot device
-17	Partial data returned
-18	Seek error
-19	Read failure
-20	No devices found
-21	Operation not allowed
-22	Firmware update process failed
-23	Firmware image not compatible
-24	Unable to open firmware image file
-25	Unable to perform because device has valid partitions
-26	Unable to read the firmware image file
-27	Firmware image file size is zero
-28	No parameters in drive statistics page
-29	Unable to write in the output file
-30	Operation failed on one or more requested devices
-31	Invalid device entry in cache

Table 4-5. Error codes (continued)

Error Code	Description
-32	Unable to create threads
-33	Invalid firmware image
-34	Unable to write all the data
-35	Inquiry failed for PAGE3 due to data unavailability
-36	Directory already exists
-37	Directory path not found
-38	Directory creation failed
-39	Unable to delete the folder
-40	Unable to open the file
-41	Unable to open the zip file
-42	Invalid image file type
-43	Unable to lock the device
-44	Access denied
-45	Self-test aborted
-46	Self-test failed
-47	Self-test not executed
-48	Registration not found
-49	Registration already exists
-50	BMSinterval value is not in the supported range.
-51	statsloginterval value is not in supported range
-52	Firmware update not allowed without defer activate
-53	Operation is not supported
-54	Given protection type is not supported by device
-55	Protection type and block size are not compatible
-56	No device found
-57	Not an NVMe device
-58	Unable to get device port type
-59	No dumptrace exists in the device
-60	Unable to delete the dump trace
-61	Unable to get the LED blinking status
-62	Unable to get version information
-63	Unable to get the bus, device and function number of the device
-64	maxlba is out of range

Table 4-5. Error codes (continued)

Error Code	Description
-65	No supported device found
-66	Invalid input; command terminated
-67	Operation canceled; command terminated
-68	Device list is empty
-69	Syntax mismatch found in device list
-70	Unknown option passed
-71	Mandatory option is not provided
-72	Invalid value given for an option
-73	Value is not provided for an option
-74	Option token is missing
-75	Option given more than once
-76	Combination of options provided not allowed
-77	Invalid path
-78	Unknown argument provided
-79	Option is missing
-80	Option value is out of range
-81	Device has dumptrace; cannot update the firmware
-82	Unable to delete the dump trace but fetched successfully
-83	Driver error
-84	Driver is busy; retry after a minute
-85	Unable to delete file
-86	Invalid file
-87	Unable to unzip the input file
-88	Log file not found in input file
-89	Inquiry EVPD 3h file not found in input file
-90	Supported log page not found
-91	Unknown command provided
-92	Option not used in appropriate combination
-93	Specified block size is not supported
-94	Given mode-page field is invalid
-95	Given page control for mode sense is invalid
-96	Index exceeds number of entries
-97	Given modepage field value is invalid

Table 4-5. Error codes (continued)

Error Code	Description
-98	Value of a mode-page field is not changeable; mode-page field value cannot be changed
-99	Invalid value for RxTxAssertBits
-100	Invalid hex input given
-101	Argument is missing
-102	ATA error
-103	ATA aborted
-104	ATA CRC Error
-105	ATA Sense data available
-106	ATA alignment error
-107	ATA Device Fault
-108	ATA uncorrectable error
-109	ATA ID not found
-110	Unable to get boot status
-111	It is not a SATA device
-112	It is not a SanDisk device
-113	Online or immediate-activate is currently not supported
-114	Invalid device list
-115	48-bit commands not supported
-116	HPA feature set not supported
-117	Cannot set maximum LBA greater than current maximum LBA
-118	Invalid device list; maximum of 32 devices supported at a time
-119	Power management feature not supported
-120	Many devices present
-121	Activation mode not supported
-122	Operation failed for all the devices in the list
-123	Log address not supported
-124	Security not supported
-125	Security frozen
-126	Security locked
-128	Sanitize not supported by the device
-129	Operation is not supported by the device
-130	Failed to enable security

Table 4-5. Error codes (continued)

Error Code	Description
-131	Device still in security enabled state
-132	Invalid page list
-133	No log pages found
-134	AHCI controller not found in the system
-135	Device initiated power management not supported
-136	Advanced power management not supported
-137	Invalid value for advanced power management
-138	Write cache not supported
-139	Automatic partial to slumber transition is not supported
-140	DEVSLP not supported
-141	Trim bit not supported in DSM command
-142	Failed with error invalid function.
-143	Invalid input parameter
-144	Invalid input value
-145	Unrecoverable error occurred
-146	SCTP STAT signature missing
-147	SCTP error in status
-148	Invalid section found
-149	Duplicate DLE format
-150	Duplicate firmware
-151	Duplicate configuration
-152	Duplicate format
-153	Missing DLE format
-154	Missing firmware
-155	Missing configuration
-156	Duplicate SSA
-157	Duplicate APP
-158	Cannot switch device to ROM mode
-159	Firmware image not compatible with device
-160	Bot file provided is not supported
-161	Power cycle is required
-162	Invalid percentage
-163	Invalid time

Table 4-5. Error codes (continued)

Error Code	Description
-164	Invalid I/O
-165	Invalid QDepth
-166	Cannot switch from ROM mode, power cycle is required
-167	Device format failed
-168	Invalid FFU max transfer size obtained from device
-169	Unable to detect the device after ROM mode change
-170	Unable to create thread
-171	Unable to release lock
-172	Test failed
-173	Order of the provided option is wrong
-174	No LLS or counter logs found in the device
-175	All the devices are selected instead of the device name
-176	Device list is provided instead of the device name
-177	Job cannot be canceled by the user
-178	Job deleted
-179	Job already deleted
-180	Job in progress
-181	Job already open
-182	Failed to create job
-183	Job canceled
-184	Job wait timed out
-185	Quick format is not supported by the device
-186	Number of threads is greater than queue depth
-187	Invalid combination
-188	Critical failure to allocate resource
-189	ATA trusted command set not supported
-190	TCG protocol not supported
-191	TCG version 2 not supported
-192	TCG OPAL not activated
-193	Failed to open TCG session
-194	Unable to get SMART trip details
-195	No SMART trip
-196	Unsupported page

Table 4-5. Error codes (continued)

Error Code	Description
-197	Improper progress time interval
-198	TCG protocol reset failed
-199	Failed to open TCG session
-200	Failed to close TCG session
-201	Segment size greater than transfer limit
-202	Option not required in this case
-203	Device is in ROM mode; enter serial number and WWN
-204	Device is not in ROM mode; new WWN not supported
-205	Device is busy; try the operation after sometime
-206	Unable to decode page due to invalid data
-207	Device LED is already blinking
-208	Device LED is not blinking
-209	Dump trace exists in the device
-210	Self-test completed successfully
-211	Self-test aborted by the host
-212	Self-test aborted as unknown test error occurred
-213	Self-test completed – unknown element failed
-214	Self-test completed – electrical element failed
-215	Self-test completed – servo/seek element failed
-216	Self-test completed – read element failed
-217	Self-test completed – device suspected to have handling damage
-218	Operation in progress
-219	Invalid object name
-220	Syntax error
-221	Duplicate entry
-222	Entry not found
-223	EOF reached
-224	Device is in busy state
-225	Device is in reduced functionality mode
-226	Update not required
-227	Incompatible image
-228	Invalid protocol identifier
-229	No operation in progress

Table 4-5. Error codes (continued)

Error Code	Description
-230	Invalid SAS address
-231	Unreachable device
-232	Job message queue is empty
-233	Too many devices as input
-234	One or more drive update failed
-235	One or more SEB update failed
-236	FPGA update failed
-237	FRU inaccessible
-238	Operation initiated
-239	No operation in progress on any drive
-240	Not supported for OS names
-241	Incompatible package
-242	Boot code update failed for one or more SEBs
-243	ISTR update failed for one or more SEBs
-244	Different ISTR across SEBs, undefined default behavior for zoning
-245	Chassis detected with SEBs from different products
-246	Update with only \encl\ type allowed in busy state
-247	Maximum allowed regions exceeded
-248	Zero valid regions
-249	Option not supported
-250	Invalid drive list
-251	Dev link error
-252	Required image not found in FFU package
-253	A deprecated zone configuration name provided\nUse -f/--force option

Command error

In all the commands described in this guide, ErrorCode appears if there is an error in the command or an error occurs during execution.

ErrorCode	The error code is listed in Table 4-5 on page 24
ErrorDescription	Description of the error code

Jobs

Some operations, such as firmware update, formatting, and so on, take longer than a few seconds to complete. Instead of making the client wait for the request to complete, the RESTful web service casts those operations as jobs that can be monitored at any time. A job object is created whenever one of these operations is initiated. The web service returns an HTTP status code of 201 ("Created") to indicate the creation of the job. The body of the HTTP response references the job as a URL that can be used to monitor the job.

For example if an update action is performed on an enclosure and the returned HTTP status code is 201 and returned job identifier is job92, HTTP GET on `https://<hostname>:[port]/sndk/infiniflash/job_92` returns the status of the update operation and the completion value as a percentage.

Authentication and authorization

Authentication validates the identity of a client that is trying to access services. It usually involves determining if the client has provided an existing user name with valid credentials, such as a pass token. Once a client is authenticated and attempts to interact with the RESTful web service, authorization determines if the user is allowed to access and invoke actions on specific resources.

Within the InfiniFlash Management stack, two roles are defined: *admin* and *guest*. Any user having the admin role is allowed to perform GET or POST operations, while only GET operations are allowed for guest users.

The web has standardized protocols for authentication.

Keystone authentication

While there are many standard authentication types available, Keystone authentication is used here. It involves sending a user token from the client to the server.

To perform authentication, the client must send a request with the Authorization header set to the user token. For example, the authenticated GET request might be:

```
GET /encl / HTTP/1.1
Authorization: <token>
```

The client needs to send this authorization header with every request it makes to the server.

For an invalid client access, the response header appears as follows:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: <desc>
```

The 401 response tells the client that it is not authorized to access the URL it tried to invoke on. The WWW-Authenticate header specifies which authentication protocol the client should use.

The Keystone container maintains the mapping between each authentication token issued for the user and the user's role (admin or guest). You do not have to specify it in request for authorization.

Anonymous authentication

Apart from Keystone authentication and authorization, anonymous access is also possible.

5

InfiniFlash RESTful interfaces

This section describes all InfiniFlash IF150 enclosure elements and RESTful APIs for InfiniFlash drive cards.

Enclosure operations

This section includes RESTful APIs of all enclosure operations.

Get enclosure properties

HTTP Method – GET

URL\

```
http://<hostname>:[port]/sndk/infiniflash/
encl[id]?[asset|state|overall|topology|version|producttype]
```

or

```
http://<hostname>:[port]/sndk/infiniflash/wwn[logical-
id]?asset|state|overall|topology|version|producttype]
```

Description – This operation returns a list of enclosures available in the system with basic enclosure information.

Request

Syntax – The following are the HTTP header details for REST API:

```
GET /encl[id]?[asset|state|topology|version|overall|producttype] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters – This implementation of the operation uses the following request parameters in the URL:

- `asset` – Returns the enclosure's asset information.
- `state` – Returns the enclosure's state information.
- `topology` – Returns the enclosure's elements topology.
- `version` – Returns version information for enclosure elements.
- `overall` – Overall state of information for various enclosure element types.
- `producttype` - Information about the product type

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

If no id/logical-id is passed in the URL, all of attached enclosure details will be returned.

Response

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-1. Response elements

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, LogicanlIdentifier, State, SerialNumber, Vendor Ancestry: Enclosures
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`ProductID`	Product Identifier of the device Type: String Ancestry: Enclosure
`State`	State of the device Type: String Valid Values: "OK"| "Non-critical" | "Critical" | "Unrecoverable"| "Busy" Ancestry: Enclosure
`Vendor`	Vendor name of the device Type: String Ancestry: Enclosure
`ZoneName`	Zone configuration of the enclosure Type: String Ancestry: Enclosure
`Identify`	LED State of enclosure requested Type: Boolean String Valid Values: “True” | “False” Ancestry: Enclosure
`AlernateDeviceName`	Alternate Name of the device (for example, if the encl<id> is passed, then the wwn name will be the AlternateDeviceName) Type: String

Table 5-2. Enclosure response elements - state

Name	Description
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosures
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, State, ProductID, Vendor Ancestry: Enclosure
`State`	State of the device Type: String Valid Values: "OK"| "Non-critical" | "Critical" | "Unrecoverable"| "Busy" Ancestry: Enclosure

Table 5-3. Enclosure response elements - asset parameter

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, LogicanlIdentifier, State, SerialNumber, Vendor Ancestry: Enclosures
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Vendor`	Vendor name of the device Type: String Ancestry: Enclosure
`ProductID`	Product Identifier of the device Type: String Ancestry: Enclosure

Table 5-3. Enclosure response elements - asset parameter (continued)

Name	Description
`SerialNumber`	Serial Number of the device Type: String Ancestry: Enclosure
`LogicalID`	FRU ID of the Device. Type: String Ancestry: Enclosure
`Fpga0`	Field programmable gate array version 0 Type: String Ancestry: Asset
`Fpga1`	Field programmable gate array version 1 Type: String Ancestry: Enclosure

Table 5-4. Enclosure response elements - version

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, LogicanlIdentifier, State, SerialNumber, Vendor Ancestry: Enclosures
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Elements`	Container of the enclosure elements Type: Container Ancestry: Enclosure
`Element`	Container of individual element Type: String Ancestry: Elements
`ElementName`	Name of the element Type: String Ancestry: Element

Table 5-4. Enclosure response elements - version (continued)

Name	Description
`fwVersion`	Firmware version of the element Type: String Ancestry: Element
`BootCodeVersion`	BootCode version of the element Type: String Ancestry: Element
`PCUBootVersion`	PCU Boot version of the element Type: String Ancestry: Element
`PCUFWVersion`	PCU firmware version of the element Type: String Ancestry: Element

Table 5-5. Enclosure response elements - topology parameter

Name	Description
`Enclosures`	Container for the output. Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for sub-elements Type: Container Ancestry: Enclosures
`HSEs`	Container of HSE element links Type: Container Ancestry: Enclosure
`HSE`	HSE with link Type: Container Ancestry: HSEs
`DSEs`	Container of DSE elements links Type: Container Ancestry: HSE
`DSE`	DSE with link Type: Container Ancestry: DSEs

Table 5-5. Enclosure response elements - topology parameter (continued)

Name	Description
`FanModules`	Container of Fan elements links Type: Container Ancestry: Enclosure
`FanModule`	Fan details Type: Container Ancestry: FanModules
`Fans`	Container of Fan motors elements links Type: Container Ancestry: FanModule
`Fan`	Fan motors details Type: Container Ancestry: Fans
`DeviceSlots`	Container of slot elements links Type: Container Ancestry: Enclosure
`DeviceSlot`	Slot details Type: Container Ancestry: Slots
`TemperatureSensors`	Container of Temperature Sensor elements links Type: Container Ancestry: HSE | DSE | Fan
`TemperatureSensor`	Temperature Sensor details Type: Container Ancestry: TemperatureSensors
`VoltageSensors`	Container of Voltage Sensor elements links Type: Container Ancestry: HSE | DSE
`VoltageSensor`	Voltage Sensor details Type: Container Ancestry: VoltageSensors

Table 5-5. Enclosure response elements - topology parameter (continued)

Name	Description
`State`	State of the device Type: String For Enclosure State: Valid Values: "OK"| "Non-Critical"| "Critical" | "Unrecoverable"| "Busy" For other elements: Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported"
`DeviceName`	Name of the device Type: String
`ProductID`	Product Identifier of the device Type: String Ancestry: Enclosure
`SerialNumber`	Serial Number of the device Type: String
`LogicalID`	FRU ID of the device Type: String Ancestry: Enclosure
`RevisionLevel`	Firmware Version for the device Type: String
`OSName`	OS name of the device Type: String

Table 5-6. Enclosure response elements - overall

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, LogicanlIdentifier, State, SerialNumber, Vendor Ancestry: Enclosures

Table 5-6. Enclosure response elements - overall (continued)

Name	Description
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Elements`	Container of the enclosure elements Type: Container Ancestry: Enclosure
`Element`	Container of Element details Type: String Ancestry: Elements
`Type`	type of the element Type: String Ancestry: Element
`OverallState`	Overall state of the element Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Not Found" Ancestry: Element
`PredictiveFailure`	Failure prediction for the element Type: String Ancestry: Element

Table 5-7. Enclosure response elements - producttype

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, producttype Ancestry: Enclosures
`DeviceName`	Name of the device Type: String Ancestry: Enclosure

Table 5-7. Enclosure response elements - producttype (continued)

Name	Description
`producttype`	Product type of the device Type: String Ancestry: Enclosure Example: IF150

Perform create support bundle

HTTP Method - POST

URL

`http://<hostname>:[port]/sndk/infiniflash/encl[id]?createSUB`

or

`http://<hostname>:[port]/sndk/infiniflash/wwn[logical-id]/createSUB`

Description – This command extracts information about the enclosure and its elements and attached drives, into an archive file for offline failure analysis and debugging. The data extracted includes important SES pages and log pages from all the attached SEs, as well as crash dump data from the attached drives.

This operation accepts a request to create a support bundle. A job identifier is returned as part of the response with the response code 201 (Created).

This command supports single enclosure at a time. If multiple enclosures are attached `encl<id>` or `wwn<logical-id>` should be used to specify the appropriate enclosure.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl[id]?createSUB / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters – `createSUB` is the request parameter in this operation.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 201 (Created) and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-8. Create support bundle response elements

Name	Description
`Jobs`	Container for operation result Type: Container
`Job`	Container for sub elements Type: Container Ancestry: Jobs
`URL`	URL to monitor the created job for the operation. In case of failure, this is not present as a response element. Type: URL Ancestry: Job
`Percentcomplete`	Percent complete of the job Type: URL Ancestry: Job
`Deletiontime`	Deletion time of the job. The value is in seconds Type: URL Ancestry: Job
`State`	Current state of the job Type: String Ancestry: Job Valid Values: "Started" | "Completed" | "In Progress" | "Failed"
`Filename`	Created support bundle file name Type: URL Ancestry: Job
`Jobtype`	Value: “createSUB” Type: String Ancestry: Job

Enclosure update (FFU)

HTTP Method - POST

URL – `http://<hostname>:[port]/sndk/infiniflash/encl<id>?update`

or

URL – `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>?update`

Description – Field Firmware Update (FFU) validates and updates the given firmware image for the specified enclosure elements. Activation triggers a background task that can be tracked using job polling. See the Job operations for further details.

NOTE: This operation is not supported on multiple enclosures. That means a particular enclosure ID has to be given in the URL.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl<id>?update / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: multipart/form-data;boundary=infiniflash-upload-body-separator
Date: date
Authorization: <token>
```

or

```
POST / wwn<logical-id>?update / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: multipart/form-data;boundary=infiniflash-upload-body-separator
Date: date
Authorization: <token>
```

Request Body – Request body is HTTP multipart. The first part can be XML or JSON, and the second part is the binary data for the firmware image.

Sample multipart for XML option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/xml
<?xml version="1.0" encoding="UTF-8"?>
<ffuoption><updateopt>value</updateopt><unode>value</unode><type>value
</type><force>value</force></ffuoption>
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename=
"7.0.0.Generic.bin"
```

```
Content-Type:application/octet-stream
<binary data for the ffu package>
--infiniflash-upload-body-separator--
```

Sample multipart for JSON option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/json
<?xml version="1.0" encoding="UTF-8"?>

{
    "ffuoption":
        {
    "updateopt": "value",
    "unode":     "value",
    "type":      "value",
    "force":     "value"
        }
}
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename=
"7.0.0.Generic.bin"
Content-Type:application/octet-stream

<binary data for the ffu package>
--infiniflash-upload-body-separator--
```

The `ffuoption` value will have the following options:

`updateopt`: Possible values are `validate` and `activate`:
`validate` validates the firmware image of specified enclosures.
`activate` updates the firmware image of the specified enclosures.

`type`: Specifies type of element for which the firmware image is updated or validated (`all/encl/drive/expander`).

`list`: This field is required with type as ‘drive ‘or ‘expander’. It takes comma separated drives or expander.

`force`: Possible values are `true` or `false`. If `true`, specified firmware activated will be forced immediately. This value is invalid during `validate`.

Request Parameters – `update` is the request parameter for this operation.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 200 and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-9. Enclosure update response elements

Name	Description
`Validate`	Container of enclosure elements to be validated Type: Container Children: Zero or more Enclosure
`Enclosure`	Device ID of enclosure to be affected Type: Container Ancestry: Validate
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Logical Id`	Logical ID of the enclosure Type: String Ancestry: Enclosure
`Compatible`	Whether the image is compatible or not Type: String Valid Values: “Yes” | “No” Ancestry: Enclosure
`IOtoSuspend`	Whether the IO will be suspended during firmware upgrade or not Type: String Valid Values: “Yes” | “No” Ancestry: Enclosure
`PowerCycleRequired`	Whether the power cycle will be required during firmware upgrade or not Type: String Valid Values: “Yes” | “No” Ancestry: Enclosure
`Element`	Different element of the enclosure Type: Container Ancestry: Enclosure

Table 5-9. Enclosure update response elements (continued)

Name	Description
`DeviceName`	Name of the device Type: String Ancestry: Element
`Upgradable`	Upgradable state of the enclosure element. The possible output values will be Compatible | Not Required | Updatable | Unreachable Type: String Valid Values: “Yes” | “No” Ancestry: Enclosure
`Current Ver`	Current firmware version of the enclosure’s element Type: String Ancestry: Element
`New Ver`	New firmware version of the enclosure’s element Type: String Ancestry: Element
`OS Name`	OS name for the enclosure element Type: String Ancestry: Element
`Url`	Job URL, only available during activate operation Type: String Ancestry: Enclosure

NOTE: Should the update fail for a drive, the drive ID will not appear in the update summary. Run the validate command on the enclosure again to determine the failed drive ID.

Firmware update will fail if the device is in a busy state. Run the update command with the enclosure option to clear the failure.

Some situations require a power cycle after an update. If power is not cycled, commands such as “version command on enclosure” could return improper or empty results because the underlying hardware components are in inconsistent states.

Locate enclosure

HTTP Method – POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl[id]?locate`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn[logical-id]?locate`

Description - This operation identifies the specified enclosure by blinking the Identify LED.

This command supports a single enclosure at a time. If multiple enclosures are attached, `encl<id>` or `wwn<logical-id>` should be used to specify the appropriate enclosure.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl<id>?locate / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

or

```
POST / wwn[logical-id]?locate / HTTP/1.1 Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml Date: date
Authorization: <token>
```

Request Body – Request body should be json/xml:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Identifyoption>
  <identify>true</identify>
</Identifyoption>
```

JSON

```
{
    "Identifyoption":
        {
            "identify": "true"
        }
}
```

Request Parameters - `locate` is the request parameter used to identify the enclosure.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-10. Locate enclosure request elements

Name	Description
`IdentifyOption`	Container for identify enclosure Type: Container
`Identify`	LED State of enclosure requested Type: Boolean String Valid Values: “true” | “false” Ancestry: IdentifyOption

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-11. Locate enclosure response elements

Name	Description
`Enclosures`	Container for Container of enclosures Type: Container
`Enclosure`	Container for enclosure identify Type: Container Ancestry: Enclosures
`DeviceName`	Name of the device Type: String Ancestry: Enclosure
`Identify`	LED state of enclosure Type: Boolean String Valid Values: “true” | “false” Ancestry: Enclosure

Enclosure Reboot

HTTP Method - POST

URL - `http://<hostname>:[port]/sndk/infiniflash/encl[id]?reboot`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn[logical-id]?reboot`

Description - This operation reboots the specified enclosure.

This command supports a single enclosure at a time. If multiple enclosures are attached `encl<id>` or `wwn<logical-id>` should be used to specify the appropriate enclosure.

Request

Syntax - Below are the http header details for REST API:

```
POST /encl[id]?[reboot] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Parameters - This implementation of the operation uses the following request parameters in the URL:

- `reboot` – reboots the enclosure.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Response

Response Status code - On successful acceptance of the request, http response status code is returned as 200. On failure, http status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-12. Response elements - all enclosures (encl)

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, RebootStatus. Ancestry: Enclosures
`DeviceName`	Name of the device. Type: String Ancestry: Enclosure
`RebootStatus`	Operation execution status. Type: String Ancestry: Enclosure

Table 5-13. Response elements - single enclosure (encl)

Name	Description
`Enclosures`	Container for the output Type: Container Children: Zero or more Enclosure
`Enclosure`	Container for enclosure information Type: Container Children: DeviceName, RebootStatus. Ancestry: Enclosures
`DeviceName`	Name of the device. Type: String Ancestry: Enclosure
`RebootStatus`	Operation execution status. Type: String Ancestry: Enclosure

Fan operations

This section includes RESTful APIs of all the fan operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/fan[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/fan[id]?[state]`

Description - This operation returns basic information about a particular fan or a list of fans available in the specified enclosure.

Requests

Syntax - The following are the http header details for REST API:

```
GET /encl<id>/fan<id>?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Parameters - This implementation of the operation uses the following request parameters in URL:

- `State` - Returns the enclosure's state information. This is per fan in a specified enclosure.
- `Operationalparam` - Returns the enclosure's elements reference URLs. This is per fan in a specified enclosure.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-14. Fan operations response elements - all fans

Name	Description
`FanModules`	Container for output Type: Container Children: Zero or more FanModule
`FanModule`	Container for fan details Type: Container Ancestry: FanModules
`DeviceName`	Name of the device Type: String Ancestry: Fan
`SerialNumber`	Fan serial number Type: string Ancestry: FanModule
`Identify`	Fan identify status Type: Boolean String Valid Values: “ON” | “OFF” Ancestry: FanModule
`State`	FanModule state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: FanModule

Table 5-15. Fan operations response elements - single fan

Name	Description
`FanModules`	Container for Output Type: Container Children: Zero or more FanModule
`FanModule`	Container for FanModule details Type: Container Ancestry: FanModules
`DeviceName`	Name of the device Type: String Ancestry: FanModule
`SerialNumber`	Fan serial number Type: string Ancestry: FanModule
`Identify`	Fan identify status Type: Boolean String Valid Values: “ON” | “OFF” Ancestry: FanModule
`State`	Fan state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: FanModule

Table 5-16. Fan operations response elements - state

Name	Description
`FanModules`	Container for Output Type: Container Children: Zero or more FanModule
`FanModule`	Container for FanModule details Type: FanModule
`DeviceName`	Name of the device Type: String Ancestry: FanModule
`State`	Status of the FanModule Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: FanModule

Locate fan

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/fan[id]?locate`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/fan[id]?locate`

Description - This operation identifies the specified fan in the enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/fan<id>?locate / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Body - Request body should be json/xml:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Identifyoption>
  <identify>true</identify>
```

```
    </Identifyoption>
```

JSON

```
    {
        "Identifyoption":
            {
                "identify": "true"
            }
    }
```

Request Parameters - The request parameter `locate` is used to identify the fan elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-17. Locate fan request elements

Name	Description
`IdentifyOption`	Container for identify fan elements Type: Container
`Identify`	LED state of fan requested Type: Boolean String Valid Values: “True” | “False” Ancestry: IdentifyOption

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-18. Locate fan response elements

Name	Description
`FanModules`	Container for Output Type: Container Children: Zero or more FanModule

Table 5-18. Locate fan response elements (continued)

Name	Description
`FanModule`	Container for fan details Type: Container Ancestry: FanModules
`DeviceName`	Name of the device Type: String Ancestry: FanModule
`Identify`	LED state of fan Type: Boolean String Valid Values: “True” | “False” Ancestry: FanModule

Fan motor operations

This section includes RESTful APIs of all the fan motor operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/fan<id>/fm[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/fan<id>/fm[id]?[state]`

Description - This operation returns information about two fan motors or any one fan motor in the fan module available in the specified enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET encl<id>/fan<id>/fm<id>?[state|operationalparam] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters - This implementation of the operation uses the following request parameters in the URL:

- `State` - Returns the fan’s state information. This is per fan in a specified fan module.
- `Operationalparam` - Returns the enclosure’s elements reference URLs. This is per fan in a specified enclosure.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, http response status code is returned as 200. On failure, http status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-19. Fan motor response elements - all fans

Name	Description
`Fans`	Container for Output Type: Container Children: Zero or more Fan
`Fan`	Container for fan details Type: Container Ancestry: Fans
`DeviceName`	Name of the device Type: String Ancestry: Fan
`CurrentSpeed`	Fan speed in RPM Type: Integer Ancestry: Fan
`CurrentSpeedMode`	Fan speed mode Type: Enum Valid Values: Lowest, Highest, Second Lowest ... Ancestry: Fan

Table 5-19. Fan motor response elements - all fans (continued)

Name	Description
`State`	Fan state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: Fan

Table 5-20. Fan motor response elements - single fan

Name	Description
`Fans`	Container for Output Type: Container Children: Zero or more Fan
`Fan`	Container for fan details Type: Container Ancestry: Fans
`DeviceName`	Name of the device Type: String Ancestry: Fan
`CurrentSpeed`	Fan speed in RPM Type: Integer Ancestry: Fan
`CurrentSpeedMode`	Fan speed mode Type: Enum Valid Values: TURBO | ... Ancestry: Fan
`State`	Fan state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: Fan

Table 5-21. Fan motor response elements - state parameter

Name	Description
`Fans`	Container for Output Type: Container Children: Zero or more Fan
`Fan`	Container for fan details Type: Container Ancestry: Fans
`DeviceName`	Name of the device Type: String Ancestry: Fan
`State`	Status of the fan Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: Fan

Power supply unit operations

This section includes RESTful APIs of all power supply unit operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/psu[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/psu[id]?[state]`

Description - This operation returns basic information about a particular PSU or a list of PSUs available in the specified enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET /encl<id>/psu[id]?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters - This implementation of the operation uses the following request parameters in URL:

- `State` - Returns the enclosure's state information. This is per PSU in a specified enclosure.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-22. PSU response elements - all power supplies

Name	Description
`PowerSupplies`	Container for Output Type: Container Children: Zero or more PowerSupply
`PowerSupply`	Container for power supply unit details Type: Container Ancestry: PowerSupplies
`DeviceName`	Name of the device Type: String Ancestry: PowerSupply
`State`	Status of the power supply unit Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DeviceName
`DC Over Current`	DC Over Current Type: Boolean String Valid Values: "true" | "false" Ancestry: DeviceName

Table 5-22. PSU response elements - all power supplies (continued)

Name	Description
`DC Under Voltage`	DC Under Voltage Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`DC Over Voltage`	DC Over Voltage Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`DC Fail`	DC Over Voltage Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`AC Fail`	DC Over Voltage Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`Temp Warning`	DC Over Voltage. Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`Power Supply Unit Off`	DC Over Voltage Type: Boolean String Valid Values: “true” | “false” Ancestry: DeviceName
`SerialNumber`	DC Over Voltage Type: String Ancestry: Power Supply Unit Off

Table 5-23. PSU response elements - single power supply

Name	Description
`PowerSupplies`	Container for Output Type: Container Children: Zero or more PowerSupply

Table 5-23. PSU response elements - single power supply (continued)

Name	Description
`PowerSupply`	Container for power supply unit details Type: Container Ancestry: PowerSupplies
`DC Over Current`	DC Over Current Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`DC Under Voltage`	DC Under Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`DC Over Voltage`	DC Over Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`DC Fail`	DC Over Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`AC Fail`	DC Over Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`Temp Warning`	DC Over Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`Power Supply Unit Off`	DC Over Voltage Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName
`SerialNumber`	DC Over Voltage Type: String Ancestry: Power Supply Unit Off

Table 5-24. PSU response elements - state parameter

Name	Description
`PowerSupplies`	Container for Output Type: Container Children: Zero or more PowerSupply
`PowerSupply`	Container for power supply unit details Type: Container Ancestry: PowerSupplies
`DeviceName`	Name of the device Type: String Ancestry: PowerSupply
`State`	Status of the power supply unit Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DeviceName

Host SE operations

This section includes RESTful APIs of all host SE operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse[id]?[state]`

Description - This operation returns basic information about a particular host SE or list of HSEs available in the specified enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET /encl<id>/hse<id>?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters - The request parameter `state` is the operation applicable to a particular HSE ID.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements

Table 5-25. HSE response elements - all HSE

Name	Description
HSEs	Container for Output Type: Container Children: Zero or more HSE

Table 5-25. HSE response elements - all HSE (continued)

Name	Description
`HSE`	HSE element details Type: Container Ancestry: HSEs
`DeviceName`	Name of the device Type: String Ancestry: HSE
`State`	Status of the host SE Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: HSE
`SASAddress`	SAS address of the host SE Type: String Ancestry: HSE
`Type`	Type of the SE; the value is HSE Type: String Ancestry: HSE
`Identify`	Identify status of the SE element Type: String Ancestry: HSE
`FirmwareVersion`	Firmware version of the device Type: String Ancestry: HSE
`SerialNumber`	Serial Number of the device Type: String Ancestry: HSE
`WWN`	WWN for the element Type: String Ancestry: HSE
`BootCodeVersion`	BootCodeVersion for the element Type: String Ancestry: HSE
`ModelNumber`	Model Number for the element Type: String Ancestry: HSE

Table 5-25. HSE response elements - all HSE (continued)

Name	Description
`OSName`	OS Name for the element Type: String Ancestry: HSE

Table 5-26. HSE Response elements - single HSE

Name	Description
`HSEs`	Container for Output Type: Container Children: Zero or more HSE
`HSE`	HSE element details Type: Container Ancestry: HSEs
`DeviceName`	Name of the device Type: String Ancestry: HSE
`State`	Status of the host SE Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: HSE
`SASAddress`	SAS address of the host SE Type: String Ancestry: HSE
`Type`	Type of the SE Type: String Ancestry: HSE
`Identify`	Identify status of the SE element Type: String Ancestry: HSE
`FirmwareVersion`	Firmware version of the device Type: String Ancestry: HSE

Table 5-26. HSE Response elements - single HSE (continued)

Name	Description
`SerialNumber`	Serial number of the device Type: String Ancestry: HSE
`WWN`	WWN for the element Type: String Ancestry: HSE
`BootCodeVersion`	BootCodeVersion for the element Type: String Ancestry: HSE
`ModelNumber`	Model number for the element Type: String Ancestry: HSE
`OSName`	OS name for the element Type: String Ancestry: HSE

Table 5-27. HSE response elements - state parameter

Name	Description
`HSEs`	Container for Output Type: Container Children: Zero or more HSE
`HSE`	HSE element details Type: Container Ancestry: HSEs
`DeviceName`	Name of the device Type: String Ancestry: HSE
`State`	Fan state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: HSE

Perform self-test

HTTP Method - POST

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse[id]?selftest`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse[id]?selftest`

Description - This command runs a self-test diagnostic in the specified enclosure HSEs.

Requests

Syntax - `Below are the http header details for REST API:`

```
POST /encl<id>/hse<id>?selftest / HTTP/1.1
Host: hostname
Content-Type: application/json | application/xml
Accept: application/json,application/xml
Date: date
Authorization : <token>
```

Request Parameters -The request parameter for the operation is `selftest`.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 201 (Created) and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-28. Self-test response elements

Name	Description
HSEs	Container for Output Type: Container Children: Zero or more HSE
HSE	HSE element details Type: Container Ancestry: HSEs

Table 5-28. Self-test response elements **(continued)**

Name	Description
`DeviceName`	Name of the device Type: Container Ancestry: HSE
`Status`	Value: “Self-test execution completed Successfully” if operation is successful Type: String Ancestry: HSE
`ErrorCode`	Error Code if the operation fails, refer to the Error Codes section Type: String Ancestry: HSE
`ErrorDescription`	Error description if the operation fails Type: String Ancestry: HSE

Locate HSE

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/hse[id]?locate`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse[id]?locate`

Description - This operation identifies a specified HSE in the enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/hse<id>?locate / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Body - Request body should be json/xml:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<identifyoption>
<identify>true</identify>
</identifyoption>
```

JSON

```
{
    "identifyoption":
        {
            "identify": "true"
        }
}
```

Request Parameters - `locate` is the request parameter used in conjunction to identify the HSE elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-29. Locate HSE request elements

Name	Description
`IdentifyOption`	Container for identify HSE elements Type: Container
`Identify`	LED state of HSE requested Type: Boolean String Valid Values: “True” | “False” Ancestry: IdentifyOption

Responses

Response Status code - On successful acceptance of the request, http response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-30. Locate HSE response elements

Name	Description
`HSEs`	Container for Output Type: Container Children: Zero or more HSE
`HSE`	Container for HSE details Type: Container Ancestry: HSEs
`DeviceName`	Name of the device Type: String Ancestry: HSE

Table 5-30. Locate HSE response elements **(continued)**

Name	Description
`Identify`	State of LED attached to HSE Type: Boolean String Valid Values: “True” | “False” Ancestry: HSE

DSE operations

This section includes RESTful APIs of all the DSE operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse<id>/d[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<encl-id>/hse<id>/d[id]?[state]`

Description - This operation returns basic information about a particular DSE or a list of DSEs available for the specified host SE.

Requests

Syntax - The following are the http header details for REST API:

```
GET /encl<id>/hse<id>/d<id>?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters - `state` is the request parameter in this operation applicable for a particular DSE ID.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements:

Table 5-31. DSE response elements - all DSE

Name	Description
`DSEs`	Container for Output Type: Container Children: Zero or more DSE
`DSE`	Container for DSE details Type: Container Ancestry: DSEs
`DeviceName`	Name of the device Type: String Ancestry: DSE
`State`	Status of the HSE Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DSE
`SASAddress`	SAS address of the HSE Type: String Ancestry: DSE
`Type`	Type of the SE; the value is DSE Type: String Ancestry: DSE
`Identify`	Identify status of the SE element Type: String Ancestry: DSE
`SerialNumber`	Serial Number of the device Type: String Ancestry: DSE
`FirmwareVersion`	Firmware version of the device Type: String Ancestry: DSE
`WWN`	WWN for the element Type: String Ancestry: DSE

Table 5-31. DSE response elements - all DSE (continued)

Name	Description
`BootCodeVersion`	BootCodeVersion for the element Type: String Ancestry: DSE
`ModelNumber`	Model number for the element Type: String Ancestry: DSE
`OSName`	OS name for the element Type: String Ancestry: DSE

Table 5-32. DSE response elements - single DSE

Name	Description
`DSEs`	Container for Output Type: Container Children: Zero or more DSE
`DSE`	Container for DSE details Type: Container Ancestry: DSEs
`DeviceName`	Name of the device Type: String Ancestry: DSE
`State`	Fan state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported " Ancestry: DSE
`SASAddress`	SAS address of the HSE Type: String Ancestry: DSE
`Type`	Type of the SE Type: String Ancestry: DSE

Table 5-32. DSE response elements - single DSE (continued)

Name	Description
`Identify`	Identify status of the SE element Type: String Ancestry: DSE
`SerialNumber`	Serial number of the device Type: String Ancestry: DSE
`FirmwareVersion`	Firmware version of the device Type: String Ancestry: DSE
`WWN`	WWN for the element Type: String Ancestry: DSE
`BootCodeVersion`	BootCodeVersion for the element Type: String Ancestry: DSE
`ModelNumber`	Model number for the element Type: String Ancestry: DSE
`OSName`	OS name for the element Type: String Ancestry: DSE

Table 5-33. DSE response elements - state parameter

Name	Description
`DSEs`	Container for Output Type: Container Children: Zero or more DSE
`DSE`	Container for DSE details Type: Container Ancestry: DSEs
`DeviceName`	Name of the device Type: String Ancestry: DSE

Table 5-33. DSE response elements - state parameter

Name	Description
State	SE state Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DSE

Perform self-test

HTTP Method - POST

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse<id>/d[id]?selftest`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse<id>/d[id]?selftest`

Description - This command runs a self-test diagnostic in the specified enclosure DSEs.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/hse<id>/d<id>?selftest / HTTP/1.1
Host: hostname
Content-Type: application/json | application/xml
Accept: application/json,application/xml
Date: date
Authorization : <token>
```

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

NOTE: In the current InfiniFlash drive card firmware, extended and short self-test are the same.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 201 (Created) and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-34. DSE self-test response elements

Name	Description
`DSEs`	Container for Output Type: Container Children: Zero or more DSE
`DSE`	Container for DSE details Type: Container Ancestry: DSEs
`DeviceName`	Name of the device Type: Container Ancestry: DSE
Status	Value: “Self-test execution completed successfully” if operation is successful Type: String Ancestry: DSE
ErrorCode	Code if the operation fails; refer to Error Codes section Type: String Ancestry: DSE
ErrorDescription	Error description if the operation fails Type: String Ancestry: DSE

Locate DSE

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/hse<id>/d[id]?locate`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse<id>/d[id]?locate`

Description - This operation identifies specified DSE in the enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/d<id>?locate / HTTP/1.1
   Host: hostname
   Accept: application/json, application/xml
   Content-Type: application/json (or) application/xml
   Date: date
```

```
Authorization: <token>
```

Request Body - Request body should be json/xml:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Identifyoption>
  <identify>true</identify>
</Identifyoption>
```

JSON

```
{
    "Identifyoption":
        {
            "identify": "true"
        }
}
```

Request Parameters - The request parameter `locate` is used to identify the DSE elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-35. Locate DSE request elements

Name	Description
`IdentifyOption`	Container for identify DSE elements Type: Container
`Identify`	LED state of DSE requested Type: Boolean String Valid Values: “True” | “False” Ancestry: IdentifyOption

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-36. Locate DSE response elements

Name	Description
DSEs	Container for Output Type: Container Children: Zero or more DSE
DSE	Container for DSE details Type: Container Ancestry: DSEs
DeviceName	Name of the device Type: String Ancestry: DSE
Identify	State of LED attached to DSE Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName

Temperature sensor operations

This section includes RESTful APIs of all the temperature sensor operations.

Get properties

HTTP Method - GET

URL -

```
http://<hostname>:[port]/sndk/infiniflash/encl<id>/fan[hse[d[id]]/hse<id>[/d<id>
] | /ts[id]?[state]
```

or

```
http://<hostname>:[port]/sndk/infiniflash/wwn<logical-
id>/fan[hse[d[id]]/hse<id>[/d<id>] | /ts[id]?[state]
```

Description - This operation returns basic information for a particular temperature sensor or list of temperature sensors available for the specified HSE, DSE, or fan.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET /encl<id>/fan<id>/hse<id>[/d<id>]/ts[id]?[state] / HTTP/1.1
  Host: hostname
  Accept: application/json, application/xml
  Date: date
 Authorization : <token>
```

Request Parameters - `state` is the request parameter in this operation applicable for a particular temperature sensor within a particular HSE, DSE, or Fan.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements:

Table 5-37. Temperature sensor response elements

Name	Description
`TemperatureSensors`	Container for Output Type: Container Children: TemperatureSensor
`TemperatureSensor`	Temperature sensor with link Type: Container Ancestry: TemperatureSensors
`DeviceName`	Name of the device Type: String Ancestry: TemperatureSensor
`State`	Status of the host SE Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: TemperatureSensor
`CurrentTemperature`	Temperature in Celsius Type: String Ancestry: TemperatureSensor
`TemperatureLevel`	Level of the temperature Type: Enum Valid Values: Low | Medium | Normal | High Ancestry: TemperatureSensor

Table 5-37. Temperature sensor response elements (continued)

Name	Description
`HighCriticalThreshold`	Threshold value for high critical in Celsius Type: Integer Ancestry: TemperatureSensor
`HighWarningThreshold`	Threshold value for high warning in Celsius Type: Integer Ancestry: TemperatureSensor
`LowCriticalThreshold`	Threshold value for low critical in Celsius Type: Integer Ancestry: TemperatureSensor
`LowWarningThreshold`	Threshold value for low warning in Celsius Type: Integer Ancestry: TemperatureSensor

Table 5-38. Single temperature sensor response

Name	Description
`TemperatureSensors`	Container for Output Type: Container Children: TemperatureSensor
`TemperatureSensor`	Temperature sensor with link Type: Container Ancestry: TemperatureSensors
`DeviceName`	Name of the device Type: String Ancestry: TemperatureSensor
`State`	Status of the host SE. Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: TemperatureSensor
`CurrentTemperature`	Temperature in Celsius Type: string Ancestry: TemperatureSensor

Table 5-38. Single temperature sensor response (continued)

Name	Description
`TemperatureLevel`	Level of the temperature Type: Enum Valid Values: Low | Medium| Normal | High Ancestry: TemperatureSensor
`HighCriticalThreshold`	Threshold value for high critical in Celsius Type: Integer Ancestry: TemperatureSensor
`HighWarningThreshold`	Threshold value for high warning in Celsius Type: Integer Ancestry: TemperatureSensor
`LowCriticalThreshold`	Threshold value for low critical in Celsius Type: Integer Ancestry: TemperatureSensor
`LowWarningThreshold`	Threshold value for low warning in Celsius Type: Integer Ancestry: TemperatureSensor

Table 5-39. Temperature sensor response elements - state

Name	Description
`TemperatureSensors`	Container for Output Type: Container Children: TemperatureSensor
`TemperatureSensor`	Temperature sensor with link Type: Container Ancestry: TemperatureSensors
`DeviceName`	Name of the device Type: String Ancestry: TemperatureSensor
`State`	Status of the fan Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: TemperatureSensor

Voltage sensor operations

This section includes the RESTful APIs of all the voltage sensor operations.

Get properties

HTTP Method - GET

URL -
`http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse<id>[/d<id>]/vs[id]?[state]`

or

`http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse<id>[/d<id>]/vs[id]?[state]`

Description - This operation returns basic information for a particular voltage sensor or list of voltage sensors available within the specified HSE or a DSE.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET /encl<id>/hse<id>[/d<id>]/vs[id]?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters -The request parameter `state` is the operation applicable for a particular voltage sensor within a particular HSE or DSE.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements:

Table 5-40. Voltage sensor response elements - all sensors

Name	Description
`VoltageSensors`	Container for Output Type: Container Children: Zero or more VoltageSensor
`VoltageSensor`	Voltage sensor details Type: Container Ancestry: VoltageSensors
`DeviceName`	Name of the device Type: String Ancestry: VoltageSensor
`State`	Status of the host SE Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: VoltageSensor
`CurrentVoltage`	Current voltage in Volts Type: String Ancestry: VoltageSensor
`VoltageLevel`	Level of the voltage Type: Enum Valid Values: OVER_VOLT_WARNING | NORMAL_UNDER_VOLT_WARNING | CRITICAL_OVER_VOLT | CRITICAL_UNDER_VOLT Ancestry: VoltageSensor
`HighCriticalThreshold`	Threshold value for high critical in volts Type: Integer Ancestry: VoltageSensor
`HighWarningThreshold`	Threshold value for high warning in volts Type: Integer Ancestry: VoltageSensor
`LowCriticalThreshold`	Threshold value for low critical in volts Type: Integer Ancestry: VoltageSensor
`LowWarningThreshold`	Threshold value for low warning in volts Type: Integer Ancestry: VoltageSensor

Table 5-41. Voltage sensor response elements for a HSE/DSE

Name	Description
`VoltageSensors`	Container for Output Type: Container Children: Zero or more VoltageSensor
`VoltageSensor`	Voltage sensor details Type: Container Ancestry: VoltageSensors
`DeviceName`	Name of the device Type: String Ancestry: VoltageSensor
`State`	Status of the voltage sensor Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: VoltageSensor
`CurrentVoltage`	Current voltage in volts Type: String Ancestry: VoltageSensor
`VoltageLevel`	Level of the voltage Type: Enum Valid Values: Low | Medium | Normal | High Ancestry: VoltageSensor
`HighCriticalThreshold`	Threshold value for high critical in volts Type: Integer Ancestry: VoltageSensor
`HighWarningThreshold`	Threshold value for high warning in volts Type: Integer Ancestry: VoltageSensor
`LowCriticalThreshold`	Threshold value for low critical in volts Type: Integer Ancestry: VoltageSensor
`LowWarningThreshold`	Threshold value for low warning in volts Type: Integer Ancestry: VoltageSensor

Table 5-42. Voltage sensor response elements - state

Name	Description
`VoltageSensors`	Container for Output Type: Container Children: Zero or more VoltageSensor
`VoltageSensor`	Voltage sensor details Type: Container
`DeviceName`	Name of the device. Type: String
`State`	Status of the Voltage Sensor. Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | " Unsupported" Ancestry: VoltageSensor

Device slot operations

This section includes the RESTful APIs of all the device slot operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/slot<id>`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot<id>`

Description - This operation returns basic information about a particular device slot or list of device slots available in the specified enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
GET /encl<id>/slot<id>?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters -The request parameter `state` is the operation applicable for a particular device slot.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements:

Table 5-43. Device slot response elements - all device slots

Name	Description
`DeviceSlots`	Container for Output Type: Container Children: Zero or more DeviceSlot
`DeviceSlot`	Device slot details Type: Container Ancestry: DeviceSlots
`DeviceName`	Name of the device Type: String Ancestry: DeviceSlot
`State`	Status of the device slot Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DeviceSlot
`SlotAddress`	Slot address of the device slot Type: String Ancestry: DeviceSlot
`Activated`	This field currently has no meaning
`Identify`	State of visual Identifier of a device slot Type: Boolean String Valid Values: “On” | “Off” Ancestry: DeviceSlot

Table 5-43. Device slot response elements - all device slots (continued)

Name	Description
`Phys`	Array of device PHY elements Type: Container
`Phy`	Device PHY details Type: Container Ancestry: PHYs
`PhyIDs`	PhyIDs associated with device slots Type: Integer Ancestry: DeviceSlot
`PhySASAddresses`	PhySASAddresses associated with device slots Type: String Ancestry: DeviceSlot (only visible if the drive is present in a specific slot)
`PhyAttachedSASAddresses`	PhyAttachedSASAddresses associated with device slots Type: String Ancestry: DeviceSlot (only visible if the drive is present in a specific slot)

Table 5-44. Device slot response elements - state parameter

Name	Description
`DeviceSlots`	Container for Output Type: Container Children: Zero or more DeviceSlot
`DeviceSlot`	Device slot details Type: Container
`DeviceName`	Name of the device Type: String
`State`	Status of the device slot Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: DeviceSlot

Attached drive reset

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/slot[id]?devreset`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot[id]?devreset`

Description - This operation disables and enables the slot, power cycling the attached drive.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/slot<id>?devreset / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters -The request parameter `devreset` is used to reset the slot elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements

Table 5-45. Attach drive reset response elements

Name	Description
`DeviceSlots`	Container for Output Type: Container Children: Zero or more DeviceSlot
`DeviceSlot`	Container for slot details Type: Container Ancestry: DeviceSlots
`SlotName`	Name of the slot Type: String Ancestry: Slot

Table 5-45. Attach drive reset response elements (continued)

Name	Description
ResetStatus	Reset status for the slot Type: String Valid Values: “No devices found” if no drive/IGBlank attached to the slot | “Device type not supported” if IGBlank is attached to the slot | “Success” if a drive is attached to the slot and successfully reset | “Failure” for any other reason Ancestry: DeviceName

Locate slot

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/slot[id]?locate`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot[id]?locate`

Description - This operation identifies a specified drive slot in the enclosure.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/slot<id>?locate / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Body - Request body should be json/xml:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Identifyoption>
  <identify>true</identify>
</Identifyoption>
```

JSON

```
{
    "Identifyoption":
        {
            "identify": "true"
        }
}
```

Request Parameters - The request parameter `locate` is used to identify the slot elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-46. Locate slot request elements

Name	Description
`IdentifyOption`	Container for identify slot elements Type: Container
`Identify`	LED state of slot requested Type: Boolean String Valid Values: “True” | “False” Ancestry: IdentifyOption

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-47. Locate slot response elements

Name	Description
`Slots`	Container for Output Type: Container Children: Zero or more Slot
`Slot`	Container for slot details Type: Container Ancestry: Slots
`DeviceName`	Name of the device Type: String Ancestry: Slot
`Identify`	State of LED attached to slot Type: Boolean String Valid Values: “True” | “False” Ancestry: DeviceName

Attached drive shutdown

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/slot[id]?shutdown`

or

URL - `http//<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot[id]?shutdown`

Description - This operation disables the slot, powering off the attached drive.

Requests

Syntax - Below are the HTTP header details for REST API:

```
POST /encl<id>/slot<id>?shutdown / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters - The request parameter `shutdown` is used to power off the slot elements.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-48. Attached drive shutdown response elements

Name	Description
`DeviceSlots`	Container for array of slot elements Type: Container
`DeviceSlot`	Container for slot details Type: Container Ancestry: DeviceSlots
`SlotName`	Name of the slot Type: String Ancestry: DeviceSlot

Table 5-48. Attached drive shutdown response elements (continued)

Name	Description
`ShutdownStatus`	Shutdown status for the slot Type: String Valid Values: “No devices found” if no drive/IGBlank attached to the slot | “Device type not supported” if IGBlank is attached to the slot | “Success” if a drive is attached to the slot and successfully reset | “Failure” for any other reason Ancestry: DeviceSlot

InfiniFlash drive card operations

This section includes the RESTful APIs for all InfiniFlash drive card operations.

Get list of InfiniFlash drive cards

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl[id]/slot/drive`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn[logical-id]/slot/drive`

Description - This operation returns a list of InfiniFlash drive cards in a particular enclosure with their basic properties.

Requests

Syntax:

```
GET /encl<id>/slot/drive / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Parameters - This implementation of the operation does not use request parameters.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-49. InfiniFlash drive card response elements

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive

Table 5-49. InfiniFlash drive card response elements (continued)

Name	Description
`Drive`	Drive details Type: Container Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`Size`	Capacity in bytes Type: Integer Ancestry: Drive
`State`	State of the device Type: Any of the possible InfiniFlash drive card states (see Drive states) Ancestry: Drive
`Bootable`	Boot device indicator Type: Boolean String Valid Values: “True” | “False” Ancestry: Drive
`SerialNumber`	Serial Number of the device Type: String Ancestry: Drive
`Vendor`	Model of the device Type: String Ancestry: Drive
`OSName`	OS Device Name Type: String Ancestry: Drive

Get InfiniFlash drive card properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/slot[id]/drive?[parameter]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot[id]/drive?[parameter]`

Description - This operation returns basic information about the drive attached to a specific enclosure slot.

NOTE: To perform this operation on all InfiniFlash drive cards attached to a particular enclosure, the URL should be:
`http//<hostname>:[port]/sndk/infiniflash/encl[id]/slot/drive?[parameter]`

To perform this operation on an InfiniFlash drive card using its OS Name, the URL should be:
`http://<hostname>:[port]/sndk/infiniflash/<OSName>?[parameter]`

Requests

Syntax - The following are the HTTP header details where 'p' is a request parameter described in the section Request Parameters:

```
GET /encl<id>/slot<id>/drive?p / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Parameters - This implementation of the operation supports the following request parameters:

- `<no parameter>`: Returns basic information about the InfiniFlash drive cards.
- `asset`: Returns asset information such as drive name, serial number, capacity, and revision level.
- `geometry`: Returns geometry information such as capacity.
- `state`: Returns state information of InfiniFlash drive cards.
- `stats`: Returns statistics such as the total number of read commands and the total number of write commands.
- `getsmart`: Returns all S.M.A.R.T (Self-Monitoring, Analysis and Reporting Technology) attributes, status, and summarizes as a list of attribute value pairs.
- multipath : Show multipath for drive(s). Multipath name display is not supported for OS names

Request Headers: This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements: This implementation of the operation does not use request elements.

Responses

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements - The following are the response elements for each type of request parameters and sample outputs.

Table 5-50. Get InfiniFlash drive card properties – no parameter

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Drive details Type: Container Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`Capacity`	Capacity in bytes Type: Integer Ancestry: Drive
`State`	State of the device Type: String Value: “Good” | “Not Ready” | “Warning” | "Critical" | "Unknown" Ancestry: Drive
`BootDevice`	Boot device indicator Type: Boolean Ancestry: Drive
`SerialNumber`	Serial number of the device Type: String Ancestry: Drive
`Model`	Model of the device Type: String Ancestry: Drive
`OSName`	OS device name Type: String Ancestry: Drive

Table 5-51. InfiniFlash drive card properties – asset

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Asset details Type: Container Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`Vendor`	Device vendor Type: String Ancestry: Drive
`ProductID`	Product identifier of the device Type: String Ancestry: Drive
`SerialNumber`	Serial number of the device Type: String Ancestry: Drive
`WWNLUN`	WWNLUN identifier of the device Type: String Ancestry: Drive
`WWNTarget`	WWNTarget identifier of the device Type: String Ancestry: Drive
`OSName`	OS name for the drive Type: String Ancestry: Drive
`RevisionLevel`	Revision level for the drive Type: String Ancestry: Drive
`CBCSupport`	CBC supported or not for the drive Type: String Ancestry: Drive

Table 5-51. InfiniFlash drive card properties – asset (continued)

Name	Description
`FormFactor`	Form factor for the drive Type: String Ancestry: Drive

Table 5-52. InfiniFlash drive card properties – state

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Drive's State details Type: Drives Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`State`	State of the device Type: String Value: “Good” | “Not Ready” | “Warning” | "Critical" | "Unknown" Ancestry: Drive
`OSName`	OS name for the drive Type: String Ancestry: Drive
`Description`	This field shows the reason for the change in the BSSD's state. This field exists if the BSSD's state is other than "OK."

Table 5-53. InfiniFlash drive card properties – statistics

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Drive's Statistics details Type: Drives Ancestry: Drives

Table 5-53. InfiniFlash drive card properties – statistics (continued)

Name	Description
`DeviceName`	Name of the device Type: String Ancestry: Drive
`TotalReadCommands`	Number of read commands Type: Integer Ancestry: Drive
`TotalWriteCommands`	Number of write commands Type: Integer Ancestry: Drive
`NumberOfLogicalBlocksReceived`	Number of logical blocks received Type: Integer Ancestry: Statistics
`NumberOfLogicalBlocksTransmitted`	Number of logical blocks transmitted Type: Integer Ancestry: Drive
`ReadCommandProcessingIntervals`	Read command processing intervals Type: Integer Ancestry: Drive
`WriteCommandProcessingIntervals`	Write command processing intervals Type: Integer Ancestry: Drive
`WeightedReadPlusWriteCommands`	Weighted number of read commands plus write Commands Type: Integer Ancestry: Drive
`WeightedReadPlusWriteCommandProc`	Weighted read command processing plus write command processing Type: Integer Ancestry: Drive
`TotalCorrectedWriteErrors`	Total corrected read errors Type: Integer Ancestry: Drive
`TotalUncorrectedWriteErrors`	Total uncorrected read errors Type: Integer Ancestry: Statistics

Table 5-53. InfiniFlash drive card properties – statistics (continued)

Name	Description
`TotalCorrectedReadErrors`	Total corrected write errors Type: Integer Ancestry: Drive
`TotalUncorrectedReadErrors`	Total uncorrected write errors Type: Integer Ancestry: Drive
`OSName`	Drive OS name Type: String Ancestry: Drive

Table 5-54. InfiniFlash drive card properties – getsmart

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Drive's SMART details Type: Drives Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`OSName`	OS name of the device Type: String Ancestry: Drive
`Attribute`	SMART attribute container Type: Container Ancestry: Drive
`Parameter`	SMART parameter name Type: Container Ancestry: Attribute
`CurrentValue`	Current value for the SMART attribute Type: String Ancestry: Attribute

Table 5-54. InfiniFlash drive card properties – getsmart (continued)

Name	Description
`ThresholdValue`	Threshold value for SMART attribute Type: String Ancestry: Attribute
`MaxReported`	Max value reported for a SMART attribute Type: String Ancestry: Attribute
`Detail`	Name of the device Type: String Ancestry: Drive

Table 5-55. InfiniFlash drive card properties - geometry

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Drive's geometry details Type: Drives Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`BlockSize`	Block size of the device Type: Integer Ancestry: Drive
`MaxLBA`	Current Maximum LBA Type: Integer Ancestry: Drive
`Size`	Capacity of the device Type: Integer Ancestry: Statistics
`OSName`	OS Device Name Type: Integer Ancestry: Drive

Table 5-55. InfiniFlash drive card properties - geometry (continued)

Name	Description
`BootDevice`	Boot Device indicator Type: Boolean Ancestry: Drive

Table 5-56. InfiniFlash drive card properties - multipath

Name	Description
`Drives`	Container for Output Type: Container Children: Zero or more Drive
`Drive`	Asset details Type: Container Ancestry: Drives
`DeviceName`	Name of the device Type: String Ancestry: Drive
`OS Multipath names`	Multipaths to the drive Type: String Ancestry: Drive

Format InfiniFlash drive card

NOTE: : If the format command is issued with neither `maxlba` nor `blocksize`, the existing `blocksize` and `blockcount` is retained.

HTTP Method - POST

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/slot[id]/drive?format`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot[id]/drive?format`

Description - This is a destructive operation that formats the InfiniFlash drive card.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl<id>/slot[id]/drive?format / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters – The request parameter `format` is used to format the InfiniFlash drive card.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-57. Format InfiniFlash drive card request elements

Name	Description
`FormatOption`	Container for format option Type: Container
`BlockSize`	Block size If only the `blocksize` option is specified without mentioning the `maxlba` option, the `blockcount` defaults to the maximum LBA supported by the device for the specified `blocksize`. Type: Integer Valid Values: 512 | 4096 Ancestry: `FormatOption`
`maxlba`	Maximum LBA If the value zero is specified, the value defaults to the maximum LBA supported by the device for the existing `blocksize`. Type: Integer Ancestry: `FormatOption`

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-58. Format InfiniFlash drive card response elements

Name	Description
`Jobs`	Container for Output Type: Container Children: Zero or more Job
`Job`	Container for Job details Type: Container Ancestry: Jobs
`URL`	URL to monitor the job created for the operation. In case of failure, this is not present as a response element. Type: URL Ancestry: job
`Percentcomplete`	Percent complete of the job Type: URL Ancestry: Job
`Deletiontime`	Deletion time of the job Type: URL Ancestry: Job
`State`	Current state of the job Type: String Valid Values: "Started" | "Completed" | "In Progress" | "Failed" Ancestry: Job
`Jobtype`	Value: "Format" Type: String Ancestry: Job

Table 5-59. LBA count values for various capacities and overprovision percentages For SDIFC11-2Y04 drives calculated using IDEMA standard

Overprovisioning	Capacity (GB)	LBA Count (512-byte sector format)	LBA Count (4K sector format)
7.00%	3840	7501476528	937684566
14.00%	3648	7126403760	890800470
21.00%	3392	6626306736	828288342
28.00%	3200	6251233968	781404246
35.00%	3072	6001185456	750148182

Table 5-59. LBA count values for various capacities and overprovision percentages For SDIFC11-2Y04 drives calculated using IDEMA standard (continued)

Overprovisioning	Capacity (GB)	LBA Count (512-byte sector format)	LBA Count (4K sector format)
42.00%	2944	5751136944	718892118
49.00%	2752	5376064176	672008022
56.00%	2688	5251039920	656379990
63.00%	2560	5000991408	625123926
70.00%	2432	4750942896	593867862
77.00%	2368	4625918640	578239830
84.00%	2240	4375870128	546983766
91.00%	2176	4250845872	531355734
98.00%	2112	4125821616	515727702

Table 5-60. LBA count values for various capacities and overprovision percentages for SDIFC10-0720801 drives calculated using IDEMA standard

Overprovisioning	Capacity (GB)	LBA Count (512-byte sector format)	LBA Count (4K sector format)
7.00%	7680	15002931888	1875366486
14.00%	7296	14252786352	1781598294
21.00%	6784	13252592304	1656574038
28.00%	6400	12502446768	1562805846
35.00%	6144	12002349744	1500293718
42.00%	5888	11502252720	1437781590
49.00%	5504	10752107184	1344013398
56.00%	5376	10502058672	1312757334
63.00%	5120	10001961648	1250245206
70.00%	4864	9501864624	1187733078
77.00%	4736	9251816112	1156477014
84.00%	4480	8751719088	1093964886
91.00%	4352	8501670576	1062708822
98.00%	4224	8251622064	1031452758

Erase InfiniFlash drive card

HTTP Method - POST

URL - `http://127.0.0.1:8080/sndk/infiniflash/encl/slot<id>/drive?erase`

or

URL - `http://127.0.0.1:8080/sndk/infiniflash/wwn/<logical-id>/slot[id]/drive?erase`

Description - This is a destructive operation that erases the InfiniFlash drive card.

Requests

Syntax - The following are the HTTP header details for REST API:

```
POST /encl<id>/slot[id]/drive?erase / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters – The request parameter `erase` is used to erase the InfiniFlash drive card.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-61. Erase InfiniFlash drive card request elements

Name	Description
`EraseOption`	Container for erase option Type: Container
`Crypto`[1]	Performs crypto erase Type: boolean Valid Values: true Ancestry: EraseOption

1.Crypto Erase allows customers to securely erase a BSSD and put it back into its factory default state prior to RMA'ing the drive. After a Crypto Erase, all security keys get deleted; therefore, user data is destroyed.

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-62. Erase InfiniFlash drive card response elements

Name	Description
`Jobs`	Container for Output Type: Container Children: Zero or more Job
`Job`	Container for Job details Type: Container Ancestry: Jobs
`URL`	URL to monitor the job created for the operation. In case of failure, this is not present as a response element. Type: URL Ancestry: Job
`Percentcomplete`	Percent complete of the job Type: URL Ancestry: Job
`Deletiontime`	Deletion time of the job Type: URL Ancestry: Job
`State`	Current state of the job Type: String Valid Values: "Started" | "Completed" | "In Progress" | "Failed" Ancestry: Job
`Jobtype`	Value: "Erase" Type: String Ancestry: Job

Perform create support bundle

HTTP Method - POST

URL -
`http://<hostname>:[port]/sndk/infiniflash/encl<id>/slot<id>/drive?createSUB`

or

URL -

`http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot<id>/drive?createSUB`

Description – This command extracts necessary information about the drives into an archive file for offline failure analysis or debugging. The data extracted includes crash dump information from the drives.

This operation accepts a request to create support bundle. An operation identifier is returned as part of the response with the response code 201 (Created).

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl0/slot0/drive?createSUB / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization : <token>
```

Request Parameters – The request parameter `createSUB` is used for this operation.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 201 (Created) and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-63. Create support bundle response elements

Name	Description
`Jobs`	Container for operation result Type: Container Children: Zero or more Job
`Job`	Container for sub elements Type: Container Ancestry: Jobs
`url`	URL to monitor the job created for the operation. In case of failure this is not present as a response element. Type: URL Ancestry: Job
`Percentcomplete`	Percent complete of the job Type: URL Ancestry: Job

Table 5-63. Create support bundle response elements (continued)

Name	Description
`Deletiontime`	Deletion time of the job Type: URL Ancestry: Job
`State`	Current state of the job Type: String Valid Values: "Started" | "Completed" | "In Progress" | "Failed" Ancestry: Job
`Filename`	Created support bundle file name Type: URL Ancestry: Job
`Jobtype`	Value: "createSUB" Type: String Ancestry: Job

Perform self-test

HTTP Method - POST

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/slot<id>/drive[id]?selftest`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/slot<id>/drive[id]?selftest`

Description - This command runs a self-test diagnostic on the specified drives. The short test or extended test requires only a few minutes to finish.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /encl<id>/slot<id>/drive?selftest / HTTP/1.1
Host: hostname
Content-Type: application/json | application/xml
Accept: application/json,application/xml
Date: date
Authorization : <token>
```

Request Body – Request body can be XML or JSON, based on that set Content-Type in the header and the body content.

Sample XML

```
<?xml version="1.0" encoding="UTF-8"?>
<typeOption>
  <Type>short | extended</Type>
</typeOption>
```

Sample JSON

```
{
    "typeOption":
        {
            "Type": "short | extended"
        }
}
```

Request Parameters – The request parameter `selftest` is used for this operation.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-64. Perform self-test request elements

Name	Description
`typeOption`	Container for the self-test type Type: Container
`Type`	Type of self-test Type: Enum Valid Values: Short | Extended Ancestry: SelftestRequest

Responses

Response Status code – On successful acceptance of the request, HTTP response status code is returned as 201 (Created) and the operation identifier is returned as part of the response to monitor the long-running operation. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-65. Perform self-test response elements

Name	Description
`Jobs`	Container for operation result Type: Container Children: Job

Table 5-65. Perform self-test response elements (continued)

Name	Description
`Job`	Container for subelements Type: Container Ancestry: Jobs
`url`	URL to monitor the job created for the operation. In case of failure, this is not present as a response element. Type: URL Ancestry: Job
`Percentcomplete`	Percent complete of the job Type: URL Ancestry: Job
`Deletiontime`	Deletion time of the job Type: URL Ancestry: Job
`State`	Current state of the job Type: String Valid Values: "Started" | "Completed" | "In Progress" | "Failed" Ancestry: Job
`Jobtype`	Value: "Self-test" Type: String Ancestry: Job

Connector operations

This section includes RESTful APIs of all connector operations.

Get properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/encl<id>/hse<id>/cn[id]?[state]`

or

URL - `http://<hostname>:[port]/sndk/infiniflash/wwn<logical-id>/hse<id>/cn[id]?[state]`

Description - This operation returns basic information about a particular connector or a list of connectors available in the specified HSE of an enclosure.

Requests

Syntax - Below are the HTTP header details for REST API:

```
GET /encl<id>/hse<id>/cn<id>?[state] / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
```

```
Date: date
Authorization : <token>
```

Request Parameters - The request parameter in this operation applicable for a Particular connector is `state`.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131

Request Elements - This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-66. Connector response elements - all connector slots

Name	Description
`Connectors`	Container for Output Type: Container Children: Zero or more Connector
`Connector`	Container for connector details Type: Container Children: DeviceName, State, Type, Identify Ancestry: Connectors
`DeviceName`	Name of the device. Type: String Ancestry: Connector
`State`	Status of the connector slot Type: String Valid Values: "OK"| "Not Critical" | "Critical" | "Unrecoverable"| "Not Installed"| "Unknown/Busy"| "Not Available"| "Access Not Allowed" | "Unsupported" Ancestry: Connector
`Type`	type of the Connector Type: String Ancestry: Connector

Table 5-66. Connector response elements - all connector slots

Name	Description
`Identify`	State of visual Identifier of the connector Type: Boolean String Valid Values: “On” | “Off” Ancestry: Connector

Locate connector

HTTP Method - POST

URL - `http//<hostname>:[port]/sndk/infiniflash/encl<id>/hse[id]/cn[id]?locate`

Description - This operation identifies the specified SAS connector in the enclosure.

Requests

Syntax - Below are the HTTP header details for REST API:

```
POST /encl<id>/hse<id>/cn[id]?locate / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Body - Request body should be json/xml.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Identifyoption>
<identify>true</identify>
</Identifyoption>
```

JSON

```
{
"Identifyoption":
{
"identify": "true"
}
}
```

Request Parameters - The request parameter used to identify the connector elements is `locate`.

Request Headers - This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-67. Locate connector request elements - all connector slots

Name	Description
`IdentifyOption`	Container for identify connector elements. Type: Container
`Identify`	LED State of connector. Type: Boolean String Valid Values: “True” | “False” Ancestry: IdentifyOption

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-68. Connector response elements - all connector slots

Name	Description
`Connectors`	Container for Output Type: Container Children: Zero or more Connector
`Connector`	Container for connector details Type: Container Children: DeviceName, Identify Ancestry: Connectors
`DeviceName`	Name of the device. Type: String Ancestry: Connector
`Identify`	LED State of Connector Type: Boolean String Valid Values: “True” | “False” Ancestry: Connector

Zone operations

This section includes RESTful APIs of all Zone operations.

Validate zone configuration

HTTP Method - POST

URL - `http://<hostname>:<port>/sndk/infiniflash/zone`

Description - This operation validates if the zone package file is applicable to the enclosure and provides the prebuilt zoning configuration names. The REST client needs to send the zone package in binary multipart format along with the request packet.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /zone / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: multipart/form-data;boundary=infiniflash-upload-body-separator
Date: date
Authorization : <token>
```

Request Body – Request body is HTTP multipart. The first part can be XML or JSON, and the second part is the binary data for the zone package.

Sample multipart for XML option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/xml
<?xml version="1.0" encoding="UTF-8"?>
<zoneparam>
<operation>validate</validate>
<id>encl0</id>
</zoneparam>
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename="test.zpkg"
Content-Type:application/octet-stream
<binary data for the zone package>
--infiniflash-upload-body-separator–
```

Sample multipart for JSON option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/json
<?xml version="1.0" encoding="UTF-8"?>
{
"zoneparam":
{
"operation":"validate",
"id": "encl0",
}
```

```
}
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename="test.zpkg"
Content-Type:application/octet-stream
<binary data for the zone package>
--infiniflash-upload-body-separator–
```

Request Parameters – This operation does not use any request parameters.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131

.

Table 5-69. Validate zone response elements

Name	Description
`zoneinfo`	Container for Output Type: Container Children: Enclosure
`enclosure`	Enclosure details Type: Container Children: id, logicalid, compatible, zoneconfigs Ancestry: zoneinfo
`id`	ID of the enclosure Type: String Ancestry: enclosure
`logicalid`	Logical ID of the enclosure Type: String Ancestry: Enclosure
`compatible`	Compatibility of zone package with enclosure Type: String (Yes | No) Ancestry: Enclosure

Table 5-69. Validate zone response elements (continued)

Name	Description
`zoneconfigs`	Container for zoneconfig Type: Container Children: Zero or more zoneconfig Ancestry: Enclosure
`zoneconfig`	Container for zoneconfig information Type: Container Children: id, Description Ancestry: zoneconfigs
`id`	Zone config Name Type: String Ancestry: zoneconfig
`Description`	Description for the zone config Type: String Ancestry: zoneconfig

Update zone configuration to a zone package specific configuration

HTTP Method - POST

URL - `http://<hostname>:<port>/sndk/infiniflash/zone`

Description - This operation updates zoning of the enclosure to a zone configuration from the zone package. The REST client needs to send the zone package in binary multipart format along with the request packet.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /zone / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: multipart/form-data;boundary=infiniflash-upload-body-separator
Date: date
Authorization : <token>
```

Request Body – Request body is HTTP multipart. The first part can be XML or JSON, and the second part is the binary data for the zone package.

Sample multipart for XML option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/xml
<?xml version="1.0" encoding="UTF-8"?>
<zoneparam>
```

```
<operation>update</operation>
<id>encl0</id>
<name>Z8</name>
</zoneparam>
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename="test.zpkg"
Content-Type:application/octet-stream
<binary data for the zone package>
--infiniflash-upload-body-separator–
```

Sample multipart for JSON option

```
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="parameter"
Content-Type:application/json
<?xml version="1.0" encoding="UTF-8"?>
{
"zoneparam":
{
“operation”:”update”,
"id": "encl0",
"name":"Z8"
}
}
--infiniflash-upload-body-separator
Content-Disposition: form-data; name="upload_file" filename="test.zpkg"
Content-Type:application/octet-stream
<binary data for the zone package>
--infiniflash-upload-body-separator–
```

Request Parameters – This operation does not use any request parameters.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131

.

Table 5-70. Update zone configuration to a zone package specific configuration response elements

Name	Description
`zoneinfo`	Container for Output Type: Container Children: enclosure
`enclosure`	Enclosure details Type: Container Children: `id, Description` Ancestry: `zoneinfo`
`id`	Id of the zone config Type: String Ancestry**:** `enclosure`
`Description`	Description for the zone config Type: String Ancestry**:** `enclosure`

Update zone configuration to default

HTTP Method - POST

URL - `http://<hostname>:<port>/sndk/infiniflash/zone`

Description - This operation updates zoning to default zone setting.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /zone / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters – This operation does not use any request parameters.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-71. Update zone configuration to default request elements

Name	Description
Zoneparam	Container for zone parameters Type: container

Table 5-71. Update zone configuration to default request elements

Name	Description
Operation	Operation to be performed Type: String Valid values: update Ancestry: zoneparam
Id	Enclosure id Type: String Example: encl0 Ancestry: zoneparam
default	Default switch Type: boolean Valid values: true Ancestry: zoneparam

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131

.

Table 5-72. Update zone configuration to default response elements

Name	Description
`zoneinfo`	Container for Output Type: Container Children: enclosure
`enclosure`	Enclosure details Type: Container Children: `id, Description` Ancestry: `zoneinfo`
`id`	Id of the zone config Type: String Ancestry**:** `enclosure`
`Description`	Description for the zone config Type: String Ancestry**:** `enclosure`

Get zone configuration

HTTP Method - POST

URL - `http://<hostname>:<port>/sndk/infiniflash/zone`

Description - This operation provides current zone configuration details of the requested enclosure. If connector detail is passed in the request packet, zoning details of the requested connector are returned, or else the entire zoning details are returned for the specified enclosure.

Requests

Syntax – The following are the HTTP header details for REST API:

```
POST /zone / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Content-Type: application/json (or) application/xml
Date: date
Authorization: <token>
```

Request Parameters – This operation does not use any request parameters.

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Table 5-73. Get zone configuration request elements

Name	Description
Zoneparam	Container for zone parameters Type: container
Operation	Operation to be performed Type: String Valid values: get Ancestry: zoneparam
Id	Enclosure id Type: String Example: encl0 Ancestry: zoneparam
connector	connector Type: String Example: `hse0/cn0` Ancestry: zoneparam

Responses

Response Status code - On successful acceptance of the request, HTTP response status code is returned as 200. On failure, HTTP status code 500 is returned.

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131

.

Table 5-74. Get zone configuration response elements

Name	Description
`zoneinfo`	Container for Output Type: Container Children: `ZoneName, zoneconfig`
`ZoneName`	Name of the zone Type: String Ancestry: `zoneinfo`
`zoneconfig`	Container for `zoneconfig` Type: Container Ancestry: `zoneinfo` Children: `id` , `Connectors`
`id`	Name of the enclosure Type: String Ancestry: zoneconfig
`Connectors`	Container for `Connector` Type: Container Children: `zero or more Connector` Ancestry: zoneconfig
`Connector`	Container for `Connector` information Type: Container Children: `id, groupid` Ancestry: Connectors
`id`	Id of the connector Type: String Ancestry: Connector
`groupid`	`groupid` of the `Connector` Type: String Ancestry: Connector
`DeviceSlots`	Container for slots Type: Container Children: `zero or more Slot` Ancestry: Connector

Table 5-74. Get zone configuration response elements (continued)

Name	Description
`slot`	Container for slot information Type: Container Children: `id, groupid` Ancestry: DeviceSlots
`id`	Id of the slot Type: String Ancestry: slot
`groupid`	`groupid` of the slot Type: String Ancestry: slot

Job operations

This section includes the RESTful APIs of all the Job operations.

Get job properties

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/job_<id>`

Description – This operation returns detailed properties for the specified job identifier.

Requests

Syntax – The following are the HTTP header details:

```
GET /job_1UD099343 / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Headers - This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-75. Job response elements

Name	Description
`Jobs`	Container for Output Type: Container Children: Zero or more Job
`Job`	Container for job details Type: Container Ancestry: Jobs
`URL`	The URL for the affected element Type: String Ancestry: Job

Table 5-75. Job response elements (continued)

Name	Description
`State`	Current state of the job Type: String Valid Values: "Started" | "Completed" | "In Progress" | "Failed" Ancestry: Job
`PercentComplete`	Operation progress info in percentage Type: Integer Ancestry: Job
`Filename`	Link to a resource created on completion of the long-running operation. Example: Create support bundle operation would return a link like /file_<id>. This is only available for createSUB. Type: URL Ancestry: Job
`Type`	Type of the Job; possible values are createSUB | update Type: String Ancestry: Job

File download

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/FILE_{id}`

Description - This operation returns the content of a specified file to the caller.

Requests

Syntax - The following are the HTTP header details:

```
GET /FILE_1UD099343 / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Response Elements – The content of the file will be returned.

Version information

HTTP Method - GET

URL - `http://<hostname>:[port]/sndk/infiniflash/version`

Description – This operation returns the current ifserver version to the caller. The format is xx.xx.xx (major.minor.patch)

Requests

Syntax – The following are the HTTP header details:

```
GET /version / HTTP/1.1
Host: hostname
Accept: application/json, application/xml
Date: date
Authorization: <token>
```

Request Headers – This implementation of the operation uses only request headers that are common to all operations. For more information, see Common request or response headers on page 131.

Request Elements – This implementation of the operation does not use request elements.

Responses

Response Headers – This implementation of the operation uses only response headers that are common to most responses. For more information, see Common request or response headers on page 131.

Table 5-76. Version information response elements

Name	Description
`InfiniFlash`	Container array for version information Type: Container
`ifserver-version`	Version information of ifserver Type: String

6

Common request or response headers

The table that follows describes headers that can be used by various InfiniFlash management REST requests.

Table 6-1. Headers in REST requests

Header Name	Description
`Content-Length`	Length of the message (without the headers) according to RFC 2616. This header is required for PUTs and operations that load XML/JSON/Binary, such as Format/Firmware update.
`Content-Type`	The content type of the resource in case the request content in the body. Example: text/plain | application/octet-stream
`Content-MD5`	The base64 encoded 128-bit MD5 digest of the message (without the headers) according to RFC 1864. This header can be used as a message integrity check to verify that the data is the same data that was originally sent. Although it is optional, we recommend using the Content-MD5 mechanism as an end-to-end integrity check. For more information about REST request authentication, go to Authentication and authorization on page 32.
`Date`	The current date and time according to the requester. Example: Wed, 01 Mar 2006 12:00:00 GMT. When you specify the Authorization header, you must specify the Date header.
`Expect`	When an application uses 100-continue, it does not send the request body until it receives an acknowledgment. If the message is rejected based on the headers, the body of the message is not sent. This header can be used only if it is sending a body with the message. Valid Values: 100-continue
`Host`	For path-style requests, the value is the host URL. For example, ems.sandisk.com This header is required for HTTP 1.1 (most toolkits add this header automatically); optional for HTTP/1.0 requests.

7

Contacting technical support

We offer customer services and support by telephone, email, or through our web portal:

- Telephone: This is the preferred method for escalations and high-severity cases.

  North America toll free: 1.877.816.5740 or 1.855.322.5767

  Telephone numbers for calls originating outside of North America can be found at:

  https://link.sandisk.com/commercialsupport/global-contacts.html

- Email: commercialsupport@sandisk.com

- Web Portal: https://link.sandisk.com/commercialsupport

A

Appendix A: Open source attribution

This product incorporates open source work.

Table A-1. Open Source Attributions

Header Name	Description
Jansson	Copyright (c) 2009-2014 Petri Lehtinen <petri@digip.org> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
pugixml	pugixml 1.4 - an XML processing library Copyright (C) 2006-2014, by Arseny Kapoulkine ( ) Report bugs and download new versions at This is the distribution of pugixml, which is a C++ XML processing library that consists of a DOM-like interface with rich traversal/modification capabilities, an extremely fast XML parser, which constructs the DOM tree from an XML file/buffer, and an XPath 1.0 implementation for complex data-driven tree queries. Full Unicode support is also available, with Unicode interface variants and conversions between different Unicode encodings (which happen automatically during parsing/saving). The distribution contains the following folders: contrib/ - various contributions to pugixml docs/ - documentation docs/samples - pugixml usage examples docs/quickstart.html - quick start guide docs/manual.html - complete manual scripts/ - project files for IDE/build systems src/ - header and source files readme.txt - this file. This library is distributed under the MIT License: Copyright (c) 2006-2014 Arseny Kapoulkine Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
Zlib	/* zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013 Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler This software is provided "AS-IS," without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution. Jean-loup Gailly Mark Adler jloup@gzip.org madler@alumni.caltech.edu */

Table A-1. Open Source Attributions

Header Name	Description
Civetweb	### Included with all features. Copyright (c) 2004-2013 Sergey Lyubka Copyright (c) 2013 No Face Press, LLC (Thomas Davis) Copyright (c) 2013 F-Secure Corporation Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
	Lua License ------ ### Included only if built with Lua support. http://www.lua.org/license.html Copyright (c) 1994-2013 Lua.org, PUC-Rio. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. SQLite3 License ------ ### Included only if built with Lua support. http://www.sqlite.org/copyright.html 2001 September 15

Table A-1. Open Source Attributions

Header Name	Description
	The author disclaims copyright to this source code. In place of a legal notice, here is a blessing: May you do good and not evil. May you find forgiveness for yourself and forgive others. May you share freely, never taking more than you give. lsqlite3 License ------ ### Included only if built with Lua support. lsqlite3 Copyright (C) 2002-2007 Tiago Dionizio, Doug Currie All rights reserved. Author : Tiago Dionizio <tiago.dionizio@ist.utl.pt> Author : Doug Currie <doug.currie@alum.mit.edu> Library : lsqlite3 - a SQLite 3 database binding for Lua 5 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
	Lua File System License ------ ### Included only if built with Lua support. http://keplerproject.github.io/luafilesystem/license.html Copyright © 2003 Kepler Project. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
	LuaXML License ------ ### Included only if built with Lua support. LuaXml is licensed under the terms of the MIT license reproduced below, the same as Lua itself. This means that LuaXml is free software and can be used for both academic and commercial purposes at absolutely no cost. Copyright (C) 2007-2013 Gerald Franz, eludi.net Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Table A-1. Open Source Attributions

Header Name	Description
	OpenSSL License Copyright (c) 1998-2016 The OpenSSL Project. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this * software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)" 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. 5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. 6. Redistributions of any form whatsoever must retain the following * acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)" THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).This product includes software written by Tim * Hudson (tjh@cryptsoft.com).

Table A-1. Open Source Attributions

Header Name	Description
	Original SSLeay License Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved. This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL. This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com). Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)" The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-). 4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

Table A-1. Open Source Attributions

Header Name	Description
	THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]