SolidFire Documentation

Release 1.6.0.126

Author

Mar 18, 2020

Contents

Contents:

CHAPTER 1

SolidFire Python SDK

Python SDK library for interacting with SolidFire Element API

1.1 Current Release

Version 1.6.0.126

1.2 Description

The SolidFire Python SDK is a collection of libraries that facilitate integration and orchestration between proprietary systems and third-party applications. The Python SDK allows developers to deeply integrate SolidFire system API with the Python programming language. The SolidFire Python SDK reduces the amount of additional coding time required for integration.

1.3 Compatibility

Component	Version
SolidFire Element OS	7.0 - 10.0

1.4 Getting Help

If you have any questions or comments about this product, contact ng-sf-host-integrations-sdk@netapp.com or reach out to the online developer community at ThePub. Your feedback helps us focus our efforts on new features and capabilities.

1.5 Documentation

Latest Docs

Release Notes

1.6 Installation

From PyPI

```
pip install solidfire-sdk-python
```

From Source

Note: It is recommended using virtualenv for isolating the python environment to only the required libraries.

Alternatively, for development purposes or to inspect the source, the following will work:

```
git clone git@github.com:solidfire/solidfire-sdk-python.git
cd solidfire-sdk-python
git checkout develop
pip install -e ".[dev, test, docs, release]"
python setup.py install
```

Then append the location of this directory to the `PYTHONPATH` environment variable to use the SDK in other python scripts:

```
export PYTHONPATH=$PYTHONPATH:/path/to/sf-python-sdk/
```

That's it – you are ready to start interacting with your SolidFire cluster using Python!

1.7 Examples

1.7.1 Step 1 - Build an Element object using the factory

This is the preferred way to construct the Element object. The factory will make a call to the SolidFire cluster using the credentials supplied to test the connection. It will also set the version to communicate with based on the highest number supported by the SDK and Element OS. Optionally, you can choose to set the version manually and whether or not to verify SSL. Read more about it in the ElementFactory documentation.

```
from solidfire.factory import ElementFactory

# Use ElementFactory to get a SolidFireElement object.
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")
```

1.7.2 Step 2 - Call the API method and retrieve the result

All service methods in SolidFireElement call API endpoints and they all return result objects. The naming convention is `[method_name]_result`. For example, `list_accounts` returns a `list_accounts_result` object which has a property called `accounts` that can be iterated.

This example sends a request to list accounts then pulls the first account from the `add_account_result` object.

```
# Send the request and wait for the result then pull the AccountID
list_accounts_result = sfe.list_accounts()
account = list_accounts_result.accounts[0];
```

1.7.3 More examples using the Python SDK

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# --------- EXAMPLE 1 - CREATE AN ACCOUNT -----------
# Send the request with required parameters and gather the result
add_account_result = sfe.add_account(username="example-account")
# Pull the account ID from the result object
account_id = add_account_result.account_id

# --------- EXAMPLE 2 - CREATE A VOLUME -------------
# Send the request with required parameters and gather the result
create_volume_result = sfe.create_volume(name="example-volume",
                                         account_id=account_id,
                                         total_size=1000000000,
                                         enable512e=False)
# Pull the VolumeID off the result object
volume_id = create_volume_result.volume_id

# --------- EXAMPLE 3 - LIST ONE VOLUME FOR AN ACCOUNT -------------
# Send the request with desired parameters and pull the first volume in the
# result
volume = sfe.list_volumes(accounts=[account_id], limit=1).volumes[0]
# pull the iqn from the volume
iqn = volume.iqn

# --------- EXAMPLE 3 - MODIFY A VOLUME -------------
# Send the request with the desired parameters
sfe.modify_volume(volume_id=volume_id, total_size=2000000000)
```

1.8 More Examples

More specific examples are available here

1.9 Logging

To configure logging responses, execute the following:

```
import logging
from solidfire import common
common.setLogLevel(logging.DEBUG)
```

To access the logger for the Element instance:

```
from solidfire.common import LOG
```

1.10 Timeouts

Connection timeout (useful for failing fast when a host becomes unreachable):

```
from solidfire.factory import ElementFactory
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")
sfe.timeout(600)
```

Read timeout (useful for extending time for a service call to return):

```
from solidfire.factory import ElementFactory
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")
sf.read_timeout(600)
```

1.11 License

CHAPTER 2

Account Management Examples

2.1 Manage Accounts

These examples walk through all interactions with an Account on the SolidFire cluster.

Examples for:

- *List all Accounts*
- *Get one Account*
- *Create an Account*
- *Modify an Account*

2.1.1 Documentation

Further documentation for each method and type can be found in our PyPI documentation repository.

2.1.2 List all Accounts

To list all accounts on a cluster:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request and gather the result
list_accounts_result = sfe.list_accounts()

# iterate the accounts array on the result object and display each Account
for account in list_accounts_result.accounts:
    print(account)
```

2.1.3 Get one Account

To get a single account by ID:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request with a specific account id and gather the result
get_account_result = sfe.get_account_by_id(1)

# Display the account from the result object
print(get_account_result.account)
```

To get a single account by username:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request with a specific account username and gather the result
get_account_result = sfe.get_account_by_name('username-of-account')

# Display the account from the result object
print(get_account_result.account)
```

2.1.4 Create an Account

To create an account you must specify the `username`. Optionally, you can also specify the `initiator_secret` and `target_secret` which are **CHAPSecret** objects. If those secrets are not specified, they will be auto-generated.

First, we create an account with only a username:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request and gather the result
add_account_result = sfe.add_account(username="my-new-account")

# Grab the account ID from the result object
new_account_id = add_account_result.account_id
```

Now we create an account and specify the `username` and `initiator_secret`. Notice we created a new **CHAPSecret** object and set the string value for the `intitiator_secret`. The `target_secret` will be auto-generated during the process on the cluster:

```
from solidfire.factory import ElementFactory
from solidfire import CHAPSecret

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

```

(continues on next page)

(continued from previous page)

```
# Send the request and gather the result
add_account_result = sfe.add_account(username="my-new-account",
                                     initiator_secret=CHAPSecret(
                                         "a12To16CharValue"))
# The initiator and target secrets can be set many different ways
# Below are more examples
# Passing strings into add account.
add_account_result = sfe.add_account("my-new-account", "a12To16CharValue")
# Passing string into CHAPSecret() as a parameter
add_account_result = sfe.add_account("my-new-account", CHAPSecret("a12To16CharValue"))
# Explicitly setting 'secret' in CHAPSecret()
add_account_result = sfe.add_account("my-new-account", CHAPSecret(secret=
↪"a12To16CharValue"))
# Creating a kwarg for secret and passing it in
kwarg = {"secret":"a12To16CharValue"}
add_account_result = sfe.add_account("my-new-account", CHAPSecret(**kwarg))

# Grab the account ID from the result object
new_account_id = add_account_result.account_id
```

2.1.5 Modify an Account

To modify an account, all you need is the `account_id` and the values you want to change. Any values you leave off will remain as they were before this call is made.

In this example, we will instruct the API to autogenerate a new `target_secret` value for an account. In order to do so we need to call the static `auto_generate()` method on the **CHAPSecret** class.

```
from solidfire.factory import ElementFactory
from solidfire import CHAPSecret

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request with the account_id and gather the result
add_account_result = sfe.modify_account(account_id=1,
                                        target_secret=CHAPSecret.auto_generate())
```

CHAPTER 3

Snapshot Scheduling Examples

3.1 Snapshot Scheduling

These examples walk through all interactions with a Schedule. Schedules control when automatic Snapshots will be taken of volumes on the SolidFire cluster.

Examples for:

- *List all Schedules*
- *Get one Schedule*
- *Create a Schedule*
- *Modify a Schedule*

3.1.1 Documentation

Further documentation for each method and type can be found in our PyPI documentation repository.

3.1.2 List all Schedules

To list all the schedules on a cluster:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request and gather the result
list_schedules_result = sfe.list_schedules()

# iterate the schedules array on the result object and display each Schedule
```

(continues on next page)

(continued from previous page)

```
for schedule in list_schedules_result.schedules:
    print(schedule)
```

3.1.3 Get one Schedule

To get a single schedule:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request with the schedule_id and gather the result
get_schedule_result = sfe.get_schedule(schedule_id=56)

# Display the schedule from the result object
print(get_schedule_result.schedule)
```

3.1.4 Create a Schedule

In order for automatic snapshots to be taken, you need to create a schedule. There are three types of schedules that can be created:

- *Time Interval*
- *Days Of Week*
- *Days Of Month*

All three types of schedules are demonstrated here:

3.1.4.1 Time Interval Schedule

This type of schedule will base snapshots on a time interval frequency. Each snapshot will be taken after the specified amount of time has passed. Control the duration by setting days, hours, and minutes on the TimeIntervalFrequency object.

```
from solidfire.custom.models import TimeIntervalFrequency
from solidfire.models import Schedule

sched = Schedule()
sched.name = "SnapshotEvery3AndAHalfDays"
sched.frequency = TimeIntervalFrequency(days=3, hours=12)
```

3.1.4.2 Days Of Week Schedule

This type of schedule will base snapshots on a weekly frequency. Each snapshot will be taken on the specified weekdays at the time specified in the hours and minutes properties. Control the schedule by setting weekdays, hours, and minutes on the DaysOfWeekFrequency object.

```
from solidfire.custom.models import DaysOfWeekFrequency, Weekday
from solidfire.models import Schedule

sched = Schedule()
sched.name = "SnapshotOnMonWedFriAt3am"
sched.frequency = DaysOfWeekFrequency(
            weekdays=[
                Weekday.from_name("Monday"),
                Weekday.from_name("Wednesday"),
                Weekday.from_name("Friday")],
            hours=3)
```

3.1.4.3 Days Of Month Schedule

This type of schedule will base snapshots on a monthly frequency. Each snapshot will be taken on the specified month days at the time specified in the hours and minutes properties. Control the schedule by setting monthdays, hours, and minutes on the DaysOfMonthFrequency object.

```
from solidfire.custom.models import DaysOfMonthFrequency
from solidfire.models import Schedule

sched = Schedule()
sched.name = "SnapshotOn7th14thAnd21stAt0130Hours"
sched.frequency = DaysOfMonthFrequency(
            monthdays=[7,14,21],
            hours=3,
            monutes=30)
```

3.1.4.4 Create a Schedule (cont.)

After creating the schedule and setting the frequency to Time Interval, Days Of Week, or Days Of Month, complete the object by setting the schedule_info property. This controls information about the resulting snapshot such as which volumes are in it, its name, and how long it should be retained.

Continuing on with the *Time Interval* example from above:

```
from solidfire.custom.models import TimeIntervalFrequency
from solidfire.models import Schedule, ScheduleInfo
from solidfire.factory import ElementFactory

sched = Schedule()
sched.name = "SnapshotEvery12Hours"
sched.frequency = TimeIntervalFrequency(hours=12)
sched.schedule_info = ScheduleInfo(
    volume_ids = [1, 3, 5],
    snapshot_name = '12th hour snapshot',
    retention="72:00:00" # in HH:mm:ss format
)
# When should the schedule start?
sched.starting_date = "2016-12-01T00:00:00Z"

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")
```

(continues on next page)

(continued from previous page)

```
# Call the create_schedule method with the newly created schedule object
create_schedule_result = sfe.create_schedule(sched)

# Grab the schedule ID from the result object
new_schedule_id = create_schedule_result.schedule_id
```

At this point we have created a new schedule called SnapshotEvery12Hours that creates a snapshot whose name is prepended with "12th hour snapshot" every 12 hours for volumes 1, 3, and 5 that is retained for 72 hours.

3.1.5 Modify a Schedule

To modify a schedule, first you must have a valid schedule object with its schedule_id set. You can create one manually but it is preferred to retrieve it from the cluster, modify the properties needed and then send it back. Here is an example:

```
from solidfire.factory import ElementFactory

# Create connection to SF Cluster
sfe = ElementFactory.create("ip-address-of-cluster", "username", "password")

# Send the request with the schedule_id and gather the result
get_schedule_result = sfe.get_schedule(schedule_id=new_schedule_id)

# set a schedule variable from the schedule in the result for ease of use
sched = get_schedule_result.schedule

# display the retrieved schedule
print(sched)

# set paused to True in order to pause the schedule
sched.paused = True

# send the request to modify this schedule
sfe.modify_schedule(sched)

# Send another get_schedule request and gather the result
get_modified_schedule_result = sfe.get_schedule(schedule_id=new_schedule_id)

# display the newly modified schedule
print(get_modified_schedule_result.schedule)
```

This is the output:

```
Schedule(frequency=TimeIntervalFrequency(days=0, hours=12, minutes=0), has_
↪error=False, last_run_status='Success', last_run_time_start=None, name=
↪'SnapshotsEvery12Hours', paused=False, recurring=False, run_next_interval=False,␣
↪schedule_id=56, schedule_info=ScheduleInfo(enable_remote_replication=None,␣
↪retention='72:00:00', snapshot_name='12th hour snapshot', volume_ids='[1, 3, 5]'),␣
↪starting_date='2016-12-01T00:00:00Z', to_be_deleted=False)

Schedule(frequency=TimeIntervalFrequency(days=0, hours=12, minutes=0), has_
↪error=False, last_run_status='Success', last_run_time_start=None, name=
↪'SnapshotsEvery12Hours', paused=True, recurring=False, run_next_interval=False,␣
↪schedule_id=56, schedule_info=ScheduleInfo(enable_remote_replication=None,␣
↪retention='72:00:00', snapshot_name='12th hour snapshot', volume_ids='[1, 3, 5]'),␣
↪starting_date='2016-12-01T00:00:00Z', to_be_deleted=False)
```

Notice the *paused* field changes from `False` to `True`

CHAPTER 4

solidfire package

4.1 Subpackages

4.1.1 solidfire.adaptor package

4.1.1.1 Submodules

4.1.1.2 solidfire.adaptor.schedule_adaptor module

Module implements Schedule Simplification conversion logic.

class solidfire.adaptor.schedule_adaptor.**ScheduleAdaptor**

This class contains the implementation of the schedule specific adaptor calls for simplifying Snapshot Scheduling.

static create_schedule(*element*, *params*, *since*, *deprecated*)

Calls to this static method should ONLY originate from the create_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static get_schedule(*element*, *params*, *since*, *deprecated*)

Calls to this static method should ONLY originate from the get_schedule method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static list_schedules(*element*, *params*, *since*, *deprecated*)

Calls to this static method should ONLY originate from the list_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static modify_schedule(*element*, *params*, *since*, *deprecated*)

Calls to this static method should ONLY originate from the modify_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static to_api_schedule(*schedule*)

Converts a Schedule object into an ApiSchedule object :param schedule: the Schedule object :type schedule: Schedule

Returns solidfire.apiactual.ApiSchedule

static to_api_schedule_info(*info*)
Converts a ScheduleInfo object into an ApiScheduleInfo object

Parameters **info** (ScheduleInfo) – the ScheduleInfo object

Returns solidfire.apiactual.ApiScheduleInfo

static to_schedule(*api*)
Converts an ApiSchedule object into a Schedule object

Parameters **api** (solidfire.apiactual.ApiSchedule) – the ApiSchedule object to be converted

Returns solidfire.models.Schedule

static to_schedule_info(*api*)
Convert an ApiScheduleInfo object into a ScheduleInfo object

Parameters **api** (solidfire.apiactual.ApiScheduleInfo) – the ApiScheduleInfo object

Returns solidfire.models.ScheduleInfo

static to_weekdays(*api*)
Converts an ApiWeekday object array into a Weekday object array

Parameters **api** (solidfire.apiactual.ApiWeekday[]) – array of ApiWeekday objects

Returns solidfire.custom.models.Weekday[]

4.1.1.3 Module contents

Module implements the generated adaptor calls from solidfire.Element

class solidfire.adaptor.**ElementServiceAdaptor**
Bases: object

This class contains the implementation of the generated adaptor calls from solidfire.Element.

static create_schedule(*element*, *params*, *since*, *deprecated*)
Calls to this static method should ONLY originate from the create_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static get_node_stats(*element*, *params*, *since*, *deprecated*)
This adaptor includes the original Node ID from the request in the response object. It is returned as null from the original API call.

Parameters

- **element** (Element) – an instance of Element
- **params** (*dict*) – the parameters supplied to the get_node_stats call
- **since** (*float or str or None*) – service method inception version
- **deprecated** (*float or str or None*) – service method deprecation version

Returns a response

Return type *GetNodeStatsResult*

static get_schedule(*element*, *params*, *since*, *deprecated*)
Calls to this static method should ONLY originate from the get_schedule method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static invoke_sfapi(*element*, *params*, *since*, *deprecated*)

static list_schedules(*element*, *params*, *since*, *deprecated*)
Calls to this static method should ONLY originate from the list_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

static modify_schedule(*element*, *params*, *since*, *deprecated*)
Calls to this static method should ONLY originate from the modify_schedules method in the Element class. DO NOT CALL THIS directly. Documentation here is intentionally brief.

4.1.2 solidfire.apiactual package

4.1.2.1 Module contents

Module contains objects directly mapped to the Element API. These are generated and then moved here before the API is regenerated with changes. Adaptors are used to transform actual API object into custom object and vice-versa.

class solidfire.apiactual.**ApiGetScheduleResult**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

The object returned by the "get_schedule" API Service call.

Parameters **schedule** (Schedule) – [required] The schedule attributes.

schedule = <class 'solidfire.apiactual.ApiSchedule'>

class solidfire.apiactual.**ApiListSchedulesResult**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

The object returned by the "list_schedules" API Service call.

Parameters **schedules** (Schedule) – [required] The list of schedules currently on the cluster.

schedules = <class 'solidfire.apiactual.ApiSchedule[]'>

class solidfire.apiactual.**ApiModifyScheduleResult**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

The object returned by the "modify_schedule" API Service call.

schedule = <class 'solidfire.apiactual.ApiSchedule'>

class solidfire.apiactual.**ApiSchedule**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

Schedule is an object containing information about each schedule created to autonomously make a snapshot of a volume. The return object includes information for all schedules. If *schedule_id* is used to identify a specific schedule then only information for that *schedule_id* is returned. Schedules information is returned with the API method, see *list_schedules* on the SolidFire API guide page 245.

Parameters

- **attributes** (*dict*) – [required] Indicates the frequency of the schedule occurrence.

  Valid values are:

  Day of Week

  Day of Month

Time Interval

- **has_error** (*bool*) – Indicates whether or not the schedule has errors.
- **hours** (*int*) – [required] Shows the hours that will elapse before the next snapshot is created.

  Valid values are: 0 - 24
- **last_run_status** (*str*) – Indicates the status of the last scheduled snapshot.

  Valid values are:

  Success

  Failed
- **last_run_time_started** (*str*) – Indicates the last time the schedule started n ISO 8601 date string. Valid values are:

  Success

  Failed
- **minutes** (*int*) – [required] Shows the minutes that will elapse before the next snapshot is created. Valid values are: 0 - 59
- **monthdays** (*int*) – Shows the days of the month that the next snapshot will be created on. Valid values are: 0 - 31
- **paused** (*bool*) – Indicates whether or not the schedule is paused.
- **recurring** (*bool*) – Indicates whether or not the schedule is recurring.
- **run_next_interval** (*bool*) – Indicates whether or not the schedule will run the next time the scheduler is active. When set to "true", the schedule will run the next time the scheduler is active and then reset back to "false".
- **schedule_id** (*int*) – Unique ID of the schedule
- **schedule_info** (ScheduleInfo) – [required] Includes the unique name given to the schedule, the retention period for the snapshot that was created, and the volume ID of the volume from which the snapshot was created.
- **schedule_name** (*str*) – Unique name assigned to the schedule.
- **schedule_type** (*str*) – [required] Only "snapshot" is supported at this time.
- **starting_date** (*str*) – Indicates the date the first time the schedule began of will begin. Formatted in UTC time.
- **to_be_deleted** (*bool*) – Indicates if the schedule is marked for deletion.
- **weekdays** (*Weekday*) – Indicates the days of the week that a snapshot will be made.

attributes = <type 'dict'>

has_error = <type 'bool'>

hours = <type 'int'>

last_run_status = <type 'str'>

last_run_time_started = <type 'str'>

minutes = <type 'int'>

monthdays = <type 'int[]'>

paused = <type 'bool'>

recurring = <type 'bool'>

run_next_interval = <type 'bool'>

schedule_id = <type 'int'>

schedule_info = <class 'solidfire.apiactual.ApiScheduleInfo'>

schedule_name = <type 'str'>

schedule_type = <type 'str'>

starting_date = <type 'str'>

to_be_deleted = <type 'bool'>

to_json()
Converts DataObject to json.

Returns the DataObject as a json structure.

weekdays = <class 'solidfire.apiactual.ApiWeekday[]'>

class solidfire.apiactual.**ApiScheduleInfo**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

This represents the ScheduleInfo object in the ApiSchedule class. It should not be used by users of this SDK. The ScheduleAdaptor will convert between ApiScheduleInfo and ScheduleInfo during calls. Refer to documentation about Schedule, ScheduleInfo, and Weekday for more information.

Parameters

- **volume_id** (*int*) – (optional) The ID of the volume to be included in the snapshot.
- **volumes** (*int*) – (required) A list of volume *ids* to be included in the group snapshot.
- **name** (*str*) – (optional) The snapshot name to be used.
- **enable_remote_replication** (*bool*) – (optional) Indicates if the snapshot should be included in remote replication.
- **retention** (*str*) – (optional) The amount of time the snapshot will be retained in HH:mm:ss.

enable_remote_replication = <type 'bool'>

name = <type 'str'>

retention = <type 'str'>

volume_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.apiactual.**ApiWeekday**(***kwargs*)
Bases: *solidfire.common.model.DataObject*

This represents the Weekday object used by the API when an ApiSchedule is setup for a Days Of Week frequency. It should not be used by users of this SDK. The ScheduleAdaptor will convert between ApiWeekday and Weekday during calls. Refer to documentation about Schedule, ScheduleInfo, and Weekday for more information.

Parameters

- **day** (*int*) – [required]

- **offset** (*int*) – [required]

day = <type 'int'>

offset = <type 'int'>

4.1.3 solidfire.common package

4.1.3.1 Submodules

4.1.3.2 solidfire.common.model module

class solidfire.common.model.**DataObject**(***kwargs*)
Bases: *solidfire.common.model.ModelProperty*

DataObject is the base type for all generated types, including the MetaData properties, as described from the api descriptors.

classmethod extract(*data*, *strict=True*)
Converts json to a DataObject.

Parameters

- **data** (*str*) – json data to be deserialized back to a DataObject
- **strict** (*bool*) – If True, missing values will raise an error, otherwise, missing values will None or empty.

Returns a class deserialized from the data provided.

get_properties()
Exposes the type properties for a Data Object.

Returns the dictionary of property names and thier type information.

Return type dict

to_json()
Converts DataObject to json.

Returns the DataObject as a json structure.

class solidfire.common.model.**MetaDataObject**(*name*, *bases*, *classdict*)
Bases: type

MetaDataObject defines a method for attributing ModelProperties to a type.

class solidfire.common.model.**ModelProperty**(*member_name*, *member_type*, *array=False*, *optional=False*, *documentation=None*, *dictionaryType=None*)
Bases: object

ModelProperty metadata container for API data type information.

ModelProperty constructor.

Parameters

- **member_name** (*str*) – the name of the property.
- **member_type** (*str*) – the type of the property.
- **array** (*bool*) – is the property an array.
- **optional** (*bool*) – is the property optional.

- **documentation** (*str*) – documentation for the property.

array()

Returns is the property an array

documentation()

Returns the property documentation

extend_json(*out*, *data*)
Serialize the property as json-like structure.

Parameters

- **out** – the resulting output.
- **data** – the data to be converted.

extract_from(*data*)
Deserialize the property from json.

Parameters **data** – the data to be converted.

Returns the extracted data.

known_default()
Helps convert a property to a default value.

Returns a known default for a type.

member_name()

Returns the member name.

member_type()

Returns the member type.

optional()

Returns is the property optional

solidfire.common.model.**extract**(*typ*, *src*)
DataObject value type converter.

Parameters

- **typ** – the type to extract.
- **src** – the source to extract as type typ.

Returns if the type has the ability to extract (convert), otherwise the original version is returned.

solidfire.common.model.**property**(*member_name*, *member_type*, *array=False*, *optional=False*, *documentation=None*, *dictionaryType=None*)
Constructs the type for a DataObject property.

Parameters

- **member_name** (*str*) – the name of the property.
- **member_type** (*type*) – the type of the property.
- **array** (*bool*) – is the property an array.
- **optional** (*bool*) – is the property optional.
- **documentation** (*str or NoneType*) – documentation for the property.

Returns the constructed type of a property

solidfire.common.model.**serialize**(*val*)
DataObject serializer value based on MetaData attributes.

Parameters **val** – any value

Returns the serialized value

4.1.3.3 Module contents

API Common Library

exception solidfire.common.**ApiConnectionError**(*message*)
Bases: exceptions.Exception

exception solidfire.common.**ApiMethodVersionError**(*method_name*, *api_version*, *since*, *deprecated=None*)
Bases: exceptions.Exception

An ApiMethodVersionError occurs when a service method is not compatible with the version of the connected server.

ApiMethodVersionError constructor.

Parameters

- **method_name** (*str*) – name of the service method where the error occurred.
- **api_version** (*str or float*) – the version of API used to instantiate the connection to the server.
- **since** (*str or float*) – the earliest version of the API a service method is compatible.
- **deprecated** (*str or float*) – the latest version of the API that a method is compatible.

api_version
The version of the Element API Service

deprecated
The version a service was deprecated

method_name
The name of the service method causing the error.

since
The version a service was introduced

exception solidfire.common.**ApiParameterVersionError**(*method_name*, *api_version*, *params*)
Bases: exceptions.Exception

An ApiParameterVersionError occurs when a parameter supplied to a service method is not compatible with the version of the connected server.

ApiParameterVersionError constructor.

Parameters

- **method_name** (*str*) – name of the service method where the error occurred.
- **api_version** (*str or float*) – the version of API used to instantiate the connection to the server.

- **params** (*list of tuple*) – the list of incompatible parameters provided to a service method call. This tuple should include name, value, since, and deprecated values for each offending parameter.

api_version
The version of the Element API Service

method_name
The name of the service method causing the error.

params
The parameters checked with a service call

violations
The parameters violated with the service call

exception solidfire.common.**ApiServerError**(*method_name*, *err_json*)
Bases: exceptions.Exception

ApiServerError is an exception that occurs on the server and is passes as a response back to the sdk.

ApiServerError constructor.

Parameters

- **method_name** (*str*) – name of the service method where the error occurred.
- **err_json** (*str*) – the json formatted error received from the service.

error_code
The numeric code for this error.

error_name
The name of the error.

message
A user-friendly message returned from the server.

method_name
The name of the service method causing the error.

exception solidfire.common.**ApiVersionExceededError**(*api_version*, *current_version*)
Bases: exceptions.Exception

An ApiVersionExceededError occurs when connecting to a server with a version lower then the provided api_version.

ApiVersionExceededError constructor.

Parameters

- **api_version** (*str or float*) – the version of API used to instantiate the connection to the server.
- **current_version** (*float*) – the current version of the server.

api_version
The version of the Element API Service

current_version
The current version of the connected Element OS

exception solidfire.common.**ApiVersionUnsupportedError**(*api_version*, *supported_versions*)
Bases: exceptions.Exception

An ApiVersionUnsupportedError occurs when connecting to a server unable to support the provided api_version.

ApiVersionUnsupportedError constructor.

Parameters

- **api_version** (*str or float*) – the version of API used to instantiate the connection to the server.
- **supported_versions** (*float[]*) – the list of supported versions provided by a server.

api_version
The version of the Element API Service

supported_versions
The versions supported by the connected Element OS

class solidfire.common.**CurlDispatcher**(*endpoint*, *username*, *password*, *verify_ssl*)
Bases: object

The CurlDispatcher is responsible for connecting, sending, and receiving data to a server.

The CurlDispatcher constructor.

Parameters

- **endpoint** (*str*) – the server URL
- **username** (*str*) – the username for authentication
- **password** (*str*) – the password for authentication
- **verify_ssl** (*bool*) – If True, ssl errors will cause an exception to be raised, otherwise, if False, they are ignored.

connect_timeout(*timeout_in_sec*)
Set the time to wait for a connection to be established before timeout.

Parameters **timeout_in_sec** (*int*) – the connection timeout in seconds.

Raises **ValueError** – if timeout_in_sec is less than 0

post(*data*)
Post data to the associated endpoint and await the server's response.

Parameters **data** (*str or json*) – the data to be posted.

restore_timeout_defaults()
Restores the Connection and Read Timeout to their original durations of 30 seconds for connection timeout and 300 seconds (5 minutes) for read timeout.

timeout(*timeout_in_sec*)
Set the time to wait for a response before timeout.

Parameters **timeout_in_sec** (*int*) – the read timeout in seconds.

Raises **ValueError** – if timeout_in_sec is less than 0

exception solidfire.common.**SdkOperationError**(**args*, ***kwargs*)
Bases: exceptions.Exception

class solidfire.common.**ServiceBase**(*mvip=None*, *username=None*, *password=None*, *api_version=8.0*, *verify_ssl=True*, *dispatcher=None*)
Bases: object

The base type for API services. This performs the sending, encoding and decoding of requests.

Constructor for initializing a connection to an instance of Element OS

Parameters

- **mvip** (*str*) – the management IP (IP or hostname)
- **username** (*str*) – username use to connect to the Element OS instance.
- **password** (*str*) – authentication for username
- **api_version** (*float*) – specific version of Element OS to connect.
- **verify_ssl** (*bool*) – disable to avoid ssl connection errors especially when using an IP instead of a hostname
- **dispatcher** – a prebuilt or custom http dispatcher

Returns a configured connection to an Element OS instance

api_version
Returns the version of the Element API

Returns the version of the Element API

Return type float

connect_timeout(*timeout_in_sec*)
Set the time to wait for a connection to be established before timeout.

Parameters **timeout_in_sec** (*int*) – the connection timeout in seconds.

Raises **ValueError** – if timeout_in_sec is less than 0

restore_timeout_defaults()
Restores the Connection and Read Timeout to their original durations of 300 seconds (5 minutes) each.

send_request(*method_name*, *result_type*, *params=None*, *since=None*, *deprecated=None*, *return_response_raw=False*)

Parameters

- **method_name** (*str*) – the name of the API method to call
- **result_type** (DataObject) – the type of the result object returned from the API method called.
- **params** (*dict*) – the parameters supplied to the API call.
- **since** (*str or float*) – the first version this service was available
- **deprecated** (*str or float*) – the final version this service was available

Returns the result of the API service call

Return type *DataObject*

timeout(*timeout_in_sec*)
Set the time to wait for a response before timeout.

Parameters **timeout_in_sec** (*int*) – the read timeout in seconds.

Raises **ValueError** – if timeout_in_sec is less than 0

solidfire.common.**setLogLevel**(*level*)
Set the logging level of Element logger and all handlers.

```
>>> from logging
>>> from solidfire import common
>>> common.setLogLevel(logging.DEBUG)
```

Parameters **level** – level must be an int or a str.

4.1.4 solidfire.custom package

4.1.4.1 Module contents

Module contains user defined objects directly implemented to map to the Element API.

4.1.5 solidfire.util package

4.1.5.1 Module contents

API Utilities

solidfire.util.**ascii_art**(*version*)
Used to build SolidFire ASCII art.

Returns a string with the SolidFire ASCII art.

4.2 Submodules

4.3 solidfire.factory module

class solidfire.factory.**ElementFactory**
The Factory for creating a SolidFire Element object.

static create(*target*, *username*, *password*, *version=None*, *verify_ssl=False*, *port=443*, *print_ascii_art=True*, *timeout=30*)

Factory method to create a Element object which is used to call the SolidFire API. This method runs multiple checks and logic to ensure the Element object creation is valid for the cluster you are attempting to connect to. It is preferred to use this factory method over the standard constructor.

Parameters

- **target** (*str*) – the target IP or hostname of the cluster or node.
- **username** (*str*) – username used to connect to the Element OS instance.
- **password** (*str*) – authentication for username
- **version** (*float or str*) – specific version of Element OS to connect to. If this doesn't match the cluster or is outside the versions supported by this SDK, you will get an exception.
- **verify_ssl** (*bool*) – enable this to check ssl connection for errors especially when using a hostname. It is invalid to set this to true when using an IP address in the target.
- **port** (*int*) – a port to connect to if other than 443, which is the default for a SolidFire cluster. Specify 442 if connecting to a SoldiFire node.
- **print_ascii_art** (*bool*) – When True, print the SolidFire Robot to the log. Production deployments might consider disabling this feature.
- **timeout** (*int*) – The number of seconds to wait before timing out a request.

Returns a configured and tested instance of Element

Raises SdkOperationError: verify_ssl is true but target is an IP address SdkOperationError: version is unable to be determined as float ApiVersionUnsupportedError: version is not supported by

instance of Element OS.

4.4 solidfire.models module

class solidfire.models.**Account**(*account_id*, *username*, *status*, *volumes*, *initiator_secret=None*, *target_secret=None*, *storage_container_id=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

The object containing information about an account. This object only includes "configured" information about the account, not any runtime or usage information.

Parameters

- **account_id** (*int*) – [required] Unique AccountID for the account.
- **username** (*str*) – [required] User name for the account.
- **status** (*str*) – [required] Current status of the account.
- **volumes** (*int*) – [required] List of VolumeIDs for Volumes owned by this account.
- **initiator_secret** (CHAPSecret) – CHAP secret to use for the initiator.
- **target_secret** (CHAPSecret) – CHAP secret to use for the target (mutual CHAP authentication).
- **storage_container_id** (*UUID*) – The id of the storage container associated with the account
- **attributes** (*dict*) – List of Name/Value pairs in JSON object format.

account_id = <type 'int'>

attributes = <type 'dict'>

initiator_secret = <class 'solidfire.models.CHAPSecret'>

status = <type 'str'>

storage_container_id = <class 'uuid.UUID'>

target_secret = <class 'solidfire.models.CHAPSecret'>

username = <type 'str'>

volumes = <type 'int[]'>

class solidfire.models.**AddAccountRequest**(*username*, *initiator_secret=None*, *target_secret=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use AddAccount to add a new account to the system. You can create new volumes under the new account. The CHAP settings you specify for the account apply to all volumes owned by the account.

Parameters

- **username** (*str*) – [required] Specifies the username for this account. (Might be 1 to 64 characters in length).

- **initiator_secret** (CHAPSecret) – The CHAP secret to use for the initiator. This secret must be 12-16 characters in length and should be impenetrable. The initiator CHAP secret must be unique and cannot be the same as the target CHAP secret. If unspecified, a random secret is created.
- **target_secret** (CHAPSecret) – The CHAP secret to use for the target (mutual CHAP authentication). This secret must be 12-16 characters in length and should be impenetrable. The target CHAP secret must be unique and cannot be the same as the initiator CHAP secret. If unspecified, a random secret is created.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

initiator_secret = <class 'solidfire.models.CHAPSecret'>

target_secret = <class 'solidfire.models.CHAPSecret'>

username = <type 'str'>

class solidfire.models.**AddAccountResult**(*account_id*, *account=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **account_id** (*int*) – [required] AccountID for the newly created Account.
- **account** (Account) – The full account object

account = <class 'solidfire.models.Account'>

account_id = <type 'int'>

class solidfire.models.**AddClusterAdminRequest**(*username*, *password*, *access*, *accept_eula=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use AddClusterAdmin to add a new cluster admin account. A cluster ddmin can manage the cluster using the API and management tools. Cluster admins are completely separate and unrelated to standard tenant accounts. Each cluster admin can be restricted to a subset of the API. NetApp recommends using multiple cluster admin accounts for different users and applications. You should give each cluster admin the minimal permissions necessary; this reduces the potential impact of credential compromise. You must accept the End User License Agreement (EULA) by setting the acceptEula parameter to true to add a cluster administrator account to the system.

Parameters

- **username** (*str*) – [required] Unique username for this cluster admin. Must be between 1 and 1024 characters in length.
- **password** (*str*) – [required] Password used to authenticate this cluster admin.
- **access** (*str*) – [required] Controls which methods this cluster admin can use. For more details on the levels of access, see Access Control in the Element API Reference Guide.
- **accept_eula** (*bool*) – Required to indicate your acceptance of the End User License Agreement when creating this cluster. To accept the EULA, set this parameter to true.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

accept_eula = <type 'bool'>

access = <type 'str[]'>

attributes = <type 'dict'>

password = <type 'str'>

username = <type 'str'>

class solidfire.models.**AddClusterAdminResult**(*cluster_admin_id*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_admin_id** (*int*) – [required] ClusterAdminID for the newly created Cluster Admin.

cluster_admin_id = <type 'int'>

class solidfire.models.**AddDrivesRequest**(*drives*, *force_during_upgrade=None*, *force_during_bin_sync=None*)
Bases: *solidfire.common.model.DataObject*

AddDrives enables you to add one or more available drives to the cluster, enabling the drives to host a portion of the cluster's data. When you add a node to the cluster or install new drives in an existing node, the new drives are marked as "available" and must be added via AddDrives before they can be utilized. Use the ListDrives method to display drives that are "available" to be added. When you add multiple drives, it is more efficient to add them in a single AddDrives method call rather than multiple individual methods with a single drive each. This reduces the amount of data balancing that must occur to stabilize the storage load on the cluster. When you add a drive, the system automatically determines the "type" of drive it should be. The method is asynchronous and returns immediately. However, it can take some time for the data in the cluster to be rebalanced using the newly added drives. As the new drives are syncing on the system, you can use the ListSyncJobs method to see how the drives are being rebalanced and the progress of adding the new drive. You can also use the GetAsyncResult method to query the method's returned asyncHandle.

Parameters

- **drives** (NewDrive) – [required] Returns information about each drive to be added to the cluster. Possible values are: driveID: The ID of the drive to add. (Integer) type: (Optional) The type of drive to add. Valid values are "slice" or "block". If omitted, the system assigns the correct type. (String)
- **force_during_upgrade** (*bool*) – Allows the user to force the addition of drives during an upgrade.
- **force_during_bin_sync** (*bool*) – Allows the user to force the addition of drives during a bin sync operation.

drives = <class 'solidfire.models.NewDrive[]'>

force_during_bin_sync = <type 'bool'>

force_during_upgrade = <type 'bool'>

class solidfire.models.**AddDrivesResult**(*async_handle=None*)
Bases: *solidfire.common.model.DataObject*

Parameters **async_handle** (*int*) –

async_handle = <type 'int'>

class solidfire.models.**AddInitiatorsToVolumeAccessGroupRequest**(*volume_access_group_id*, *initiators*)
Bases: *solidfire.common.model.DataObject*

AddInitiatorsToVolumeAccessGroup enables you to add initiators to a specified volume access group.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group to modify.

- **initiators** (*str*) – [required] The list of initiators to add to the volume access group.

initiators = <type 'str[]'>

volume_access_group_id = <type 'int'>

class solidfire.models.**AddLdapClusterAdminRequest**(*username*, *access*, *accept_eula=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

AddLdapClusterAdmin enables you to add a new LDAP cluster administrator user. An LDAP cluster administrator can manage the cluster via the API and management tools. LDAP cluster admin accounts are completely separate and unrelated to standard tenant accounts. You can also use this method to add an LDAP group that has been defined in Active Directory. The access level that is given to the group is passed to the individual users in the LDAP group.

Parameters

- **username** (*str*) – [required] The distinguished user name for the new LDAP cluster admin.
- **access** (*str*) – [required] Controls which methods this Cluster Admin can use. For more details on the levels of access, see the Access Control appendix in the SolidFire API Reference.
- **accept_eula** (*bool*) – Accept the End User License Agreement. Set to true to add a cluster administrator account to the system. If omitted or set to false, the method call fails.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

accept_eula = <type 'bool'>

access = <type 'str[]'>

attributes = <type 'dict'>

username = <type 'str'>

class solidfire.models.**AddLdapClusterAdminResult**(*cluster_admin_id=None*)

Bases: *solidfire.common.model.DataObject*

Parameters **cluster_admin_id** (*int*) –

cluster_admin_id = <type 'int'>

class solidfire.models.**AddNodesRequest**(*pending_nodes*, *auto_install=None*)

Bases: *solidfire.common.model.DataObject*

AddNodes enables you to add one or more new nodes to a cluster. When a node that is not configured starts up for the first time, you are prompted to configure the node. After you configure the node, it is registered as a "pending node" with the cluster. Note: It might take several seconds after adding a new node for it to start up and register its drives as available.

Parameters

- **pending_nodes** (*int*) – [required] List of pending NodeIDs for the nodes to be added. You can obtain the list of pending nodes using the ListPendingNodes method.
- **auto_install** (*bool*) – Whether these nodes should be autoinstalled

auto_install = <type 'bool'>

pending_nodes = <type 'int[]'>

class solidfire.models.**AddNodesResult**(*nodes*, *auto_install=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **auto_install** (*bool*) –
- **nodes** (AddedNode) – [required] An array of objects mapping the previous "pendingNodeID" to the "nodeID".

auto_install = <type 'bool'>

nodes = <class 'solidfire.models.AddedNode[]'>

class solidfire.models.**AddVirtualNetworkRequest**(*virtual_network_tag*, *name*, *address_blocks*, *netmask*, *svip*, *gateway=None*, *namespace=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use the AddVirtualNetwork method to add a new virtual network to a cluster configuration. When you add a virtual network, an interface for each node is created and each interface will require a virtual network IP address. The number of IP addresses you specify as a parameter for this API method must be equal to or greater than the number of nodes in the cluster. The system bulk provisions virtual network addresses and assigns them to individual nodes automatically. You do not need to assign virtual network addresses to nodes manually. Note: You can use AddVirtualNetwork only to create a new virtual network. If you want to make changes to an existing virtual network, use ModifyVirtualNetwork. Note: Virtual network parameters must be unique to each virtual network when setting the namespace parameter to false.

Parameters

- **virtual_network_tag** (*int*) – [required] A unique virtual network (VLAN) tag. Supported values are 1 through 4094.The number zero (0) is not supported.
- **name** (*str*) – [required] A user-defined name for the new virtual network.
- **address_blocks** (AddressBlockParams) – [required] Unique range of IP addresses to include in the virtual network. Attributes for this parameter are: start: The start of the IP address range. (String) size: The number of IP addresses to include in the block. (Integer)
- **netmask** (*str*) – [required] Unique network mask for the virtual network being created.
- **svip** (*str*) – [required] Unique storage IP address for the virtual network being created.
- **gateway** (*str*) – The IP address of a gateway of the virtual network. This parameter is only valid if the "namespace" parameter is set to true.
- **namespace** (*bool*) – When set to true, enables the Routable Storage VLANs functionality by creating and configuring a namespace and the virtual network contained by it.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

address_blocks = <class 'solidfire.models.AddressBlockParams[]'>

attributes = <type 'dict'>

gateway = <type 'str'>

name = <type 'str'>

namespace = <type 'bool'>

netmask = <type 'str'>

svip = <type 'str'>

virtual_network_tag = <type 'int'>

class solidfire.models.**AddVirtualNetworkResult**(*virtual_network_id=None*)

Bases: *solidfire.common.model.DataObject*

Parameters **virtual_network_id** (*int*) – The virtual network ID of the new virtual network.

virtual_network_id = <type 'int'>

class solidfire.models.**AddVolumesToVolumeAccessGroupRequest**(*volume_access_group_id*, *volumes*)

Bases: *solidfire.common.model.DataObject*

AddVolumesToVolumeAccessGroup enables you to add volumes to a specified volume access group.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group to which volumes are added.
- **volumes** (*int*) – [required] The list of volumes to add to the volume access group.

volume_access_group_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.models.**AddedNode**(*pending_node_id*, *node_id=None*, *active_node_key=None*, *assigned_node_id=None*, *async_handle=None*, *cip=None*, *mip=None*, *platform_info=None*, *sip=None*, *software_version=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) –
- **pending_node_id** (*int*) – [required]
- **active_node_key** (*str*) –
- **assigned_node_id** (*int*) –
- **async_handle** (*int*) –
- **cip** (*str*) –
- **mip** (*str*) –
- **platform_info** (Platform) –
- **sip** (*str*) –
- **software_version** (*str*) –

active_node_key = <type 'str'>

assigned_node_id = <type 'int'>

async_handle = <type 'int'>

cip = <type 'str'>

mip = <type 'str'>

node_id = <type 'int'>

pending_node_id = <type 'int'>

platform_info = <class 'solidfire.models.Platform'>

sip = <type 'str'>

software_version = <type 'str'>

class solidfire.models.**AddressBlock**(*start*, *size*, *available*)
Bases: *solidfire.common.model.DataObject*

Unique Range of IP addresses to include in the virtual network.

Parameters

- **start** (*str*) – [required] Start of the IP address range.
- **size** (*int*) – [required] Number of IP addresses to include in the block.
- **available** (*str*) – [required] Nuber of available blocks

available = <type 'str'>

size = <type 'int'>

start = <type 'str'>

class solidfire.models.**AddressBlockParams**(*start*, *size*)
Bases: *solidfire.common.model.DataObject*

Unique Range of IP addresses to include in the virtual network.

Parameters

- **start** (*str*) – [required] Start of the IP address range.
- **size** (*int*) – [required] Number of IP addresses to include in the block.

size = <type 'int'>

start = <type 'str'>

class solidfire.models.**AsyncHandle**(*async_result_id*, *completed*, *create_time*, *last_update_time*, *result_type*, *success*, *data*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **async_result_id** (*int*) – [required] The ID of the result.
- **completed** (*bool*) – [required] Returns true if it is completed and false if it isn't.
- **create_time** (*str*) – [required] The time at which the asyncronous result was created
- **last_update_time** (*str*) – [required] Time at which the result was last updated
- **result_type** (*str*) – [required] The type of result. Could be Clone, DriveAdd, etc.
- **success** (*bool*) – [required] Returns whether the result was a success or failure.
- **data** (*dict*) – [required] Attributes related to the result

async_result_id = <type 'int'>

completed = <type 'bool'>

create_time = <type 'str'>

data = <type 'dict'>

last_update_time = <type 'str'>

result_type = <type 'str'>

success = <type 'bool'>

class solidfire.models.**AsyncHandleResult**(*async_handle*)
Bases: *solidfire.common.model.DataObject*

Parameters **async_handle** (*int*) – [required]

async_handle = <type 'int'>

class solidfire.models.**BackupTarget**(*name*, *backup_target_id*, *attributes=None*)
Bases: *solidfire.common.model.DataObject*

The object containing information about a backup target.

Parameters

- **name** (*str*) – [required] Name for the backup target.
- **backup_target_id** (*int*) – [required] Unique identifier assigned to the backup target.
- **attributes** (*dict*) – List of Name/Value pairs in JSON object format.

attributes = <type 'dict'>

backup_target_id = <type 'int'>

name = <type 'str'>

class solidfire.models.**BulkVolumeJob**(*bulk_volume_id*, *create_time*, *elapsed_time*, *format*, *key*, *percent_complete*, *remaining_time*, *src_volume_id*, *status*, *type*, *attributes*, *script=None*, *snapshot_id=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **bulk_volume_id** (*int*) – [required] The internal bulk volume job ID.
- **create_time** (*str*) – [required] Timestamp created for the bulk volume job.
- **elapsed_time** (*int*) – [required] The number of seconds since the job began.
- **format** (*str*) – [required] Format is either “compressed” or “native”.
- **key** (*str*) – [required] The unique key created by the bulk volume session.
- **percent_complete** (*int*) – [required] The completed percentage reported by the operation.
- **remaining_time** (*int*) – [required] The estimated time remaining in seconds.
- **src_volume_id** (*int*) – [required] The source volume ID.
- **status** (*str*) – [required] Can be one of the following: preparing active done failed
- **script** (*str*) – The name of the script if one is provided.
- **snapshot_id** (*int*) – ID of the snapshot if a snapshot is in the source of the bulk volume job.
- **type** (*str*) – [required] Can be one of the following: read write
- **attributes** (*dict*) – [required] JSON attributes on the bulk volume job.

attributes = <type 'dict'>

bulk_volume_id = <type 'int'>

create_time = <type 'str'>

elapsed_time = <type 'int'>

format = <type 'str'>

key = <type 'str'>

percent_complete = <type 'int'>

remaining_time = <type 'int'>

script = <type 'str'>

snapshot_id = <type 'int'>

src_volume_id = <type 'int'>

status = <type 'str'>

type = <type 'str'>

class solidfire.models.**CHAPSecret**(***kwargs*)
Bases: solidfire.custom.models.CHAPSecret

class solidfire.models.**CancelCloneRequest**(*clone_id*)
Bases: *solidfire.common.model.DataObject*

CancelClone enables you to stop an ongoing CloneVolume or CopyVolume process. When you cancel a group clone operation, the system completes and removes the operation's associated asyncHandle.

Parameters **clone_id** (*int*) – [required] The cloneID for the ongoing clone process.

clone_id = <type 'int'>

class solidfire.models.**CancelCloneResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**CancelGroupCloneRequest**(*group_clone_id*)
Bases: *solidfire.common.model.DataObject*

CancelGroupClone enables you to stop an ongoing CloneMultipleVolumes process occurring on a group of volumes. When you cancel a group clone operation, the system completes and removes the operation's associated asyncHandle.

Parameters **group_clone_id** (*int*) – [required] The cloneID for the ongoing clone process.

group_clone_id = <type 'int'>

class solidfire.models.**CancelGroupCloneResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**ClearClusterFaultsRequest**(*fault_types=None*)
Bases: *solidfire.common.model.DataObject*

You can use the ClearClusterFaults method to clear information about both current and previously detected faults. Both resolved and unresolved faults can be cleared.

Parameters **fault_types** (*str*) – Determines the types of faults cleared. Possible values are: current: Faults that are currently detected and have not been resolved. resolved: (Default) Faults that were previously detected and resolved. all: Both current and resolved faults are cleared. The fault status can be determined by the resolved field of the fault object.

fault_types = <type 'str'>

class solidfire.models.**ClearClusterFaultsResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**CloneMultipleVolumeParams**(*volume_id*, *access=None*, *name=None*, *new_account_id=None*, *new_size=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **volume_id** (*int*) – [required] Required parameter for "volumes" array: volumeID.
- **access** (*str*) – Access settings for the new volume. readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked. If unspecified, the access settings of the clone will be the same as the source.
- **name** (*str*) – New name for the clone.
- **new_account_id** (*int*) – Account ID for the new volume.
- **new_size** (*int*) – New size Total size of the volume, in bytes. Size is rounded up to the nearest 1MB size.
- **attributes** (*dict*) – List of Name/Value pairs in JSON object format.

access = <type 'str'>

attributes = <type 'dict'>

name = <type 'str'>

new_account_id = <type 'int'>

new_size = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**CloneMultipleVolumesRequest**(*volumes*, *access=None*, *group_snapshot_id=None*, *new_account_id=None*)

Bases: *solidfire.common.model.DataObject*

CloneMultipleVolumes enables you to create a clone of a group of specified volumes. You can assign a consistent set of characteristics to a group of multiple volumes when they are cloned together. Before using groupSnapshotID to clone the volumes in a group snapshot, you must create the group snapshot by using the CreateGroupSnapshot API method or the Element OS Web UI. Using groupSnapshotID is optional when cloning multiple volumes. Note: Cloning multiple volumes is allowed if cluster fullness is at stage 2 or 3. Clones are not created when cluster fullness is at stage 4 or 5.

Parameters

- **volumes** (CloneMultipleVolumeParams) – [required] Unique ID for each volume to include in the clone. If optional parameters are not specified, the values are inherited from the source volumes. Required parameter for "volumes" array: volumeID Optional parameters for "volumes" array: access: Can be one of readOnly, readWrite, locked, or replicationTarget attributes: List of name-value pairs in JSON object format. name: New name for the clone. newAccountID: Account ID for the new volumes. newSize: New size Total size of the volume, in bytes. Size is rounded up to the nearest 1MB.
- **access** (*str*) – New default access method for the new volumes if not overridden by information passed in the volume's array.
- **group_snapshot_id** (*int*) – ID of the group snapshot to use as a basis for the clone.

- **new_account_id** (*int*) – New account ID for the volumes if not overridden by information passed in the volumes array.

access = <type 'str'>

group_snapshot_id = <type 'int'>

new_account_id = <type 'int'>

volumes = <class 'solidfire.models.CloneMultipleVolumeParams[]'>

class solidfire.models.**CloneMultipleVolumesResult**(*async_handle*, *group_clone_id*, *members*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **async_handle** (*int*) – [required] A value returned from an asynchronous method call.
- **group_clone_id** (*int*) – [required] Unique ID of the new group clone.
- **members** (GroupCloneVolumeMember) – [required] List of volumeIDs for the source and destination volume pairs.

async_handle = <type 'int'>

group_clone_id = <type 'int'>

members = <class 'solidfire.models.GroupCloneVolumeMember[]'>

class solidfire.models.**CloneVolumeRequest**(*volume_id*, *name*, *new_account_id=None*, *new_size=None*, *access=None*, *snapshot_id=None*, *attributes=None*, *enable512e=None*)

Bases: *solidfire.common.model.DataObject*

CloneVolume enables you to create a copy of a volume. This method is asynchronous and might take a variable amount of time to complete. The cloning process begins immediately when you make the CloneVolume request and is representative of the state of the volume when the API method is issued. You can use the GetAsyncResult method to determine when the cloning process is complete and the new volume is available for connections. You can use ListSyncJobs to see the progress of creating the clone. Note: The initial attributes and QoS settings for the volume are inherited from the volume being cloned. You can change these settings with ModifyVolume. Note: Cloned volumes do not inherit volume access group memberships from the source volume.

Parameters

- **volume_id** (*int*) – [required] VolumeID for the volume to be cloned.
- **name** (*str*) – [required] The name of the new cloned volume. Might be 1 to 64 characters in length.
- **new_account_id** (*int*) – AccountID for the owner of the new volume. If unspecified, the accountID of the owner of the volume being cloned is used.
- **new_size** (*int*) – New size of the volume, in bytes. Might be greater or less than the size of the volume being cloned. If unspecified, the volume size is not changed. Size is rounded to the nearest 1MB.
- **access** (*str*) – Specifies the level of access allowed for the new volume. Possible values are: readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. If unspecified, the level of access of the volume being cloned is used. replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked. If a value is not specified, the access value does not change.

- **snapshot_id** (*int*) – ID of the snapshot that is used as the source of the clone. If no ID is provided, the current active volume is used.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **enable512e** (*bool*) – Should the volume provide 512-byte sector emulation?

access = <type 'str'>

attributes = <type 'dict'>

enable512e = <type 'bool'>

name = <type 'str'>

new_account_id = <type 'int'>

new_size = <type 'int'>

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**CloneVolumeResult**(*clone_id*, *volume_id*, *async_handle*, *volume=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **volume** (Volume) – The resulting volume
- **clone_id** (*int*) – [required] The ID of the newly-created clone.
- **volume_id** (*int*) – [required] The volume ID of the newly-created clone.
- **async_handle** (*int*) – [required] Handle value used to track the progress of the clone.

async_handle = <type 'int'>

clone_id = <type 'int'>

volume = <class 'solidfire.models.Volume'>

volume_id = <type 'int'>

class solidfire.models.**ClusterAdmin**(*auth_method*, *access*, *cluster_admin_id*, *username*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **auth_method** (*str*) – [required]
- **access** (*str*) – [required]
- **cluster_admin_id** (*int*) – [required]
- **username** (*str*) – [required]
- **attributes** (*dict*) – List of Name/Value pairs in JSON object format.

access = <type 'str[]'>

attributes = <type 'dict'>

auth_method = <type 'str'>

cluster_admin_id = <type 'int'>

username = <type 'str'>

class solidfire.models.**ClusterCapacity**(*active_block_space*, *active_sessions*, *average_iops*, *cluster_recent_iosize*, *current_iops*, *max_iops*, *max_over_provisionable_space*, *max_provisioned_space*, *max_used_metadata_space*, *max_used_space*, *non_zero_blocks*, *peak_active_sessions*, *peak_iops*, *provisioned_space*, *snapshot_non_zero_blocks*, *timestamp*, *total_ops*, *unique_blocks*, *unique_blocks_used_space*, *used_metadata_space*, *used_metadata_space_in_snapshots*, *used_space*, *zero_blocks*)

Bases: *solidfire.common.model.DataObject*

High level capacity measurements for the entire cluster.

Parameters

- **active_block_space** (*int*) – [required] The amount of space on the block drives. This includes additional information such as metadata entries and space which can be cleaned up.
- **active_sessions** (*int*) – [required] Number of active iSCSI sessions communicating with the cluster
- **average_iops** (*int*) – [required] Average IPS for the cluster since midnight Coordinated Universal Time (UTC).
- **cluster_recent_iosize** (*int*) – [required] The average size of IOPS to all volumes in the cluster.
- **current_iops** (*int*) – [required] Average IOPS for all volumes in the cluster over the last 5 seconds.
- **max_iops** (*int*) – [required] Estimated maximum IOPS capability of the current cluster.
- **max_over_provisionable_space** (*int*) – [required] The maximum amount of provisionable space. This is a computed value. You cannot create new volumes if the current provisioned space plus the new volume size would exceed this number: maxOverProvisionableSpace = maxProvisionedSpace * GetClusterFull
- **max_provisioned_space** (*int*) – [required] The total amount of provisionable space if all volumes are 100% filled (no thin provisioned metadata).
- **max_used_metadata_space** (*int*) – [required] The amount of bytes on volume drives used to store metadata.
- **max_used_space** (*int*) – [required] The total amount of space on all active block drives.
- **non_zero_blocks** (*int*) – [required] Total number of 4KiB blocks with data after the last garbage collection operation has completed.
- **peak_active_sessions** (*int*) – [required] Peak number of iSCSI connections since midnight UTC.
- **peak_iops** (*int*) – [required] The highest value for currentIOPS since midnight UTC.
- **provisioned_space** (*int*) – [required] Total amount of space provisioned in all volumes on the cluster.
- **snapshot_non_zero_blocks** (*int*) – [required] Total number of 4KiB blocks in snapshots with data.
- **timestamp** (*str*) – [required] The date and time this cluster capacity sample was taken.

- **total_ops** (*int*) – [required] The total number of I/O operations performed throughout the lifetime of the cluster
- **unique_blocks** (*int*) – [required] The total number of blocks stored on the block drives. The value includes replicated blocks.
- **unique_blocks_used_space** (*int*) – [required] The total amount of data the uniqueBlocks take up on the block drives. This number is always consistent with the uniqueBlocks value.
- **used_metadata_space** (*int*) – [required] The total amount of bytes on volume drives used to store metadata
- **used_metadata_space_in_snapshots** (*int*) – [required] The amount of bytes on volume drives used for storing unique data in snapshots. This number provides an estimate of how much metadata space would be regained by deleting all snapshots on the system.
- **used_space** (*int*) – [required] Total amount of space used by all block drives in the system.
- **zero_blocks** (*int*) – [required] Total number of 4KiB blocks without data after the last round of garabage collection operation has completed.

```
active_block_space = <type 'int'>
active_sessions = <type 'int'>
average_iops = <type 'int'>
cluster_recent_iosize = <type 'int'>
current_iops = <type 'int'>
max_iops = <type 'int'>
max_over_provisionable_space = <type 'int'>
max_provisioned_space = <type 'int'>
max_used_metadata_space = <type 'int'>
max_used_space = <type 'int'>
non_zero_blocks = <type 'int'>
peak_active_sessions = <type 'int'>
peak_iops = <type 'int'>
provisioned_space = <type 'int'>
snapshot_non_zero_blocks = <type 'int'>
timestamp = <type 'str'>
total_ops = <type 'int'>
unique_blocks = <type 'int'>
unique_blocks_used_space = <type 'int'>
used_metadata_space = <type 'int'>
used_metadata_space_in_snapshots = <type 'int'>
used_space = <type 'int'>
zero_blocks = <type 'int'>
```

class solidfire.models.**ClusterConfig**(*cipi=None*, *cluster=None*, *ensemble=None*, *mipi=None*, *name=None*, *node_id=None*, *pending_node_id=None*, *role=None*, *sipi=None*, *state=None*, *encryption_capable=None*, *has_local_admin=None*, *version=None*)

Bases: *solidfire.common.model.DataObject*

Cluster Config object returns information the node uses to communicate with the cluster.

Parameters

- **cipi** (*str*) – Network interface used for cluster communication.
- **cluster** (*str*) – Unique cluster name.
- **ensemble** (*str*) – Nodes that are participating in the cluster.
- **mipi** (*str*) – Network interface used for node management.
- **name** (*str*) – Unique cluster name.
- **node_id** (*int*) –
- **pending_node_id** (*int*) –
- **role** (*str*) – Identifies the role of the node
- **sipi** (*str*) – Network interface used for storage.
- **state** (*str*) –
- **encryption_capable** (*bool*) –
- **has_local_admin** (*bool*) –
- **version** (*str*) –

cipi = <type 'str'>

cluster = <type 'str'>

encryption_capable = <type 'bool'>

ensemble = <type 'str[]'>

has_local_admin = <type 'bool'>

mipi = <type 'str'>

name = <type 'str'>

node_id = <type 'int'>

pending_node_id = <type 'int'>

role = <type 'str'>

sipi = <type 'str'>

state = <type 'str'>

version = <type 'str'>

class solidfire.models.**ClusterFaultInfo**(*severity*, *type*, *code*, *details*, *node_hardware_fault_id*, *node_id*, *service_id*, *drive_id*, *resolved*, *cluster_fault_id*, *date*, *resolved_date*, *drive_ids=None*, *network_interface=None*, *data=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **drive_ids** (*int*) –
- **network_interface** (*str*) –
- **severity** (*str*) – [required]
- **type** (*str*) – [required]
- **code** (*str*) – [required]
- **details** (*str*) – [required]
- **node_hardware_fault_id** (*int*) – [required]
- **node_id** (*int*) – [required]
- **service_id** (*int*) – [required]
- **drive_id** (*int*) – [required]
- **resolved** (*bool*) – [required]
- **cluster_fault_id** (*int*) – [required]
- **date** (*str*) – [required]
- **resolved_date** (*str*) – [required]
- **data** (*dict*) –

cluster_fault_id = <type 'int'>

code = <type 'str'>

data = <type 'dict'>

date = <type 'str'>

details = <type 'str'>

drive_id = <type 'int'>

drive_ids = <type 'int[]'>

network_interface = <type 'str'>

node_hardware_fault_id = <type 'int'>

node_id = <type 'int'>

resolved = <type 'bool'>

resolved_date = <type 'str'>

service_id = <type 'int'>

severity = <type 'str'>

type = <type 'str'>

class solidfire.models.**ClusterHardwareInfo**(*drives*, *nodes*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **drives** (*dict*) – [required]
- **nodes** (*dict*) – [required]

drives = <type 'dict'>

nodes = <type 'dict'>

class solidfire.models.**ClusterInfo**(*encryption_at_rest_state*, *ensemble*, *mvip*, *mvip_node_id*, *name*, *rep_count*, *svip*, *svip_node_id*, *unique_id*, *uuid*, *attributes*, *mvip_interface=None*, *mvip_vlan_tag=None*, *svip_interface=None*, *svip_vlan_tag=None*)

Bases: *solidfire.common.model.DataObject*

Cluster Info object returns information the node uses to communicate with the cluster.

Parameters

- **mvip_interface** (*str*) –
- **mvip_vlan_tag** (*str*) –
- **svip_interface** (*str*) –
- **svip_vlan_tag** (*str*) –
- **encryption_at_rest_state** (*str*) – [required] Encryption at rest state.
- **ensemble** (*str*) – [required] Array of Node IP addresses that are participating in the cluster.
- **mvip** (*str*) – [required] Management network interface.
- **mvip_node_id** (*int*) – [required] Node holding the master MVIP address
- **name** (*str*) – [required] Unique cluster name.
- **rep_count** (*int*) – [required] Number of replicas of each piece of data to store in the cluster. Valid value is 2
- **svip** (*str*) – [required] Storage virtual IP
- **svip_node_id** (*int*) – [required] Node holding the master SVIP address.
- **unique_id** (*str*) – [required] Unique ID for the cluster.
- **uuid** (*UUID*) – [required]
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.

attributes = <type 'dict'>

encryption_at_rest_state = <type 'str'>

ensemble = <type 'str[]'>

mvip = <type 'str'>

mvip_interface = <type 'str'>

mvip_node_id = <type 'int'>

mvip_vlan_tag = <type 'str'>

name = <type 'str'>

rep_count = <type 'int'>

svip = <type 'str'>

svip_interface = <type 'str'>

svip_node_id = <type 'int'>

svip_vlan_tag = <type 'str'>

unique_id = <type 'str'>

uuid = <class 'uuid.UUID'>

class solidfire.models.**ClusterStats**(*cluster_utilization*, *client_queue_depth*, *normalized_iops*, *read_bytes*, *read_latency_usec_total*, *read_ops*, *services_count*, *services_total*, *timestamp*, *write_bytes*, *write_latency_usec_total*, *write_ops*, *actual_iops=None*, *average_iopsize=None*, *latency_usec=None*, *read_bytes_last_sample=None*, *read_latency_usec=None*, *read_ops_last_sample=None*, *sample_period_msec=None*, *unaligned_reads=None*, *unaligned_writes=None*, *write_bytes_last_sample=None*, *write_latency_usec=None*, *write_ops_last_sample=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster_utilization** (*float*) – [required] The amount of cluster capacity being utilized.
- **client_queue_depth** (*int*) – [required]
- **normalized_iops** (*int*) – [required]
- **read_bytes** (*int*) – [required] Total bytes read by clients.
- **read_latency_usec_total** (*int*) – [required]
- **read_ops** (*int*) – [required] Total read operations.
- **services_count** (*int*) – [required] Services count
- **services_total** (*int*) – [required] Total services.
- **timestamp** (*str*) – [required] Current time in UTC format. ISO 8601 date string.
- **write_bytes** (*int*) – [required] Total bytes written by clients.
- **write_latency_usec_total** (*int*) – [required]
- **write_ops** (*int*) – [required] Total write operations.
- **actual_iops** (*int*) –
- **average_iopsize** (*int*) –
- **latency_usec** (*int*) –
- **read_bytes_last_sample** (*int*) –
- **read_latency_usec** (*int*) –
- **read_ops_last_sample** (*int*) –
- **sample_period_msec** (*int*) –
- **unaligned_reads** (*int*) –
- **unaligned_writes** (*int*) –
- **write_bytes_last_sample** (*int*) –
- **write_latency_usec** (*int*) –

- **write_ops_last_sample** (*int*) –

actual_iops = <type 'int'>

average_iopsize = <type 'int'>

client_queue_depth = <type 'int'>

cluster_utilization = <type 'float'>

latency_usec = <type 'int'>

normalized_iops = <type 'int'>

read_bytes = <type 'int'>

read_bytes_last_sample = <type 'int'>

read_latency_usec = <type 'int'>

read_latency_usec_total = <type 'int'>

read_ops = <type 'int'>

read_ops_last_sample = <type 'int'>

sample_period_msec = <type 'int'>

services_count = <type 'int'>

services_total = <type 'int'>

timestamp = <type 'str'>

unaligned_reads = <type 'int'>

unaligned_writes = <type 'int'>

write_bytes = <type 'int'>

write_bytes_last_sample = <type 'int'>

write_latency_usec = <type 'int'>

write_latency_usec_total = <type 'int'>

write_ops = <type 'int'>

write_ops_last_sample = <type 'int'>

class solidfire.models.**ClusterVersionInfo**(*node_id*, *node_version*, *node_internal_revision*)

Bases: *solidfire.common.model.DataObject*

Version information for a node in the cluster.

Parameters

- **node_id** (*int*) – [required]
- **node_version** (*str*) – [required]
- **node_internal_revision** (*str*) – [required]

node_id = <type 'int'>

node_internal_revision = <type 'str'>

node_version = <type 'str'>

class solidfire.models.**CompleteClusterPairingRequest**(*cluster_pairing_key*)
Bases: *solidfire.common.model.DataObject*

You can use the CompleteClusterPairing method with the encoded key received from the StartClusterPairing method to complete the cluster pairing process. The CompleteClusterPairing method is the second step in the cluster pairing process.

Parameters **cluster_pairing_key** (*str*) – [required] A string of characters that is returned from the "StartClusterPairing" API method.

cluster_pairing_key = <type 'str'>

class solidfire.models.**CompleteClusterPairingResult**(*cluster_pair_id*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_pair_id** (*int*) – [required] Unique identifier for the cluster pair.

cluster_pair_id = <type 'int'>

class solidfire.models.**CompleteVolumePairingRequest**(*volume_pairing_key*, *volume_id*)
Bases: *solidfire.common.model.DataObject*

You can use the CompleteVolumePairing method to complete the pairing of two volumes.

Parameters

- **volume_pairing_key** (*str*) – [required] The key returned from the StartVolumePairing method.
- **volume_id** (*int*) – [required] The ID of the volume on which to complete the pairing process.

volume_id = <type 'int'>

volume_pairing_key = <type 'str'>

class solidfire.models.**CompleteVolumePairingResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**Config**(*cluster*, *network*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster** (ClusterConfig) – [required]
- **network** (Network) – [required]

cluster = <class 'solidfire.models.ClusterConfig'>

network = <class 'solidfire.models.Network'>

class solidfire.models.**ConfigParams**(*cluster*, *network*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster** (ClusterConfig) – [required]
- **network** (NetworkParams) – [required]

cluster = <class 'solidfire.models.ClusterConfig'>

network = <class 'solidfire.models.NetworkParams'>

class solidfire.models.**CopyVolumeRequest**(*volume_id*, *dst_volume_id*, *snapshot_id=None*)
Bases: *solidfire.common.model.DataObject*

CopyVolume enables you to overwrite the data contents of an existing volume with the data contents of another volume (or snapshot). Attributes of the destination volume such as IQN, QoS settings, size, account, and volume access group membership are not changed. The destination volume must already exist and must be the same size as the source volume. NetApp strongly recommends that clients unmount the destination volume before the CopyVolume operation begins. If the destination volume is modified during the copy operation, the changes will be lost. This method is asynchronous and may take a variable amount of time to complete. You can use the GetAsyncResult method to determine when the process has finished, and ListSyncJobs to see the progress of the copy.

Parameters

- **volume_id** (*int*) – [required] VolumeID of the volume to be read from.
- **dst_volume_id** (*int*) – [required] VolumeID of the volume to be overwritten.
- **snapshot_id** (*int*) – ID of the snapshot that is used as the source of the clone. If no ID is provided, the current active volume is used.

dst_volume_id = <type 'int'>

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**CopyVolumeResult**(*clone_id*, *async_handle*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **clone_id** (*int*) – [required]
- **async_handle** (*int*) – [required] Handle value used to track the progress of the volume copy.

async_handle = <type 'int'>

clone_id = <type 'int'>

class solidfire.models.**CreateBackupTargetRequest**(*name*, *attributes*)
Bases: *solidfire.common.model.DataObject*

CreateBackupTarget enables you to create and store backup target information so that you do not need to re-enter it each time a backup is created.

Parameters

- **name** (*str*) – [required] The name of the backup target.
- **attributes** (*dict*) – [required] List of name-value pairs in JSON object format.

attributes = <type 'dict'>

name = <type 'str'>

class solidfire.models.**CreateBackupTargetResult**(*backup_target_id*)
Bases: *solidfire.common.model.DataObject*

Parameters **backup_target_id** (*int*) – [required] Unique identifier assigned to the backup target.

backup_target_id = <type 'int'>

class solidfire.models.**CreateClusterRequest**(*mvip*, *svip*, *rep_count*, *username*, *password*, *nodes*, *accept_eula=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

The CreateCluster method enables you to initialize the node in a cluster that has ownership of the "mvip" and "svip" addresses. Each new cluster is initialized using the management IP (MIP) of the first node in the cluster. This method also automatically adds all the nodes being configured into the cluster. You only need to use this method once each time a new cluster is initialized. Note: You need to log in to the node that is used as the master node for the cluster. After you log in, run the GetBootstrapConfig method on the node to get the IP addresses for the rest of the nodes that you want to include in the cluster. Then, run the CreateCluster method.

Parameters

- **accept_eula** (*bool*) – Required to indicate your acceptance of the End User License Agreement when creating this cluster. To accept the EULA, set this parameter to true.
- **mvip** (*str*) – [required] Floating (virtual) IP address for the cluster on the management network.
- **svip** (*str*) – [required] Floating (virtual) IP address for the cluster on the storage (iSCSI) network.
- **rep_count** (*int*) – [required] Number of replicas of each piece of data to store in the cluster. Valid value is "2".
- **username** (*str*) – [required] Username for the cluster admin.
- **password** (*str*) – [required] Initial password for the cluster admin account.
- **nodes** (*str*) – [required] CIP/SIP addresses of the initial set of nodes making up the cluster. This node's IP must be in the list.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

accept_eula = <type 'bool'>

attributes = <type 'dict'>

mvip = <type 'str'>

nodes = <type 'str[]'>

password = <type 'str'>

rep_count = <type 'int'>

svip = <type 'str'>

username = <type 'str'>

class solidfire.models.**CreateClusterResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**CreateGroupSnapshotRequest**(*volumes*, *name=None*, *enable_remote_replication=None*, *retention=None*, *attributes=None*, *snap_mirror_label=None*)

Bases: *solidfire.common.model.DataObject*

CreateGroupSnapshot enables you to create a point-in-time copy of a group of volumes. You can use this snapshot later as a backup or rollback to ensure the data on the group of volumes is consistent for the point in time that you created the snapshot. Note: Creating a group snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5.

Parameters

- **volumes** (*int*) – [required] Unique ID of the volume image from which to copy.
- **name** (*str*) – Name for the group snapshot. If unspecified, the date and time the group snapshot was taken is used.
- **enable_remote_replication** (*bool*) – Replicates the snapshot created to remote storage. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **retention** (*str*) – Specifies the amount of time for which the snapshots are retained. The format is HH:mm:ss.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **snap_mirror_label** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

attributes = <type 'dict'>

enable_remote_replication = <type 'bool'>

name = <type 'str'>

retention = <type 'str'>

snap_mirror_label = <type 'str'>

volumes = <type 'int[]'>

class solidfire.models.**CreateGroupSnapshotResult**(*group_snapshot*, *group_snapshot_id*, *members*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **group_snapshot** (GroupSnapshot) – [required]
- **group_snapshot_id** (*int*) – [required] Unique ID of the new group snapshot.
- **members** (GroupSnapshotMembers) – [required] List of checksum, volumeIDs and snapshotIDs for each member of the group.

group_snapshot = <class 'solidfire.models.GroupSnapshot'>

group_snapshot_id = <type 'int'>

members = <class 'solidfire.models.GroupSnapshotMembers[]'>

class solidfire.models.**CreateInitiator**(*name*, *alias=None*, *volume_access_group_id=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

Object containing characteristics of each new initiator.

Parameters

- **name** (*str*) – [required] (Required) The name of the initiator (IQN or WWPN) to create. (String)
- **alias** (*str*) – (Optional) The friendly name to assign to this initiator. (String)
- **volume_access_group_id** (*int*) – (Optional) The ID of the volume access group to which this newly created initiator will be added. (Integer)
- **attributes** (*dict*) – (Optional) A set of JSON attributes assigned to this initiator. (JSON Object)

alias = <type 'str'>

attributes = <type 'dict'>

name = <type 'str'>

volume_access_group_id = <type 'int'>

class solidfire.models.**CreateInitiatorsRequest**(*initiators*)
Bases: *solidfire.common.model.DataObject*

CreateInitiators enables you to create multiple new initiator IQNs or World Wide Port Names (WWPNs) and optionally assign them aliases and attributes. When you use CreateInitiators to create new initiators, you can also add them to volume access groups. If CreateInitiators fails to create one of the initiators provided in the parameter, the method returns an error and does not create any initiators (no partial completion is possible).

Parameters **initiators** (CreateInitiator) – [required] A list of objects containing characteristics of each new initiator. Values are: name: (Required) The name of the initiator (IQN or WWPN) to create. (String) alias: (Optional) The friendly name to assign to this initiator. (String) attributes: (Optional) A set of JSON attributes to assign to this initiator. (JSON Object) volumeAccessGroupID: (Optional) The ID of the volume access group into to which this newly created initiator will be added. (Integer)

initiators = <class 'solidfire.models.CreateInitiator[]'>

class solidfire.models.**CreateInitiatorsResult**(*initiators*)
Bases: *solidfire.common.model.DataObject*

Parameters **initiators** (Initiator) – [required] List of objects containing details about the newly created initiators

initiators = <class 'solidfire.models.Initiator[]'>

class solidfire.models.**CreateQoSPolicyRequest**(*name*, *qos*)
Bases: *solidfire.common.model.DataObject*

You can use the CreateQoSPolicy method to create a QoSPolicy object that you can later apply to a volume upon creation or modification. A QoS policy has a unique ID, a name, and QoS settings.

Parameters

- **name** (*str*) – [required] The name of the QoS policy; for example, gold, platinum, or silver.
- **qos** (QoS) – [required] The QoS settings that this policy represents.

name = <type 'str'>

qos = <class 'solidfire.models.QoS'>

class solidfire.models.**CreateQoSPolicyResult**(*qos_policy*)
Bases: *solidfire.common.model.DataObject*

Parameters **qos_policy** (QoSPolicy) – [required] The newly created QoSPolicy object.

qos_policy = <class 'solidfire.models.QoSPolicy'>

class solidfire.models.**CreateScheduleRequest**(*schedule*)
Bases: *solidfire.common.model.DataObject*

CreateSchedule enables you to schedule an automatic snapshot of a volume at a defined interval. You can use the created snapshot later as a backup or rollback to ensure the data on a volume or group of volumes is consistent for the point in time in which the snapshot was created. If you schedule a snapshot to run at a time period that is not divisible by 5 minutes, the snapshot runs at the next time period that is divisible by 5 minutes. For example, if you schedule a snapshot to run at 12:42:00 UTC, it runs at 12:45:00 UTC. Note: You can create snapshots if cluster fullness is at stage 1, 2 or 3. You cannot create snapshots after cluster fullness reaches stage 4 or 5.

Parameters **schedule** (Schedule) – [required] The "Schedule" object will be used to create a new schedule. Do not set ScheduleID property, it will be ignored. Frequency property must be of type that inherits from Frequency. Valid types are: DaysOfMonthFrequency DaysOrWeekFrequency TimeIntervalFrequency

schedule = <class 'solidfire.models.Schedule'>

class solidfire.models.**CreateScheduleResult**(*schedule_id*)

Bases: *solidfire.common.model.DataObject*

Parameters **schedule_id** (*int*) – [required]

schedule_id = <type 'int'>

class solidfire.models.**CreateSnapshotRequest**(*volume_id*, *snapshot_id=None*, *name=None*, *enable_remote_replication=None*, *retention=None*, *attributes=None*, *snap_mirror_label=None*)

Bases: *solidfire.common.model.DataObject*

CreateSnapshot enables you to create a point-in-time copy of a volume. You can create a snapshot from any volume or from an existing snapshot. If you do not provide a SnapshotID with this API method, a snapshot is created from the volume's active branch. If the volume from which the snapshot is created is being replicated to a remote cluster, the snapshot can also be replicated to the same target. Use the enableRemoteReplication parameter to enable snapshot replication. Note: Creating a snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5.

Parameters

- **volume_id** (*int*) – [required] Specifies the unique ID of the volume image from which to copy.
- **snapshot_id** (*int*) – Specifies the unique ID of a snapshot from which the new snapshot is made. The snapshotID passed must be a snapshot on the given volume.
- **name** (*str*) – Specifies a name for the snapshot. If unspecified, the date and time the snapshot was taken is used.
- **enable_remote_replication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **retention** (*str*) – Specifies the amount of time for which the snapshot is retained. The format is HH:mm:ss.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **snap_mirror_label** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

attributes = <type 'dict'>

enable_remote_replication = <type 'bool'>

name = <type 'str'>

retention = <type 'str'>

snap_mirror_label = <type 'str'>

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**CreateSnapshotResult**(*snapshot*, *snapshot_id*, *checksum*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **snapshot** (Snapshot) – [required]
- **snapshot_id** (*int*) – [required] ID of the newly-created snapshot.
- **checksum** (*str*) – [required] A string that represents the correct digits in the stored snapshot. This checksum can be used later to compare other snapshots to detect errors in the data.

checksum = <type 'str'>

snapshot = <class 'solidfire.models.Snapshot'>

snapshot_id = <type 'int'>

class solidfire.models.**CreateStorageContainerRequest**(*name*, *initiator_secret=None*, *target_secret=None*, *account_id=None*)
Bases: *solidfire.common.model.DataObject*

CreateStorageContainer enables you to create a Virtual Volume (VVol) storage container. Storage containers are associated with a SolidFire storage system account, and are used for reporting and resource allocation. Storage containers can only be associated with virtual volumes. You need at least one storage container to use the Virtual Volumes feature.

Parameters

- **name** (*str*) – [required] The name of the storage container. Follows SolidFire account naming restrictions.
- **initiator_secret** (*str*) – The secret for CHAP authentication for the initiator.
- **target_secret** (*str*) – The secret for CHAP authentication for the target.
- **account_id** (*int*) – Non-storage container account that will become a storage container.

account_id = <type 'int'>

initiator_secret = <type 'str'>

name = <type 'str'>

target_secret = <type 'str'>

class solidfire.models.**CreateStorageContainerResult**(*storage_container*)
Bases: *solidfire.common.model.DataObject*

Parameters **storage_container** (StorageContainer) – [required]

storage_container = <class 'solidfire.models.StorageContainer'>

class solidfire.models.**CreateSupportBundleRequest**(*bundle_name=None*, *extra_args=None*, *timeout_sec=None*)
Bases: *solidfire.common.model.DataObject*

CreateSupportBundle enables you to create a support bundle file under the node's directory. After creation, the bundle is stored on the node as a tar.gz file.

Parameters

- **bundle_name** (*str*) – The unique name for the support bundle. If no name is provided, "supportbundle" and the node name are used as the filename.

- **extra_args** (*str*) – Passed to the sf_make_support_bundle script. You should use this parameter only at the request of NetApp SolidFire Support.
- **timeout_sec** (*int*) – The number of seconds to allow the support bundle script to run before stopping. The default value is 1500 seconds.

bundle_name = <type 'str'>

extra_args = <type 'str'>

timeout_sec = <type 'int'>

class solidfire.models.**CreateSupportBundleResult**(*details*, *duration*, *result*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (SupportBundleDetails) – [required] The details of the support bundle.
- **duration** (*str*) – [required] The amount of time required to create the support bundle in the format HH:MM:SS.ssssss
- **result** (*str*) – [required] Whether the support bundle creation passed of failed.

details = <class 'solidfire.models.SupportBundleDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**CreateVolumeAccessGroupRequest**(*name*, *initiators=None*, *volumes=None*, *virtual_network_id=None*, *virtual_network_tags=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use CreateVolumeAccessGroup to create a new volume access group. When you create the volume access group, you need to give it a name, and you can optionally enter initiators and volumes. After you create the group, you can add volumes and initiator IQNs. Any initiator IQN that you add to the volume access group is able to access any volume in the group without CHAP authentication.

Parameters

- **name** (*str*) – [required] The name for this volume access group. Not required to be unique, but recommended.
- **initiators** (*str*) – List of initiators to include in the volume access group. If unspecified, the access group's configured initiators are not modified.
- **volumes** (*int*) – List of volumes to initially include in the volume access group. If unspecified, the access group's volumes are not modified.
- **virtual_network_id** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **virtual_network_tags** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

initiators = <type 'str[]'>

name = <type 'str'>

virtual_network_id = <type 'int[]'>

virtual_network_tags = <type 'int[]'>

volumes = <type 'int[]'>

class solidfire.models.**CreateVolumeAccessGroupResult**(*volume_access_group_id*, *volume_access_group=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **volume_access_group_id** (*int*) – [required] The ID for the newly-created volume access group.
- **volume_access_group** (VolumeAccessGroup) –

volume_access_group = <class 'solidfire.models.VolumeAccessGroup'>

volume_access_group_id = <type 'int'>

class solidfire.models.**CreateVolumeRequest**(*name*, *account_id*, *total_size*, *enable512e*, *qos=None*, *attributes=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*)

Bases: *solidfire.common.model.DataObject*

CreateVolume enables you to create a new (empty) volume on the cluster. As soon as the volume creation is complete, the volume is available for connection via iSCSI.

Parameters

- **name** (*str*) – [required] The name of the volume access group (might be user specified). Not required to be unique, but recommended. Might be 1 to 64 characters in length.
- **account_id** (*int*) – [required] AccountID for the owner of this volume.
- **total_size** (*int*) – [required] Total size of the volume, in bytes. Size is rounded up to the nearest 1MB size.
- **enable512e** (*bool*) – [required] Specifies whether 512e emulation is enabled or not. Possible values are: true: The volume provides 512-byte sector emulation. false: 512e emulation is not enabled.
- **qos** (QoS) – Initial quality of service settings for this volume. Default values are used if none are specified. Valid settings are: minIOPS maxIOPS burstIOPS You can get the default values for a volume by using the GetDefaultQoS method.
- **attributes** (*dict*) – The list of name-value pairs in JSON object format. Total attribute size must be less than 1000B, or 1KB, including JSON formatting characters.
- **associate_with_qos_policy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.
- **qos_policy_id** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. This parameter is mutually exclusive with the qos parameter.

account_id = <type 'int'>

associate_with_qos_policy = <type 'bool'>

attributes = <type 'dict'>

enable512e = <type 'bool'>

name = <type 'str'>

qos = <class 'solidfire.models.QoS'>

qos_policy_id = <type 'int'>

total_size = <type 'int'>

class solidfire.models.**CreateVolumeResult**(*volume_id*, *curve*, *volume=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **volume** (Volume) –
- **volume_id** (*int*) – [required] VolumeID for the newly created volume.
- **curve** (*dict*) – [required] The curve is a set of key-value pairs. The keys are I/O sizes in bytes. The values represent the cost performing an IOP at a specific I/O size. The curve is calculated relative to a 4096 byte operation set at 100 IOPS.

curve = <type 'dict'>

volume = <class 'solidfire.models.Volume'>

volume_id = <type 'int'>

class solidfire.models.**DeleteAllSupportBundlesResult**(*duration*, *details*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **duration** (*str*) – [required]
- **details** (*dict*) – [required]
- **result** (*str*) – [required]

details = <type 'dict'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**DeleteGroupSnapshotRequest**(*group_snapshot_id*, *save_members*)
Bases: *solidfire.common.model.DataObject*

DeleteGroupSnapshot enables you to delete a group snapshot. You can use the saveMembers parameter to preserve all the snapshots that were made for the volumes in the group, but the group association is removed.

Parameters

- **group_snapshot_id** (*int*) – [required] Specifies the unique ID of the group snapshot.
- **save_members** (*bool*) – [required] Specifies whether to preserve snapshots or delete them. Valid values are: true: Snapshots are preserved, but group association is removed. false: The group and snapshots are deleted.

group_snapshot_id = <type 'int'>

save_members = <type 'bool'>

class solidfire.models.**DeleteGroupSnapshotResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteInitiatorsRequest**(*initiators*)
Bases: *solidfire.common.model.DataObject*

DeleteInitiators enables you to delete one or more initiators from the system (and from any associated volumes or volume access groups). If DeleteInitiators fails to delete one of the initiators provided in the parameter, the system returns an error and does not delete any initiators (no partial completion is possible).

Parameters **initiators** (*int*) – [required] An array of IDs of initiators to delete.

initiators = <type 'int[]'>

class solidfire.models.**DeleteInitiatorsResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteQoSPolicyRequest**(*qos_policy_id*)
Bases: *solidfire.common.model.DataObject*

You can use the DeleteQoSPolicy method to delete a QoS policy from the system. The QoS settings for all volumes created of modified with this policy are unaffected.

Parameters **qos_policy_id** (*int*) – [required] The ID of the QoS policy to be deleted.

qos_policy_id = <type 'int'>

class solidfire.models.**DeleteQoSPolicyResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteSnapshotRequest**(*snapshot_id*)
Bases: *solidfire.common.model.DataObject*

DeleteSnapshot enables you to delete a snapshot. A snapshot that is currently the "active" snapshot cannot be deleted. You must rollback and make another snapshot "active" before the current snapshot can be deleted. For more details on rolling back snapshots, see RollbackToSnapshot.

Parameters **snapshot_id** (*int*) – [required] The ID of the snapshot to be deleted.

snapshot_id = <type 'int'>

class solidfire.models.**DeleteSnapshotResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteStorageContainerResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteStorageContainersRequest**(*storage_container_ids*)
Bases: *solidfire.common.model.DataObject*

DeleteStorageContainers enables you to remove up to 2000 Virtual Volume (VVol) storage containers from the system at one time. The storage containers you remove must not contain any VVols.

Parameters **storage_container_ids** (*UUID*) – [required] A list of IDs of the storage containers to delete. You can specify up to 2000 IDs in the list.

storage_container_ids = <class 'uuid.UUID[]'>

class solidfire.models.**DeleteVolumeAccessGroupRequest**(*volume_access_group_id*, *delete_orphan_initiators=None*, *force=None*)
Bases: *solidfire.common.model.DataObject*

DeleteVolumeAccessGroup enables you to delete a volume access group.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group to be deleted.

- **delete_orphan_initiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.
- **force** (*bool*) – Adding this flag will force the volume access group to be deleted even though it has a Virtual Network ID or Tag. true: Volume access group will be deleted. false: Default. Do not delete the volume access group if it has a Virtual Network ID or Tag.

delete_orphan_initiators = <type 'bool'>

force = <type 'bool'>

volume_access_group_id = <type 'int'>

class solidfire.models.**DeleteVolumeAccessGroupResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DeleteVolumeRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

DeleteVolume marks an active volume for deletion. When marked, the volume is purged (permanently deleted) after the cleanup interval elapses. After making a request to delete a volume, any active iSCSI connections to the volume are immediately terminated and no further connections are allowed while the volume is in this state. A marked volume is not returned in target discovery requests. Any snapshots of a volume that has been marked for deletion are not affected. Snapshots are kept until the volume is purged from the system. If a volume is marked for deletion and has a bulk volume read or bulk volume write operation in progress, the bulk volume read or write operation is stopped. If the volume you delete is paired with a volume, replication between the paired volumes is suspended and no data is transferred to it or from it while in a deleted state. The remote volume that the deleted volume was paired with enters into a PausedMisconfigured state and data is no longer sent to it or from the deleted volume. Until the deleted volume is purged, it can be restored and data transfers resume. If the deleted volume gets purged from the system, the volume it was paired with enters into a StoppedMisconfigured state and the volume pairing status is removed. The purged volume becomes permanently unavailable.

Parameters **volume_id** (*int*) – [required] The ID of the volume to be deleted.

volume_id = <type 'int'>

class solidfire.models.**DeleteVolumeResult**(*volume=None*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume** (Volume) –

volume = <class 'solidfire.models.Volume'>

class solidfire.models.**DeleteVolumesRequest**(*account_ids=None*, *volume_access_group_ids=None*, *volume_ids=None*)
Bases: *solidfire.common.model.DataObject*

DeleteVolumes marks multiple (up to 500) active volumes for deletion. Once marked, the volumes are purged (permanently deleted) after the cleanup interval elapses. The cleanup interval can be set in the SetClusterSettings method. For more information on using this method, see SetClusterSettings on page 1. After making a request to delete volumes, any active iSCSI connections to the volumes are immediately terminated and no further connections are allowed while the volumes are in this state. A marked volume is not returned in target discovery requests. Any snapshots of a volume that has been marked for deletion are not affected. Snapshots are kept until the volume is purged from the system. If a volume is marked for deletion and has a bulk volume read or bulk volume write operation in progress, the bulk volume read or write operation is stopped. If the volumes you delete are paired with a volume, replication between the paired volumes is suspended and no data is transferred to them or from them while in a deleted state. The remote volumes the deleted volumes were paired with enter into a PausedMisconfigured state and data is no longer sent to them or from the deleted volumes. Until the deleted volumes are purged, they can be restored and data transfers resume. If the deleted volumes are purged

from the system, the volumes they were paired with enter into a StoppedMisconfigured state and the volume pairing status is removed. The purged volumes become permanently unavailable.

Parameters

- **account_ids** (*int*) – A list of account IDs. All volumes from these accounts are deleted from the system.
- **volume_access_group_ids** (*int*) – A list of volume access group IDs. All of the volumes from all of the volume access groups you specify in this list are deleted from the system.
- **volume_ids** (*int*) – The list of IDs of the volumes to delete from the system.

account_ids = <type 'int[]'>

volume_access_group_ids = <type 'int[]'>

volume_ids = <type 'int[]'>

class solidfire.models.**DeleteVolumesResult**(*volumes*)
Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] Information about the newly deleted volume.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**DetailedService**(*service*, *node*, *drives*, *drive=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **service** (Service) – [required]
- **node** (Node) – [required]
- **drive** (Drive) –
- **drives** (Drive) – [required]

drive = <class 'solidfire.models.Drive'>

drives = <class 'solidfire.models.Drive[]'>

node = <class 'solidfire.models.Node'>

service = <class 'solidfire.models.Service'>

class solidfire.models.**DisableEncryptionAtRestResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DisableLdapAuthenticationResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**DisableSnmpResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**Drive**(*drive_id*, *node_id*, *async_result_ids*, *capacity*, *serial*, *drive_status*, *drive_type*, *attributes*, *assigned_service=None*, *slot=None*, *reserved_slice_file_capacity=None*, *customer_slice_file_capacity=None*, *smart_ssd_write_capable=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **drive_id** (*int*) – [required] A unique identifier for this drive.

- **node_id** (*int*) – [required] The node this drive is located. If the drive has been physically removed from the node, this is where it was last seen.
- **assigned_service** (*int*) – If this drive is hosting a service, the identifier for that service.
- **async_result_ids** (*int*) – [required] The list of asynchronous jobs currently running on the drive (for example: a secure erase job).
- **capacity** (*int*) – [required] The raw capacity of this drive in bytes.
- **serial** (*str*) – [required] The manufacturer's serial number for this drive.
- **slot** (*int*) – Slot number in the server chassis where this drive is located. If the drive has been physically removed from the node, this will not have a value.
- **drive_status** (*str*) – [required] The current status of this drive.
- **drive_type** (*str*) – [required] The type of this drive.
- **reserved_slice_file_capacity** (*int*) –
- **customer_slice_file_capacity** (*int*) –
- **smart_ssd_write_capable** (*bool*) –
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.

assigned_service = <type 'int'>

async_result_ids = <type 'int[]'>

attributes = <type 'dict'>

capacity = <type 'int'>

customer_slice_file_capacity = <type 'int'>

drive_id = <type 'int'>

drive_status = <type 'str'>

drive_type = <type 'str'>

node_id = <type 'int'>

reserved_slice_file_capacity = <type 'int'>

serial = <type 'str'>

slot = <type 'int'>

smart_ssd_write_capable = <type 'bool'>

class solidfire.models.**DriveConfigInfo**(*canonical_name*, *connected*, *dev*, *dev_path*, *drive_type*, *product*, *name*, *path*, *path_link*, *scsi_compat_id*, *security_enabled*, *security_frozen*, *security_locked*, *security_supported*, *size*, *slot*, *uuid*, *vendor*, *version*, *security_at_maximum*, *serial*, *scsi_state*, *smart_ssd_write_capable=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **canonical_name** (*str*) – [required]
- **connected** (*bool*) – [required]
- **dev** (*int*) – [required]

- **dev_path** (*str*) – [required]
- **drive_type** (*str*) – [required]
- **product** (*str*) – [required]
- **name** (*str*) – [required]
- **path** (*str*) – [required]
- **path_link** (*str*) – [required]
- **scsi_compat_id** (*str*) – [required]
- **smart_ssd_write_capable** (*bool*) –
- **security_enabled** (*bool*) – [required]
- **security_frozen** (*bool*) – [required]
- **security_locked** (*bool*) – [required]
- **security_supported** (*bool*) – [required]
- **size** (*int*) – [required]
- **slot** (*int*) – [required]
- **uuid** (*UUID*) – [required]
- **vendor** (*str*) – [required]
- **version** (*str*) – [required]
- **security_at_maximum** (*bool*) – [required]
- **serial** (*str*) – [required]
- **scsi_state** (*str*) – [required]

canonical_name = <type 'str'>

connected = <type 'bool'>

dev = <type 'int'>

dev_path = <type 'str'>

drive_type = <type 'str'>

name = <type 'str'>

path = <type 'str'>

path_link = <type 'str'>

product = <type 'str'>

scsi_compat_id = <type 'str'>

scsi_state = <type 'str'>

security_at_maximum = <type 'bool'>

security_enabled = <type 'bool'>

security_frozen = <type 'bool'>

security_locked = <type 'bool'>

security_supported = <type 'bool'>

serial = <type 'str'>

size = <type 'int'>

slot = <type 'int'>

smart_ssd_write_capable = <type 'bool'>

uuid = <class 'uuid.UUID'>

vendor = <type 'str'>

version = <type 'str'>

class solidfire.models.**DriveHardware**(*canonical_name*, *connected*, *dev*, *dev_path*, *drive_type*, *life_remaining_percent*, *lifetime_read_bytes*, *lifetime_write_bytes*, *name*, *path*, *path_link*, *power_on_hours*, *product*, *reallocated_sectors*, *reserve_capacity_percent*, *scsi_compat_id*, *scsi_state*, *security_at_maximum*, *security_enabled*, *security_frozen*, *security_locked*, *security_supported*, *serial*, *size*, *slot*, *uuid*, *vendor*, *version*, *smart_ssd_write_capable=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **canonical_name** (*str*) – [required]
- **connected** (*bool*) – [required]
- **dev** (*int*) – [required]
- **dev_path** (*str*) – [required]
- **drive_type** (*str*) – [required]
- **life_remaining_percent** (*int*) – [required]
- **lifetime_read_bytes** (*int*) – [required]
- **lifetime_write_bytes** (*int*) – [required]
- **name** (*str*) – [required]
- **path** (*str*) – [required]
- **path_link** (*str*) – [required]
- **power_on_hours** (*int*) – [required]
- **product** (*str*) – [required]
- **reallocated_sectors** (*int*) – [required]
- **reserve_capacity_percent** (*int*) – [required]
- **scsi_compat_id** (*str*) – [required]
- **scsi_state** (*str*) – [required]
- **security_at_maximum** (*bool*) – [required]
- **security_enabled** (*bool*) – [required]
- **security_frozen** (*bool*) – [required]
- **security_locked** (*bool*) – [required]

- **security_supported** (*bool*) – [required]
- **serial** (*str*) – [required]
- **size** (*int*) – [required]
- **slot** (*int*) – [required]
- **smart_ssd_write_capable** (*bool*) –
- **uuid** (*UUID*) – [required]
- **vendor** (*str*) – [required]
- **version** (*str*) – [required]

```
canonical_name = <type 'str'>
connected = <type 'bool'>
dev = <type 'int'>
dev_path = <type 'str'>
drive_type = <type 'str'>
life_remaining_percent = <type 'int'>
lifetime_read_bytes = <type 'int'>
lifetime_write_bytes = <type 'int'>
name = <type 'str'>
path = <type 'str'>
path_link = <type 'str'>
power_on_hours = <type 'int'>
product = <type 'str'>
reallocated_sectors = <type 'int'>
reserve_capacity_percent = <type 'int'>
scsi_compat_id = <type 'str'>
scsi_state = <type 'str'>
security_at_maximum = <type 'bool'>
security_enabled = <type 'bool'>
security_frozen = <type 'bool'>
security_locked = <type 'bool'>
security_supported = <type 'bool'>
serial = <type 'str'>
size = <type 'int'>
slot = <type 'int'>
smart_ssd_write_capable = <type 'bool'>
uuid = <class 'uuid.UUID'>
vendor = <type 'str'>
```

version = <type 'str'>

class solidfire.models.**DriveHardwareInfo**(*description*, *dev*, *devpath*, *drive_security_at_maximum*, *drive_security_frozen*, *drive_security_locked*, *logicalname*, *product*, *scsi_compat_id*, *security_feature_enabled*, *security_feature_supported*, *serial*, *size*, *uuid*, *version*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **description** (*str*) – [required]
- **dev** (*str*) – [required]
- **devpath** (*str*) – [required]
- **drive_security_at_maximum** (*bool*) – [required]
- **drive_security_frozen** (*bool*) – [required]
- **drive_security_locked** (*bool*) – [required]
- **logicalname** (*str*) – [required]
- **product** (*str*) – [required]
- **scsi_compat_id** (*str*) – [required]
- **security_feature_enabled** (*bool*) – [required]
- **security_feature_supported** (*bool*) – [required]
- **serial** (*str*) – [required]
- **size** (*int*) – [required]
- **uuid** (*UUID*) – [required]
- **version** (*str*) – [required]

description = <type 'str'>

dev = <type 'str'>

devpath = <type 'str'>

drive_security_at_maximum = <type 'bool'>

drive_security_frozen = <type 'bool'>

drive_security_locked = <type 'bool'>

logicalname = <type 'str'>

product = <type 'str'>

scsi_compat_id = <type 'str'>

security_feature_enabled = <type 'bool'>

security_feature_supported = <type 'bool'>

serial = <type 'str'>

size = <type 'int'>

uuid = <class 'uuid.UUID'>

version = <type 'str'>

class solidfire.models.**DriveInfo**(*capacity*, *drive_id*, *node_id*, *serial*, *chassis_slot*, *slot*, *status*, *type*, *attributes*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **capacity** (*int*) – [required] Total capacity of the drive, in bytes.
- **drive_id** (*int*) – [required] DriveID for this drive.
- **node_id** (*int*) – [required] NodeID where this drive is located.
- **serial** (*str*) – [required] Drive serial number.
- **chassis_slot** (*str*) – [required] For HCI platforms, this value is the node letter and slot number in the server chassis where this drive is located. For legacy platforms, the slot number is a string representation of the 'slot' integer.
- **slot** (*int*) – [required] Slot number in the server chassis where this drive is located, or -1 if SATADimm used for internal metadata drive.
- **status** (*str*) – [required]
- **type** (*str*) – [required]
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.

attributes = <type 'dict'>

capacity = <type 'int'>

chassis_slot = <type 'str'>

drive_id = <type 'int'>

node_id = <type 'int'>

serial = <type 'str'>

slot = <type 'int'>

status = <type 'str'>

type = <type 'str'>

class solidfire.models.**DriveStats**(*failed_die_count*, *life_remaining_percent*, *lifetime_read_bytes*, *lifetime_write_bytes*, *power_on_hours*, *read_bytes*, *read_ops*, *reallocated_sectors*, *reserve_capacity_percent*, *timestamp*, *total_capacity*, *used_memory*, *write_bytes*, *write_ops*, *active_sessions=None*, *drive_id=None*, *used_capacity=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **active_sessions** (*int*) –
- **drive_id** (*int*) –
- **failed_die_count** (*int*) – [required]
- **life_remaining_percent** (*int*) – [required]
- **lifetime_read_bytes** (*int*) – [required]

- **lifetime_write_bytes** (*int*) – [required]
- **power_on_hours** (*int*) – [required]
- **read_bytes** (*int*) – [required]
- **read_ops** (*int*) – [required]
- **reallocated_sectors** (*int*) – [required]
- **reserve_capacity_percent** (*int*) – [required]
- **timestamp** (*str*) – [required]
- **total_capacity** (*int*) – [required]
- **used_capacity** (*int*) –
- **used_memory** (*int*) – [required]
- **write_bytes** (*int*) – [required]
- **write_ops** (*int*) – [required]

active_sessions = <type 'int'>

drive_id = <type 'int'>

failed_die_count = <type 'int'>

life_remaining_percent = <type 'int'>

lifetime_read_bytes = <type 'int'>

lifetime_write_bytes = <type 'int'>

power_on_hours = <type 'int'>

read_bytes = <type 'int'>

read_ops = <type 'int'>

reallocated_sectors = <type 'int'>

reserve_capacity_percent = <type 'int'>

timestamp = <type 'str'>

total_capacity = <type 'int'>

used_capacity = <type 'int'>

used_memory = <type 'int'>

write_bytes = <type 'int'>

write_ops = <type 'int'>

class solidfire.models.**DrivesConfigInfo**(*drives*, *num_block_actual*, *num_block_expected*, *num_slice_actual*, *num_slice_expected*, *num_total_actual*, *num_total_expected*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **drives** (DriveConfigInfo) – [required]
- **num_block_actual** (*int*) – [required]
- **num_block_expected** (*int*) – [required]

- **num_slice_actual** (*int*) – [required]
- **num_slice_expected** (*int*) – [required]
- **num_total_actual** (*int*) – [required]
- **num_total_expected** (*int*) – [required]

drives = <class 'solidfire.models.DriveConfigInfo[]'>

num_block_actual = <type 'int'>

num_block_expected = <type 'int'>

num_slice_actual = <type 'int'>

num_slice_expected = <type 'int'>

num_total_actual = <type 'int'>

num_total_expected = <type 'int'>

class solidfire.models.**DrivesHardware**(*drive_hardware*)
Bases: *solidfire.common.model.DataObject*

Parameters **drive_hardware** (DriveHardware) – [required]

drive_hardware = <class 'solidfire.models.DriveHardware[]'>

class solidfire.models.**EnableEncryptionAtRestResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**EnableFeatureRequest**(*feature*)
Bases: *solidfire.common.model.DataObject*

You can use EnableFeature to enable cluster features that are disabled by default.

Parameters **feature** (*str*) – [required] Indicates which feature to enable. Valid value is: vvols: Enable the NetApp SolidFire VVols cluster feature.

feature = <type 'str'>

class solidfire.models.**EnableFeatureResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**EnableLdapAuthenticationRequest**(*server_uris*, *auth_type=None*, *group_search_base_dn=None*, *group_search_custom_filter=None*, *group_search_type=None*, *search_bind_dn=None*, *search_bind_password=None*, *user_dntemplate=None*, *user_search_base_dn=None*, *user_search_filter=None*)
Bases: *solidfire.common.model.DataObject*

The EnableLdapAuthentication method enables you to configure an LDAP directory connection to use for LDAP authentication to a cluster. Users that are members of the LDAP directory can then log in to the storage system using their LDAP credentials.

Parameters

- **auth_type** (*str*) – Identifies which user authentication method to use. Must be one of the following: DirectBind SearchAndBind

- **group_search_base_dn** (*str*) – The base DN of the tree to start the group search (will do a subtree search from here).
- **group_search_custom_filter** (*str*) – For use with the CustomFilter search type, an LDAP filter to use to return the DNs of a users groups. The string can have placeholder text of %USERNAME% and %USERDN% to be replaced with their username and full userDN as needed.
- **group_search_type** (*str*) – Controls the default group search filter used, and must be one of the following: NoGroups: No group support. ActiveDirectory: Nested membership of all of a users AD groups. MemberDN: MemberDN style groups (single level).
- **search_bind_dn** (*str*) – A fully qualified DN to log in with to perform an LDAP search for the user (needs read access to the LDAP directory).
- **search_bind_password** (*str*) – The password for the searchBindDN account used for searching.
- **server_uris** (*str*) – [required] A comma-separated list of LDAP server URIs (examples: "ldap://1.2.3.4" and ldaps://1.2.3.4:123")
- **user_dntemplate** (*str*) – A string that is used to form a fully qualified user DN. The string should have the placeholder text %USERNAME%, which is replaced with the username of the authenticating user.
- **user_search_base_dn** (*str*) – The base DN of the tree to start the search (will do a subtree search from here).
- **user_search_filter** (*str*) – The LDAP filter to use. The string should have the placeholder text %USERNAME% which is replaced with the username of the authenticating user. Example: (&(objectClass=person)(sAMAccountName=%USERNAME%)) will use the sAMAccountName field in Active Directory to match the username entered at cluster login.

auth_type = <type 'str'>

group_search_base_dn = <type 'str'>

group_search_custom_filter = <type 'str'>

group_search_type = <type 'str'>

search_bind_dn = <type 'str'>

search_bind_password = <type 'str'>

server_uris = <type 'str[]'>

user_dntemplate = <type 'str'>

user_search_base_dn = <type 'str'>

user_search_filter = <type 'str'>

class solidfire.models.**EnableLdapAuthenticationResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**EnableSnmpRequest**(*snmp_v3_enabled*)

Bases: *solidfire.common.model.DataObject*

EnableSnmp enables you to enable SNMP on cluster nodes. When you enable SNMP, the action applies to all nodes in the cluster, and the values that are passed replace, in whole, all values set in any previous call to EnableSnmp.

Parameters **snmp_v3_enabled** (*bool*) – [required] If set to "true", then SNMP v3 is enabled on each node in the cluster. If set to "false", then SNMP v2 is enabled.

snmp_v3_enabled = <type 'bool'>

class solidfire.models.**EnableSnmpResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**EventInfo**(*event_id*, *severity*, *event_info_type*, *message*, *service_id*, *node_id*, *drive_id*, *drive_ids*, *time_of_report*, *time_of_publish*, *details=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **event_id** (*int*) – [required]
- **severity** (*int*) – [required]
- **event_info_type** (*str*) – [required]
- **message** (*str*) – [required]
- **service_id** (*int*) – [required]
- **node_id** (*int*) – [required]
- **drive_id** (*int*) – [required]
- **drive_ids** (*int*) – [required]
- **time_of_report** (*str*) – [required]
- **time_of_publish** (*str*) – [required]
- **details** (*str*) –

details = <type 'str'>

drive_id = <type 'int'>

drive_ids = <type 'int[]'>

event_id = <type 'int'>

event_info_type = <type 'str'>

message = <type 'str'>

node_id = <type 'int'>

service_id = <type 'int'>

severity = <type 'int'>

time_of_publish = <type 'str'>

time_of_report = <type 'str'>

class solidfire.models.**FeatureObject**(*enabled*, *feature*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **enabled** (*bool*) – [required] True if the feature is enabled, otherwise false.
- **feature** (*str*) – [required] The name of the feature.

enabled = <type 'bool'>

feature = <type 'str'>

class solidfire.models.**FibreChannelPortInfo**(*firmware*, *hba_port*, *model*, *n_port_id*, *pci_slot*, *serial*, *speed*, *state*, *switch_wwn*, *wwnn*, *wwpn*)

Bases: *solidfire.common.model.DataObject*

Fibre Channel Node Port Info object returns information about all Fibre Channel ports on a node, or for one node in the cluster. The same information is returned for all ports or port information for one node. This information is returned with the API method ListNodeFibreChannelPortInfo (in the SolidFire API Guide).

Parameters

- **firmware** (*str*) – [required] The version of the firmware installed on the Fibre Channel port.
- **hba_port** (*int*) – [required] The ID of the individual HBA port.
- **model** (*str*) – [required] Model of the HBA on the port.
- **n_port_id** (*str*) – [required] Unique SolidFire port node ID.
- **pci_slot** (*int*) – [required] Slot in which the pci card resides on the Fibre Channel node hardware.
- **serial** (*str*) – [required] Serial number on the Fibre Channel port.
- **speed** (*str*) – [required] Speed of the HBA on the port.
- **state** (*str*) – [required] Possible values: UnknownNotPresentOnlineOfflineBlockedBypassedDiagnosti
- **switch_wwn** (*str*) – [required] The World Wide Name of the Fibre Channel switch port.
- **wwnn** (*str*) – [required] World Wide Node Name of the HBA node.
- **wwpn** (*str*) – [required] World Wide Port Name assigned to the physical port of the HBA.

firmware = <type 'str'>

hba_port = <type 'int'>

model = <type 'str'>

n_port_id = <type 'str'>

pci_slot = <type 'int'>

serial = <type 'str'>

speed = <type 'str'>

state = <type 'str'>

switch_wwn = <type 'str'>

wwnn = <type 'str'>

wwpn = <type 'str'>

class solidfire.models.**FibreChannelPortInfoResult**(*result*)

Bases: *solidfire.common.model.DataObject*

Used to return information about the Fibre Channel ports.

Parameters **result** (FibreChannelPortList) – [required] Used to return information about the Fibre Channel ports.

result = <class 'solidfire.models.FibreChannelPortList'>

class solidfire.models.**FibreChannelPortList**(*fibre_channel_ports*)
Bases: *solidfire.common.model.DataObject*

List of all Fibre Channel ports.

Parameters **fibre_channel_ports** (FibreChannelPortInfo) – [required] List of all physical Fibre Channel ports.

fibre_channel_ports = <class 'solidfire.models.FibreChannelPortInfo[]'>

class solidfire.models.**FibreChannelSession**(*initiator_wwpn*, *node_id*, *service_id*, *target_wwpn*, *volume_access_group_id=None*)
Bases: *solidfire.common.model.DataObject*

FibreChannelSession contains information about each Fibre Channel session that is visible to the cluster and what target ports it is visible on.

Parameters

- **initiator_wwpn** (*str*) – [required] The WWPN of the initiator which is logged into the target port.
- **node_id** (*int*) – [required] The node owning the Fibre Channel session.
- **service_id** (*int*) – [required] The service ID of the FService owning this Fibre Channel session
- **target_wwpn** (*str*) – [required] The WWPN of the target port involved in this session.
- **volume_access_group_id** (*int*) – The ID of the volume access group to which the initiatorWWPN beintegers. If not in a volume access group, the value will be null.

initiator_wwpn = <type 'str'>

node_id = <type 'int'>

service_id = <type 'int'>

target_wwpn = <type 'str'>

volume_access_group_id = <type 'int'>

class solidfire.models.**Frequency**(***kwargs*)
Bases: solidfire.custom.models.Frequency

class solidfire.models.**GetAPIResult**(*supported_versions*, *current_version*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **supported_versions** (*float*) – [required]
- **current_version** (*float*) – [required]

current_version = <type 'float'>

supported_versions = <type 'float[]'>

class solidfire.models.**GetAccountByIDRequest**(*account_id*)
Bases: *solidfire.common.model.DataObject*

GetAccountByID enables you to return details about a specific account, given its accountID.

Parameters **account_id** (*int*) – [required] Specifies the account for which details are gathered.

account_id = <type 'int'>

class solidfire.models.**GetAccountByNameRequest**(*username*)
Bases: *solidfire.common.model.DataObject*

GetAccountByName enables you to retrieve details about a specific account, given its username.

Parameters **username** (*str*) – [required] Username for the account.

username = <type 'str'>

class solidfire.models.**GetAccountEfficiencyRequest**(*account_id*)
Bases: *solidfire.common.model.DataObject*

GetAccountEfficiency enables you to retrieve efficiency statistics about a volume account. This method returns efficiency information only for the account you specify as a parameter.

Parameters **account_id** (*int*) – [required] Specifies the volume account for which efficiency statistics are returned.

account_id = <type 'int'>

class solidfire.models.**GetAccountResult**(*account*)
Bases: *solidfire.common.model.DataObject*

Parameters **account** (Account) – [required] Account details.

account = <class 'solidfire.models.Account'>

class solidfire.models.**GetAsyncResultRequest**(*async_handle*, *keep_result=None*)
Bases: *solidfire.common.model.DataObject*

You can use GetAsyncResult to retrieve the result of asynchronous method calls. Some method calls require some time to run, and might not be finished when the system sends the initial response. To obtain the status or result of the method call, use GetAsyncResult to poll the asyncHandle value returned by the method. GetAsyncResult returns the overall status of the operation (in progress, completed, or error) in a standard fashion, but the actual data returned for the operation depends on the original method call and the return data is documented with each method.

Parameters

- **async_handle** (*int*) – [required] A value that was returned from the original asynchronous method call.
- **keep_result** (*bool*) – If true, GetAsyncResult does not remove the asynchronous result upon returning it, enabling future queries to that asyncHandle.

async_handle = <type 'int'>

keep_result = <type 'bool'>

class solidfire.models.**GetBackupTargetRequest**(*backup_target_id*)
Bases: *solidfire.common.model.DataObject*

GetBackupTarget enables you to return information about a specific backup target that you have created.

Parameters **backup_target_id** (*int*) – [required] The unique identifier assigned to the backup target.

backup_target_id = <type 'int'>

class solidfire.models.**GetBackupTargetResult**(*backup_target*)
Bases: *solidfire.common.model.DataObject*

Parameters **backup_target** (BackupTarget) – [required] Object returned for backup target.

backup_target = <class 'solidfire.models.BackupTarget'>

class solidfire.models.**GetBootstrapConfigResult**(*cluster_name*, *node_name*, *nodes*, *version*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster_name** (*str*) – [required] Name of the cluster.
- **node_name** (*str*) – [required] Name of the node.
- **nodes** (NodeWaitingToJoin) – [required] List of descriptions for each node that is actively waiting to join this cluster: compatible - Indicates if the listed node is compatible with the node the API call was executed against. name - IP address of each node. version - version of SolidFire Element software currently installed on the node.
- **version** (*str*) – [required] Version of the SolidFire Element software currently installed.

cluster_name = <type 'str'>

node_name = <type 'str'>

nodes = <class 'solidfire.models.NodeWaitingToJoin[]'>

version = <type 'str'>

class solidfire.models.**GetClusterCapacityResult**(*cluster_capacity*)

Bases: *solidfire.common.model.DataObject*

Parameters **cluster_capacity** (ClusterCapacity) – [required]

cluster_capacity = <class 'solidfire.models.ClusterCapacity'>

class solidfire.models.**GetClusterConfigResult**(*cluster*)

Bases: *solidfire.common.model.DataObject*

Parameters **cluster** (ClusterConfig) – [required] Cluster configuration information the node uses to communicate with the cluster.

cluster = <class 'solidfire.models.ClusterConfig'>

class solidfire.models.**GetClusterFullThresholdResult**(*block_fullness*, *fullness*, *max_metadata_over_provision_factor*, *metadata_fullness*, *slice_reserve_used_threshold_pct*, *stage2_aware_threshold*, *stage2_block_threshold_bytes*, *stage3_block_threshold_bytes*, *stage3_block_threshold_percent*, *stage3_low_threshold*, *stage4_critical_threshold*, *stage4_block_threshold_bytes*, *stage5_block_threshold_bytes*, *sum_total_cluster_bytes*, *sum_total_metadata_cluster_bytes*, *sum_used_cluster_bytes*, *sum_used_metadata_cluster_bytes*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **block_fullness** (*str*) – [required] Current computed level of block fullness of the cluster. Possible values: stage1Happy: No alerts or error conditions. stage2Aware: 3 nodes of capacity available. stage3Low: 2 nodes of capacity available. stage4Critical: 1 node of

capacity available. No new volumes or clones can be created. stage5CompletelyConsumed: Completely consumed. Cluster is read-only, iSCSI connection is maintained but all writes are suspended.

- **fullness** (*str*) – [required] Reflects the highest level of fullness between "blockFullness" and "metadataFullness".
- **max_metadata_over_provision_factor** (*int*) – [required] A value representative of the number of times metadata space can be over provisioned relative to the amount of space available. For example, if there was enough metadata space to store 100 TiB of volumes and this number was set to 5, then 500 TiB worth of volumes could be created.
- **metadata_fullness** (*str*) – [required] Current computed level of metadata fullness of the cluster.
- **slice_reserve_used_threshold_pct** (*int*) – [required] Error condition; message sent to "Alerts" if the reserved slice utilization is greater than the sliceReserveUsedThresholdPct value returned.
- **stage2_aware_threshold** (*int*) – [required] Awareness condition: Value that is set for "Stage 2" cluster threshold level.
- **stage2_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage2 condition will exist.
- **stage3_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage3 condition will exist.
- **stage3_block_threshold_percent** (*int*) – [required] The percent value set for stage3. At this percent full, a warning will be posted in the Alerts log.
- **stage3_low_threshold** (*int*) – [required] Error condition; message sent to "Alerts" that capacity on a cluster is getting low.
- **stage4_critical_threshold** (*int*) – [required] Error condition; message sent to "Alerts" that capacity on a cluster is critically low.
- **stage4_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage4 condition will exist.
- **stage5_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage5 condition will exist.
- **sum_total_cluster_bytes** (*int*) – [required] Physical capacity of the cluster measured in bytes.
- **sum_total_metadata_cluster_bytes** (*int*) – [required] Total amount of space that can be used to store metadata.
- **sum_used_cluster_bytes** (*int*) – [required] Number of bytes used on the cluster.
- **sum_used_metadata_cluster_bytes** (*int*) – [required] Amount of space used on volume drives to store metadata.

```
block_fullness = <type 'str'>

fullness = <type 'str'>

max_metadata_over_provision_factor = <type 'int'>

metadata_fullness = <type 'str'>

slice_reserve_used_threshold_pct = <type 'int'>

stage2_aware_threshold = <type 'int'>
```

stage2_block_threshold_bytes = <type 'int'>

stage3_block_threshold_bytes = <type 'int'>

stage3_block_threshold_percent = <type 'int'>

stage3_low_threshold = <type 'int'>

stage4_block_threshold_bytes = <type 'int'>

stage4_critical_threshold = <type 'int'>

stage5_block_threshold_bytes = <type 'int'>

sum_total_cluster_bytes = <type 'int'>

sum_total_metadata_cluster_bytes = <type 'int'>

sum_used_cluster_bytes = <type 'int'>

sum_used_metadata_cluster_bytes = <type 'int'>

class solidfire.models.**GetClusterHardwareInfoRequest**(*type=None*)
Bases: *solidfire.common.model.DataObject*

You can use the GetClusterHardwareInfo method to retrieve the hardware status and information for all Fibre Channel nodes, iSCSI nodes and drives in the cluster. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information.

Parameters **type** (*str*) – Includes only a certain type of hardware information in the response. Possible values are: drives: List only drive information in the response. nodes: List only node information in the response. all: Include both drive and node information in the response. If this parameter is omitted, a type of "all" is assumed.

type = <type 'str'>

class solidfire.models.**GetClusterHardwareInfoResult**(*cluster_hardware_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_hardware_info** (ClusterHardwareInfo) – [required] Hardware information for all nodes and drives in the cluster. Each object in this output is labeled with the nodeID of the given node.

cluster_hardware_info = <class 'solidfire.models.ClusterHardwareInfo'>

class solidfire.models.**GetClusterInfoResult**(*cluster_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_info** (ClusterInfo) – [required]

cluster_info = <class 'solidfire.models.ClusterInfo'>

class solidfire.models.**GetClusterMasterNodeIDResult**(*node_id*)
Bases: *solidfire.common.model.DataObject*

Parameters **node_id** (*int*) – [required] ID of the master node.

node_id = <type 'int'>

class solidfire.models.**GetClusterStateRequest**(*force*)
Bases: *solidfire.common.model.DataObject*

The GetClusterState API method enables you to indicate if a node is part of a cluster or not. The three states are: Available: Node has not been configured with a cluster name. Pending: Node is pending for a specific named cluster and can be added. Active: Node is an active member of a cluster and may not be added to another cluster. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **force** (*bool*) – [required] To run this command, the force parameter must be set to true.

force = <type 'bool'>

class solidfire.models.**GetClusterStateResult**(*nodes=None*, *cluster=None*, *state=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **nodes** (NodeStateResult) – Array of NodeStateResult objects for each node in the cluster.
- **cluster** (*str*) –
- **state** (*str*) –

cluster = <type 'str'>

nodes = <class 'solidfire.models.NodeStateResult[]'>

state = <type 'str'>

class solidfire.models.**GetClusterStatsResult**(*cluster_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_stats** (ClusterStats) – [required]

cluster_stats = <class 'solidfire.models.ClusterStats'>

class solidfire.models.**GetClusterVersionInfoResult**(*cluster_apiversion*, *cluster_version*, *cluster_version_info*, *software_version_info*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster_apiversion** (*str*) – [required]
- **cluster_version** (*str*) – [required]
- **cluster_version_info** (ClusterVersionInfo) – [required]
- **software_version_info** (SoftwareVersionInfo) – [required]

cluster_apiversion = <type 'str'>

cluster_version = <type 'str'>

cluster_version_info = <class 'solidfire.models.ClusterVersionInfo[]'>

software_version_info = <class 'solidfire.models.SoftwareVersionInfo'>

class solidfire.models.**GetConfigResult**(*config*)
Bases: *solidfire.common.model.DataObject*

Parameters **config** (Config) – [required] The details of the cluster. Values returned in "config": cluster- Cluster information that identifies how the node communicates with the cluster it is associated with. (Object) network - Network information for bonding and Ethernet connections. (Object)

config = <class 'solidfire.models.Config'>

class solidfire.models.**GetCurrentClusterAdminResult**(*cluster_admin*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_admin** (ClusterAdmin) – [required] Information about all cluster and LDAP administrators that exist for a cluster.

cluster_admin = <class 'solidfire.models.ClusterAdmin'>

class solidfire.models.**GetDriveConfigResult**(*drive_config*)
Bases: *solidfire.common.model.DataObject*

Parameters **drive_config** (DrivesConfigInfo) – [required] Configuration information for the drives that are connected to the cluster

drive_config = <class 'solidfire.models.DrivesConfigInfo'>

class solidfire.models.**GetDriveHardwareInfoRequest**(*drive_id*)
Bases: *solidfire.common.model.DataObject*

GetDriveHardwareInfo returns all the hardware information for the given drive. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information.

Parameters **drive_id** (*int*) – [required] DriveID for the drive information requested. You can get DriveIDs by using the ListDrives method.

drive_id = <type 'int'>

class solidfire.models.**GetDriveHardwareInfoResult**(*drive_hardware_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **drive_hardware_info** (DriveHardwareInfo) – [required]

drive_hardware_info = <class 'solidfire.models.DriveHardwareInfo'>

class solidfire.models.**GetDriveStatsRequest**(*drive_id*)
Bases: *solidfire.common.model.DataObject*

GetDriveStats returns high-level activity measurements for a single drive. Values are cumulative from the addition of the drive to the cluster. Some values are specific to block drives. You might not obtain statistical data for both block and metadata drives when you run this method.

Parameters **drive_id** (*int*) – [required] Specifies the drive for which statistics are gathered.

drive_id = <type 'int'>

class solidfire.models.**GetDriveStatsResult**(*drive_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **drive_stats** (DriveStats) – [required]

drive_stats = <class 'solidfire.models.DriveStats'>

class solidfire.models.**GetEfficiencyResult**(*timestamp*, *missing_volumes*, *compression=None*, *deduplication=None*, *thin_provisioning=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **compression** (*float*) – The amount of space being saved by compressing data on a single volume. Stated as a ratio where "1" means data has been stored without being compressed.
- **deduplication** (*float*) – The amount of space being saved on a single volume by not duplicating data. Stated as a ratio.
- **thin_provisioning** (*float*) – The ratio of space used to the amount of space allocated for storing data. Stated as a ratio.
- **timestamp** (*str*) – [required] The last time efficiency data was collected after Garbage Collection (GC). ISO 8601 data string.

- **missing_volumes** (*int*) – [required] The volumes that could not be queried for efficiency data. Missing volumes can be caused by GC being less than hour old, temporary network loss or restarted services since the GC cycle.

compression = <type 'float'>

deduplication = <type 'float'>

missing_volumes = <type 'int[]'>

thin_provisioning = <type 'float'>

timestamp = <type 'str'>

class solidfire.models.**GetFeatureStatusRequest**(*feature=None*)
Bases: *solidfire.common.model.DataObject*

GetFeatureStatus enables you to retrieve the status of a cluster feature.

Parameters **feature** (*str*) – Specifies the feature for which the status is returned. Valid value is: vvols: Retrieve status for the NetApp SolidFire VVols cluster feature.

feature = <type 'str'>

class solidfire.models.**GetFeatureStatusResult**(*features*)
Bases: *solidfire.common.model.DataObject*

Parameters **features** (FeatureObject) – [required] An array of feature objects indicating the feature name and its status.

features = <class 'solidfire.models.FeatureObject[]'>

class solidfire.models.**GetHardwareConfigResult**(*hardware_config*)
Bases: *solidfire.common.model.DataObject*

Parameters **hardware_config** (*dict*) – [required] List of hardware information and current settings.

hardware_config = <type 'dict'>

class solidfire.models.**GetHardwareInfoResult**(*hardware_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **hardware_info** (*dict*) – [required] Hardware information for this node.

hardware_info = <type 'dict'>

class solidfire.models.**GetIpmiConfigNodesResult**(*node_id*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) – [required]
- **result** (*dict*) – [required]

node_id = <type 'int'>

result = <type 'dict'>

class solidfire.models.**GetIpmiConfigRequest**(*chassis_type=None*)
Bases: *solidfire.common.model.DataObject*

GetIpmiConfig enables you to retrieve hardware sensor information from sensors that are in your node.

Parameters **chassis_type** (*str*) – Displays information for each node chassis type. Valid values are: all: Returns sensor information for each chassis type. {chassis type}: Returns sensor information for a specified chassis type.

chassis_type = <type 'str'>

class solidfire.models.**GetIpmiConfigResult**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (GetIpmiConfigNodesResult) – [required]

nodes = <class 'solidfire.models.GetIpmiConfigNodesResult[]'>

class solidfire.models.**GetIpmiInfoNodesResult**(*node_id*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) – [required]
- **result** (GetIpmiInfoNodesResultObject) – [required]

node_id = <type 'int'>

result = <class 'solidfire.models.GetIpmiInfoNodesResultObject'>

class solidfire.models.**GetIpmiInfoNodesResultObject**(*ipmi_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **ipmi_info** (IpmiInfo) – [required]

ipmi_info = <class 'solidfire.models.IpmiInfo'>

class solidfire.models.**GetIpmiInfoResult**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (GetIpmiInfoNodesResult) – [required] Detailed information from each sensor within a node.

nodes = <class 'solidfire.models.GetIpmiInfoNodesResult[]'>

class solidfire.models.**GetLdapConfigurationResult**(*ldap_configuration*)
Bases: *solidfire.common.model.DataObject*

Parameters **ldap_configuration** (LdapConfiguration) – [required] List of the current LDAP configuration settings. This API call will not return the plain text of the search account password. Note: If LDAP authentication is currently disabled, all the returned settings will be empty with the exception of "authType", and "groupSearchType" which are set to "SearchAndBind" and "ActiveDirectory" respectively.

ldap_configuration = <class 'solidfire.models.LdapConfiguration'>

class solidfire.models.**GetLimitsResult**(*account_count_max*, *account_name_length_max*, *account_name_length_min*, *bulk_volume_jobs_per_node_max*, *bulk_volume_jobs_per_volume_max*, *clone_jobs_per_volume_max*, *cluster_pairs_count_max*, *initiator_name_length_max*, *initiator_count_max*, *initiators_per_volume_access_group_count_max*, *iscsi_sessions_from_fibre_channel_nodes_max*, *qos_policy_count_max*, *secret_length_max*, *schedule_name_length_max*, *secret_length_min*, *snapshot_name_length_max*, *snapshots_per_volume_max*, *volume_access_group_count_max*, *volume_access_group_lun_max*, *volume_access_group_name_length_max*, *volume_access_group_name_length_min*, *volume_access_groups_per_initiator_count_max*, *volume_access_groups_per_volume_count_max*, *initiator_alias_length_max*, *volume_burst_iopsmax*, *volume_burst_iopsmin*, *volume_count_max*, *volume_max_iopsmax*, *volume_max_iopsmin*, *volume_min_iopsmax*, *volume_min_iopsmin*, *volume_name_length_max*, *volume_name_length_min*, *volume_size_max*, *volume_size_min*, *volumes_per_account_count_max*, *volumes_per_group_snapshot_max*, *volumes_per_volume_access_group_count_max*, *cluster_admin_account_max=None*, *fibre_channel_volume_access_max=None*, *virtual_volumes_per_account_count_max=None*, *virtual_volume_count_max=None*)

Bases: *solidfire.common.model.DataObject*

Limits for the cluster

Parameters

- **account_count_max** (*int*) – [required]
- **account_name_length_max** (*int*) – [required]
- **account_name_length_min** (*int*) – [required]
- **bulk_volume_jobs_per_node_max** (*int*) – [required]
- **bulk_volume_jobs_per_volume_max** (*int*) – [required]
- **clone_jobs_per_volume_max** (*int*) – [required]
- **cluster_pairs_count_max** (*int*) – [required]
- **initiator_name_length_max** (*int*) – [required]
- **initiator_count_max** (*int*) – [required]
- **initiators_per_volume_access_group_count_max** (*int*) – [required]
- **iscsi_sessions_from_fibre_channel_nodes_max** (*int*) – [required]
- **qos_policy_count_max** (*int*) – [required]

- **secret_length_max** (*int*) – [required]
- **schedule_name_length_max** (*int*) – [required]
- **secret_length_min** (*int*) – [required]
- **snapshot_name_length_max** (*int*) – [required]
- **snapshots_per_volume_max** (*int*) – [required]
- **volume_access_group_count_max** (*int*) – [required]
- **volume_access_group_lun_max** (*int*) – [required]
- **volume_access_group_name_length_max** (*int*) – [required]
- **volume_access_group_name_length_min** (*int*) – [required]
- **volume_access_groups_per_initiator_count_max** (*int*) – [required]
- **volume_access_groups_per_volume_count_max** (*int*) – [required]
- **initiator_alias_length_max** (*int*) – [required]
- **volume_burst_iopsmax** (*int*) – [required]
- **volume_burst_iopsmin** (*int*) – [required]
- **volume_count_max** (*int*) – [required]
- **volume_max_iopsmax** (*int*) – [required]
- **volume_max_iopsmin** (*int*) – [required]
- **volume_min_iopsmax** (*int*) – [required]
- **volume_min_iopsmin** (*int*) – [required]
- **volume_name_length_max** (*int*) – [required]
- **volume_name_length_min** (*int*) – [required]
- **volume_size_max** (*int*) – [required]
- **volume_size_min** (*int*) – [required]
- **volumes_per_account_count_max** (*int*) – [required]
- **volumes_per_group_snapshot_max** (*int*) – [required]
- **volumes_per_volume_access_group_count_max** (*int*) – [required]
- **cluster_admin_account_max** (*int*) –
- **fibre_channel_volume_access_max** (*int*) –
- **virtual_volumes_per_account_count_max** (*int*) –
- **virtual_volume_count_max** (*int*) –

account_count_max = <type 'int'>

account_name_length_max = <type 'int'>

account_name_length_min = <type 'int'>

bulk_volume_jobs_per_node_max = <type 'int'>

bulk_volume_jobs_per_volume_max = <type 'int'>

clone_jobs_per_volume_max = <type 'int'>

```
cluster_admin_account_max = <type 'int'>
cluster_pairs_count_max = <type 'int'>
fibre_channel_volume_access_max = <type 'int'>
initiator_alias_length_max = <type 'int'>
initiator_count_max = <type 'int'>
initiator_name_length_max = <type 'int'>
initiators_per_volume_access_group_count_max = <type 'int'>
iscsi_sessions_from_fibre_channel_nodes_max = <type 'int'>
qos_policy_count_max = <type 'int'>
schedule_name_length_max = <type 'int'>
secret_length_max = <type 'int'>
secret_length_min = <type 'int'>
snapshot_name_length_max = <type 'int'>
snapshots_per_volume_max = <type 'int'>
virtual_volume_count_max = <type 'int'>
virtual_volumes_per_account_count_max = <type 'int'>
volume_access_group_count_max = <type 'int'>
volume_access_group_lun_max = <type 'int'>
volume_access_group_name_length_max = <type 'int'>
volume_access_group_name_length_min = <type 'int'>
volume_access_groups_per_initiator_count_max = <type 'int'>
volume_access_groups_per_volume_count_max = <type 'int'>
volume_burst_iopsmax = <type 'int'>
volume_burst_iopsmin = <type 'int'>
volume_count_max = <type 'int'>
volume_max_iopsmax = <type 'int'>
volume_max_iopsmin = <type 'int'>
volume_min_iopsmax = <type 'int'>
volume_min_iopsmin = <type 'int'>
volume_name_length_max = <type 'int'>
volume_name_length_min = <type 'int'>
volume_size_max = <type 'int'>
volume_size_min = <type 'int'>
volumes_per_account_count_max = <type 'int'>
volumes_per_group_snapshot_max = <type 'int'>
volumes_per_volume_access_group_count_max = <type 'int'>
```

class solidfire.models.**GetLoginBannerResult**(*login_banner*)
Bases: *solidfire.common.model.DataObject*

Parameters **login_banner** (LoginBanner) – [required]

login_banner = <class 'solidfire.models.LoginBanner'>

class solidfire.models.**GetLoginSessionInfoResult**(*login_session_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **login_session_info** (LoginSessionInfo) – [required] The authentication expiration period. Formatted in H:mm:ss. For example: 1:30:00, 20:00, 5:00. All leading zeros and colons are removed regardless of the format the timeout was entered. Objects returned are: timeout - The time, in minutes, when this session will timeout and expire.

login_session_info = <class 'solidfire.models.LoginSessionInfo'>

class solidfire.models.**GetNetworkConfigResult**(*network*)
Bases: *solidfire.common.model.DataObject*

Parameters **network** (Network) – [required]

network = <class 'solidfire.models.Network'>

class solidfire.models.**GetNodeHardwareInfoRequest**(*node_id*)
Bases: *solidfire.common.model.DataObject*

GetNodeHardwareInfo enables you to return all the hardware information and status for the node specified. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information.

Parameters **node_id** (*int*) – [required] The ID of the node for which hardware information is being requested. Information about a Fibre Channel node is returned if a Fibre Channel node is specified.

node_id = <type 'int'>

class solidfire.models.**GetNodeHardwareInfoResult**(*node_hardware_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **node_hardware_info** (*dict*) – [required] Hardware information for the specified nodeID.

node_hardware_info = <type 'dict'>

class solidfire.models.**GetNodeSSLCertificateResult**(*certificate*, *details*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **certificate** (*str*) – [required] The full PEM-encoded test of the certificate.
- **details** (*dict*) – [required] The decoded information of the certificate.

certificate = <type 'str'>

details = <type 'dict'>

class solidfire.models.**GetNodeStatsRequest**(*node_id*)
Bases: *solidfire.common.model.DataObject*

GetNodeStats enables you to retrieve the high-level activity measurements for a single node.

Parameters **node_id** (*int*) – [required] Specifies the node for which statistics are gathered.

node_id = <type 'int'>

class solidfire.models.**GetNodeStatsResult**(*node_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **node_stats** (NodeStatsInfo) – [required] Node activity information.

node_stats = <class 'solidfire.models.NodeStatsInfo'>

class solidfire.models.**GetNtpInfoResult**(*broadcastclient*, *servers*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **broadcastclient** (*bool*) – [required] Indicates whether or not the nodes in the cluster are listening for broadcast NTP messages. Possible values: true false
- **servers** (*str*) – [required] List of NTP servers.

broadcastclient = <type 'bool'>

servers = <type 'str[]'>

class solidfire.models.**GetNvramInfoRequest**(*force=None*)
Bases: *solidfire.common.model.DataObject*

GetNvramInfo enables you to retrieve information from each node about the NVRAM card.

Parameters **force** (*bool*) – Required parameter to successfully run on all nodes in the cluster.

force = <type 'bool'>

class solidfire.models.**GetNvramInfoResult**(*nvram_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **nvram_info** (*dict*) – [required] Arrays of events and errors detected on the NVRAM card.

nvram_info = <type 'dict'>

class solidfire.models.**GetOriginNode**(*node_id*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) – [required]
- **result** (GetOriginNodeResult) – [required]

node_id = <type 'int'>

result = <class 'solidfire.models.GetOriginNodeResult'>

class solidfire.models.**GetOriginNodeResult**(*origin=None*)
Bases: *solidfire.common.model.DataObject*

Parameters **origin** (Origin) –

origin = <class 'solidfire.models.Origin'>

class solidfire.models.**GetOriginResult**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (GetOriginNode) – [required]

nodes = <class 'solidfire.models.GetOriginNode[]'>

class solidfire.models.**GetPendingOperationResult**(*pending_operation*)
Bases: *solidfire.common.model.DataObject*

Parameters **pending_operation** (PendingOperation) – [required]

pending_operation = <class 'solidfire.models.PendingOperation'>

class solidfire.models.**GetQoSPolicyRequest**(*qos_policy_id*)
Bases: *solidfire.common.model.DataObject*

You can use the GetQoSPolicy method to get details about a specific QoSPolicy from the system.

Parameters **qos_policy_id** (*int*) – [required] The ID of the policy to be retrieved.

qos_policy_id = <type 'int'>

class solidfire.models.**GetQoSPolicyResult**(*qos_policy*)
Bases: *solidfire.common.model.DataObject*

Parameters **qos_policy** (QoSPolicy) – [required] Details of the requested QoS policy.

qos_policy = <class 'solidfire.models.QoSPolicy'>

class solidfire.models.**GetRemoteLoggingHostsResult**(*remote_hosts*)
Bases: *solidfire.common.model.DataObject*

Parameters **remote_hosts** (LoggingServer) – [required] List of hosts to forward logging information to.

remote_hosts = <class 'solidfire.models.LoggingServer[]'>

class solidfire.models.**GetSSLCertificateResult**(*certificate*, *details*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **certificate** (*str*) – [required] The full PEM-encoded test of the certificate.
- **details** (*dict*) – [required] The decoded information of the certificate.

certificate = <type 'str'>

details = <type 'dict'>

class solidfire.models.**GetScheduleRequest**(*schedule_id*)
Bases: *solidfire.common.model.DataObject*

You can use the GetSchedule method to retrieve information about a scheduled snapshot. You can see information about a specific schedule if there are many snapshot schedules in the system. You also retrieve information about more than one schedule with this method by specifying additional scheduleIDs in the parameter.

Parameters **schedule_id** (*int*) – [required] Specifies the unique ID of the schedule or multiple schedules to display.

schedule_id = <type 'int'>

class solidfire.models.**GetScheduleResult**(*schedule*)
Bases: *solidfire.common.model.DataObject*

Parameters **schedule** (Schedule) – [required] The schedule attributes.

schedule = <class 'solidfire.models.Schedule'>

class solidfire.models.**GetSnmpACLResult**(*networks=None*, *usm_users=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **networks** (SnmpNetwork) – List of networks and what type of access they have to the SNMP servers running on the cluster nodes. Present if SNMP v3 is disabled.

- **usm_users** (SnmpV3UsmUser) – List of users and the type of access they have to the SNMP servers running on the cluster nodes. Present if SNMP v3 is enabled.

networks = <class 'solidfire.models.SnmpNetwork[]'>

usm_users = <class 'solidfire.models.SnmpV3UsmUser[]'>

class solidfire.models.**GetSnmpInfoResult**(*enabled*, *snmp_v3_enabled*, *networks=None*, *usm_users=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **networks** (SnmpNetwork) – List of networks and access types enabled for SNMP. Note: "networks" will only be present if SNMP V3 is disabled.
- **enabled** (*bool*) – [required] If the nodes in the cluster are configured for SNMP.
- **snmp_v3_enabled** (*bool*) – [required] If the nodes in the cluster are configured for SNMP v3.
- **usm_users** (SnmpV3UsmUser) – If SNMP v3 is enabled, the values returned is a list of user access parameters for SNMP information from the cluster. This will be returned instead of the "networks" parameter.

enabled = <type 'bool'>

networks = <class 'solidfire.models.SnmpNetwork[]'>

snmp_v3_enabled = <type 'bool'>

usm_users = <class 'solidfire.models.SnmpV3UsmUser[]'>

class solidfire.models.**GetSnmpStateResult**(*enabled*, *snmp_v3_enabled*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **enabled** (*bool*) – [required] If the nodes in the cluster are configured for SNMP.
- **snmp_v3_enabled** (*bool*) – [required] If the node in the cluster is configured for SNMP v3.

enabled = <type 'bool'>

snmp_v3_enabled = <type 'bool'>

class solidfire.models.**GetSnmpTrapInfoResult**(*trap_recipients*, *cluster_fault_traps_enabled*, *cluster_fault_resolved_traps_enabled*, *cluster_event_traps_enabled*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **trap_recipients** (SnmpTrapRecipient) – [required] List of hosts that are to receive the traps generated by the cluster.
- **cluster_fault_traps_enabled** (*bool*) – [required] If "true", when a cluster fault is logged a corresponding solidFireClusterFaultNotification is sent to the configured list of trap recipients.
- **cluster_fault_resolved_traps_enabled** (*bool*) – [required] If "true", when a cluster fault is logged a corresponding solidFireClusterFaultResolvedNotification is sent to the configured list of trap recipients.

- **cluster_event_traps_enabled** (*bool*) – [required] If "true", when a cluster fault is logged a corresponding solidFireClusterEventNotification is sent to the configured list of trap recipients.

cluster_event_traps_enabled = <type 'bool'>

cluster_fault_resolved_traps_enabled = <type 'bool'>

cluster_fault_traps_enabled = <type 'bool'>

trap_recipients = <class 'solidfire.models.SnmpTrapRecipient[]'>

class solidfire.models.**GetStorageContainerEfficiencyRequest**(*storage_container_id*)
Bases: *solidfire.common.model.DataObject*

GetStorageContainerEfficiency enables you to retrieve efficiency information about a virtual volume storage container.

Parameters **storage_container_id** (*UUID*) – [required] The ID of the storage container for which to retrieve efficiency information.

storage_container_id = <class 'uuid.UUID'>

class solidfire.models.**GetStorageContainerEfficiencyResult**(*compression*, *deduplication*, *missing_volumes*, *thin_provisioning*, *timestamp*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **compression** (*float*) – [required]
- **deduplication** (*float*) – [required]
- **missing_volumes** (*int*) – [required] The volumes that could not be queried for efficiency data. Missing volumes can be caused by the Garbage Collection (GC) cycle being less than an hour old, temporary loss of network connectivity, or restarted services since the GC cycle.
- **thin_provisioning** (*float*) – [required]
- **timestamp** (*str*) – [required] The last time efficiency data was collected after Garbage Collection (GC).

compression = <type 'float'>

deduplication = <type 'float'>

missing_volumes = <type 'int[]'>

thin_provisioning = <type 'float'>

timestamp = <type 'str'>

class solidfire.models.**GetSystemStatusResult**(*reboot_required*)
Bases: *solidfire.common.model.DataObject*

Parameters **reboot_required** (*bool*) – [required]

reboot_required = <type 'bool'>

class solidfire.models.**GetVirtualVolumeCountResult**(*count*)
Bases: *solidfire.common.model.DataObject*

Parameters **count** (*int*) – [required] The number of virtual volumes currently in the system.

count = <type 'int'>

class solidfire.models.**GetVolumeAccessGroupEfficiencyRequest**(*volume_access_group_id*)
Bases: *solidfire.common.model.DataObject*

GetVolumeAccessGroupEfficiency enables you to retrieve efficiency information about a volume access group. Only the volume access group you provide as the parameter in this API method is used to compute the capacity.

Parameters **volume_access_group_id** (*int*) – [required] The volume access group for which capacity is computed.

volume_access_group_id = <type 'int'>

class solidfire.models.**GetVolumeAccessGroupLunAssignmentsRequest**(*volume_access_group_id*)
Bases: *solidfire.common.model.DataObject*

The GetVolumeAccessGroupLunAssignments method enables you to retrieve details on LUN mappings of a specified volume access group.

Parameters **volume_access_group_id** (*int*) – [required] The unique volume access group ID used to return information.

volume_access_group_id = <type 'int'>

class solidfire.models.**GetVolumeAccessGroupLunAssignmentsResult**(*volume_access_group_lun_assignments*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_access_group_lun_assignments** (VolumeAccessGroupLunAssignments) – [required] List of all physical Fibre Channel ports, or a port for a single node.

volume_access_group_lun_assignments = <class 'solidfire.models.VolumeAccessGroupLunAss

class solidfire.models.**GetVolumeCountResult**(*count*)
Bases: *solidfire.common.model.DataObject*

Parameters **count** (*int*) – [required] The number of volumes currently in the system.

count = <type 'int'>

class solidfire.models.**GetVolumeEfficiencyRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

GetVolumeEfficiency enables you to retrieve information about a volume. Only the volume you give as a parameter in this API method is used to compute the capacity.

Parameters **volume_id** (*int*) – [required] Specifies the volume for which capacity is computed.

volume_id = <type 'int'>

class solidfire.models.**GetVolumeEfficiencyResult**(*deduplication*, *missing_volumes*, *thin_provisioning*, *timestamp*, *compression=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **compression** (*float*) – The amount of space being saved by compressing data on a single volume. Stated as a ratio where "1" means data has been stored without being compressed.
- **deduplication** (*float*) – [required] The amount of space being saved on a single volume by not duplicating data. Stated as a ratio.
- **missing_volumes** (*int*) – [required] The volumes that could not be queried for efficiency data. Missing volumes can be caused by GC being less than hour old, temporary network loss or restarted services since the GC cycle.

- **thin_provisioning** (*float*) – [required] The ratio of space used to the amount of space allocated for storing data. Stated as a ratio.
- **timestamp** (*str*) – [required] The last time efficiency data was collected after Garbage Collection (GC).

compression = <type 'float'>

deduplication = <type 'float'>

missing_volumes = <type 'int[]'>

thin_provisioning = <type 'float'>

timestamp = <type 'str'>

class solidfire.models.**GetVolumeStatsRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

GetVolumeStats enables you to retrieve high-level activity measurements for a single volume. Values are cumulative from the creation of the volume.

Parameters **volume_id** (*int*) – [required] Specifies the volume for which statistics are gathered.

volume_id = <type 'int'>

class solidfire.models.**GetVolumeStatsResult**(*volume_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VolumeStats) – [required] Volume activity information.

volume_stats = <class 'solidfire.models.VolumeStats'>

class solidfire.models.**GroupCloneVolumeMember**(*volume_id*, *src_volume_id*)
Bases: *solidfire.common.model.DataObject*

Represents the relationship between the source Volume and cloned Volume IDs.

Parameters

- **volume_id** (*int*) – [required] The VolumeID of the cloned volume.
- **src_volume_id** (*int*) – [required] The VolumeID of the source volume.

src_volume_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**GroupSnapshot**(*group_snapshot_id*, *group_snapshot_uuid*, *members*, *name*, *create_time*, *status*, *attributes*)
Bases: *solidfire.common.model.DataObject*

Group Snapshot object represents a point-in-time copy of a group of volumes.

Parameters

- **group_snapshot_id** (*int*) – [required] Unique ID of the new group snapshot.
- **group_snapshot_uuid** (*UUID*) – [required] UUID of the group snapshot.
- **members** (GroupSnapshotMembers) – [required] List of volumeIDs and snapshotIDs for each member of the group.
- **name** (*str*) – [required] Name of the group snapshot, or, if none was given, the UTC formatted day and time on which the snapshot was created.
- **create_time** (*str*) – [required] The UTC formatted day and time on which the snapshot was created.

- **status** (*str*) – [required] Status of the snapshot. Possible values: Preparing: A snapshot that is being prepared for use and is not yet writable. Done: A snapshot that has finished being prepared and is now usable
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.

attributes = <type 'dict'>

create_time = <type 'str'>

group_snapshot_id = <type 'int'>

group_snapshot_uuid = <class 'uuid.UUID'>

members = <class 'solidfire.models.GroupSnapshotMembers[]'>

name = <type 'str'>

status = <type 'str'>

class solidfire.models.**GroupSnapshotMembers**(*volume_id*, *snapshot_id*, *snapshot_uuid*, *checksum*, *attributes=None*, *create_time=None*, *enable_remote_replication=None*, *expiration_reason=None*, *expiration_time=None*, *group_id=None*, *group_snapshot_uuid=None*, *name=None*, *remote_status=None*, *remote_statuses=None*, *status=None*, *total_size=None*, *virtual_volume_id=None*, *volume_pair_uuid=None*)

Bases: *solidfire.common.model.DataObject*

List of checksum, volumeIDs and snapshotIDs for each member of the group.

Parameters

- **volume_id** (*int*) – [required] The source volume ID for the snapshot.
- **snapshot_id** (*int*) – [required] Unique ID of a snapshot from which the new snapshot is made. The snapshotID passed must be a snapshot on the given volume.
- **snapshot_uuid** (*UUID*) – [required] Universal Unique ID of an existing snapshot.
- **checksum** (*str*) – [required] A string that represents the correct digits in the stored snapshot. This checksum can be used later to compare other snapshots to detect errors in the data.
- **attributes** (*dict*) –
- **create_time** (*str*) –
- **enable_remote_replication** (*bool*) –
- **expiration_reason** (*str*) –
- **expiration_time** (*str*) –
- **group_id** (*int*) –
- **group_snapshot_uuid** (*UUID*) –
- **name** (*str*) –
- **remote_status** (*str*) –
- **remote_statuses** (*dict*) –

- **status** (*str*) –
- **total_size** (*int*) –
- **virtual_volume_id** (*int*) –
- **volume_pair_uuid** (*UUID*) –

attributes = <type 'dict'>

checksum = <type 'str'>

create_time = <type 'str'>

enable_remote_replication = <type 'bool'>

expiration_reason = <type 'str'>

expiration_time = <type 'str'>

group_id = <type 'int'>

group_snapshot_uuid = <class 'uuid.UUID'>

name = <type 'str'>

remote_status = <type 'str'>

remote_statuses = <type 'dict[]'>

snapshot_id = <type 'int'>

snapshot_uuid = <class 'uuid.UUID'>

status = <type 'str'>

total_size = <type 'int'>

virtual_volume_id = <type 'int'>

volume_id = <type 'int'>

volume_pair_uuid = <class 'uuid.UUID'>

class solidfire.models.**ISCSISession**(*account_id*, *account_name*, *drive_id*, *initiator_ip*, *initiator_port_name*, *target_port_name*, *initiator_name*, *node_id*, *service_id*, *session_id*, *target_name*, *target_ip*, *virtual_network_id*, *volume_id*, *create_time*, *volume_instance*, *initiator_session_id*, *drive_ids=None*, *initiator=None*, *ms_since_last_scsi_command=None*, *ms_since_last_iscsi_pdu=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **drive_ids** (*int*) –
- **account_id** (*int*) – [required]
- **initiator** (*Initiator*) –
- **account_name** (*str*) – [required]
- **drive_id** (*int*) – [required]
- **initiator_ip** (*str*) – [required]
- **initiator_port_name** (*str*) – [required]
- **target_port_name** (*str*) – [required]

- **initiator_name** (*str*) – [required]
- **node_id** (*int*) – [required]
- **service_id** (*int*) – [required]
- **session_id** (*int*) – [required]
- **target_name** (*str*) – [required]
- **target_ip** (*str*) – [required]
- **virtual_network_id** (*int*) – [required]
- **volume_id** (*int*) – [required]
- **create_time** (*str*) – [required]
- **volume_instance** (*int*) – [required]
- **initiator_session_id** (*int*) – [required]
- **ms_since_last_scsi_command** (*int*) –
- **ms_since_last_iscsi_pdu** (*int*) –

account_id = <type 'int'>

account_name = <type 'str'>

create_time = <type 'str'>

drive_id = <type 'int'>

drive_ids = <type 'int[]'>

initiator = <class 'solidfire.models.Initiator'>

initiator_ip = <type 'str'>

initiator_name = <type 'str'>

initiator_port_name = <type 'str'>

initiator_session_id = <type 'int'>

ms_since_last_iscsi_pdu = <type 'int'>

ms_since_last_scsi_command = <type 'int'>

node_id = <type 'int'>

service_id = <type 'int'>

session_id = <type 'int'>

target_ip = <type 'str'>

target_name = <type 'str'>

target_port_name = <type 'str'>

virtual_network_id = <type 'int'>

volume_id = <type 'int'>

volume_instance = <type 'int'>

class solidfire.models.**Initiator**(*alias*, *initiator_id*, *initiator_name*, *volume_access_groups*, *attributes*)

Bases: *solidfire.common.model.DataObject*

Object containing characteristics of each initiator

Parameters

- **alias** (*str*) – [required] The friendly name assigned to this initiator. (String)
- **initiator_id** (*int*) – [required] The numeric ID of the initiator that has been created. (Integer)
- **initiator_name** (*str*) – [required] The name of the initiator that has been created. (String)
- **volume_access_groups** (*int*) – [required] A list of volumeAccessGroupIDs to which this initiator beintegers. (Array of Integers)
- **attributes** (*dict*) – [required] A set of JSON attributes assigned to this initiator. (JSON Object)

alias = <type 'str'>

attributes = <type 'dict'>

initiator_id = <type 'int'>

initiator_name = <type 'str'>

volume_access_groups = <type 'int[]'>

class solidfire.models.**InvokeSFApiRequest**(*method*, *parameters=None*)

Bases: *solidfire.common.model.DataObject*

This will invoke any API method supported by the SolidFire API for the version and port the connection is using. Returns a nested hashtable of key/value pairs that contain the result of the invoked method.

Parameters

- **method** (*str*) – [required] The name of the method to invoke. This is case sensitive.
- **parameters** (*str*) – An object, normally a dictionary or hashtable of the key/value pairs, to be passed as the params for the method being invoked.

method = <type 'str'>

parameters = <type 'str'>

class solidfire.models.**IpmiInfo**(*sensors*)

Bases: *solidfire.common.model.DataObject*

Parameters **sensors** (*dict*) – [required]

sensors = <type 'dict[]'>

class solidfire.models.**LdapConfiguration**(*auth_type*, *enabled*, *group_search_base_dn*, *group_search_custom_filter*, *group_search_type*, *search_bind_dn*, *server_uris*, *user_dntemplate*, *user_search_base_dn*, *user_search_filter*)

Bases: *solidfire.common.model.DataObject*

LDAP Configuration object returns information about the LDAP configuration on SolidFire storage. LDAP information is returned with the API method GetLdapConfiguration.

Parameters

- **auth_type** (*str*) – [required] Identifies which user authentcation method will be used. Valid values: DirectBind SearchAndBind
- **enabled** (*bool*) – [required] Identifies whether or not the system is enabled for LDAP. Valid values: true false
- **group_search_base_dn** (*str*) – [required] The base DN of the tree to start the group search (will do a subtree search from here).
- **group_search_custom_filter** (*str*) – [required] The custom search filter used.
- **group_search_type** (*str*) – [required] Controls the default group search filter used, can be one of the following: NoGroups: No group support. ActiveDirectory: Nested membership of all of a user's AD groups. MemberDN: MemberDN style groups (single-level).
- **search_bind_dn** (*str*) – [required] A fully qualified DN to log in with to perform an LDAP search for the user (needs read access to the LDAP directory).
- **server_uris** (*str*) – [required] A comma-separated list of LDAP server URIs (examples: "ldap://1.2.3.4" and ldaps://1.2.3.4:123")
- **user_dntemplate** (*str*) – [required] A string that is used to form a fully qualified user DN.
- **user_search_base_dn** (*str*) – [required] The base DN of the tree used to start the search (will do a subtree search from here).
- **user_search_filter** (*str*) – [required] The LDAP filter used.

auth_type = <type 'str'>

enabled = <type 'bool'>

group_search_base_dn = <type 'str'>

group_search_custom_filter = <type 'str'>

group_search_type = <type 'str'>

search_bind_dn = <type 'str'>

server_uris = <type 'str[]'>

user_dntemplate = <type 'str'>

user_search_base_dn = <type 'str'>

user_search_filter = <type 'str'>

class solidfire.models.**ListAccountsRequest**(*start_account_id=None*, *limit=None*, *include_storage_containers=None*)

Bases: *solidfire.common.model.DataObject*

ListAccounts returns the entire list of accounts, with optional paging support.

Parameters

- **start_account_id** (*int*) – Starting AccountID to return. If no account exists with this AccountID, the next account by AccountID order is used as the start of the list. To page through the list, pass the AccountID of the last account in the previous response + 1.
- **limit** (*int*) – Maximum number of AccountInfo objects to return.
- **include_storage_containers** (*bool*) – Includes storage containers in the response by default. To exclude storage containers, set to false.

include_storage_containers = <type 'bool'>

limit = <type 'int'>

start_account_id = <type 'int'>

class solidfire.models.**ListAccountsResult**(*accounts*)
Bases: *solidfire.common.model.DataObject*

Parameters **accounts** (Account) – [required] List of accounts.

accounts = <class 'solidfire.models.Account[]'>

class solidfire.models.**ListActiveNodesResult**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (Node) – [required]

nodes = <class 'solidfire.models.Node[]'>

class solidfire.models.**ListActivePairedVolumesRequest**(*start_volume_id=None*, *limit=None*)
Bases: *solidfire.common.model.DataObject*

ListActivePairedVolumes enables you to list all the active volumes paired with a volume. This method returns information about volumes with active and pending pairings.

Parameters

- **start_volume_id** (*int*) – The beginning of the range of active paired volumes to return.
- **limit** (*int*) – Maximum number of active paired volumes to return.

limit = <type 'int'>

start_volume_id = <type 'int'>

class solidfire.models.**ListActivePairedVolumesResult**(*volumes*)
Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] Volume information for the paired volumes.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**ListActiveVolumesRequest**(*start_volume_id=None*, *limit=None*, *include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

ListActiveVolumes enables you to return the list of active volumes currently in the system. The list of volumes is returned sorted in VolumeID order and can be returned in multiple parts (pages).

Parameters

- **start_volume_id** (*int*) – Starting VolumeID to return. If no volume exists with this VolumeID, the next volume by VolumeID order is used as the start of the list. To page through the list, pass the VolumeID of the last volume in the previous response + 1.
- **limit** (*int*) – Maximum number of Volume Info objects to return. A value of 0 (zero) returns all volumes (unlimited).
- **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

include_virtual_volumes = <type 'bool'>

limit = <type 'int'>

start_volume_id = <type 'int'>

class solidfire.models.**ListActiveVolumesResult**(*volumes*)
Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] List of active volumes.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**ListAllNodesResult**(*nodes*, *pending_nodes*, *pending_active_nodes=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **nodes** (Node) – [required]
- **pending_nodes** (PendingNode) – [required]
- **pending_active_nodes** (PendingActiveNode) – List of objects detailing information about all PendingActive nodes in the system.

nodes = <class 'solidfire.models.Node[]'>

pending_active_nodes = <class 'solidfire.models.PendingActiveNode[]'>

pending_nodes = <class 'solidfire.models.PendingNode[]'>

class solidfire.models.**ListAsyncResultsRequest**(*async_result_types=None*)
Bases: *solidfire.common.model.DataObject*

You can use ListAsyncResults to list the results of all currently running and completed asynchronous methods on the system. Querying asynchronous results with ListAsyncResults does not cause completed asyncHandles to expire; you can use GetAsyncResult to query any of the asyncHandles returned by ListAsyncResults.

Parameters **async_result_types** (*str*) – An optional list of types of results. You can use this list to restrict the results to only these types of operations. Possible values are: BulkVolume: Copy operations between volumes, such as backups or restores. Clone: Volume cloning operations. DriveRemoval: Operations involving the system copying data from a drive in preparation to remove it from the cluster. RtfiPendingNode: Operations involving the system installing compatible software on a node before adding it to the cluster

async_result_types = <type 'str[]'>

class solidfire.models.**ListAsyncResultsResult**(*async_handles*)
Bases: *solidfire.common.model.DataObject*

Parameters **async_handles** (AsyncHandle) – [required] An array of serialized asynchronous method results.

async_handles = <class 'solidfire.models.AsyncHandle[]'>

class solidfire.models.**ListBackupTargetsResult**(*backup_targets*)
Bases: *solidfire.common.model.DataObject*

Parameters **backup_targets** (BackupTarget) – [required] Objects returned for each backup target.

backup_targets = <class 'solidfire.models.BackupTarget[]'>

class solidfire.models.**ListBulkVolumeJobsResult**(*bulk_volume_jobs*)
Bases: *solidfire.common.model.DataObject*

Parameters **bulk_volume_jobs** (BulkVolumeJob) – [required] An array of information for each bulk volume job.

bulk_volume_jobs = <class 'solidfire.models.BulkVolumeJob[]'>

class solidfire.models.**ListClusterAdminsResult**(*cluster_admins*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_admins** (ClusterAdmin) – [required] Information about the cluster admin.

cluster_admins = <class 'solidfire.models.ClusterAdmin[]'>

class solidfire.models.**ListClusterFaultsRequest**(*best_practices=None*, *fault_types=None*)
Bases: *solidfire.common.model.DataObject*

ListClusterFaults enables you to retrieve information about any faults detected on the cluster. With this method, you can retrieve both current faults as well as faults that have been resolved. The system caches faults every 30 seconds.

Parameters

- **best_practices** (*bool*) – Specifies whether to include faults triggered by suboptimal system configuration. Possible values are: true false
- **fault_types** (*str*) – Determines the types of faults returned. Possible values are: current: List active, unresolved faults. resolved: List faults that were previously detected and resolved. all: (Default) List both current and resolved faults. You can see the fault status in the resolved field of the Cluster Fault object.

best_practices = <type 'bool'>

fault_types = <type 'str'>

class solidfire.models.**ListClusterFaultsResult**(*faults*)
Bases: *solidfire.common.model.DataObject*

Parameters **faults** (ClusterFaultInfo) – [required] The list of Cluster Fault objects.

faults = <class 'solidfire.models.ClusterFaultInfo[]'>

class solidfire.models.**ListClusterPairsResult**(*cluster_pairs*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster_pairs** (PairedCluster) – [required] Information about each paired cluster.

cluster_pairs = <class 'solidfire.models.PairedCluster[]'>

class solidfire.models.**ListDeletedVolumesRequest**(*include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

ListDeletedVolumes enables you to retrieve the list of volumes that have been marked for deletion and purged from the system.

Parameters **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

include_virtual_volumes = <type 'bool'>

class solidfire.models.**ListDeletedVolumesResult**(*volumes*)
Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] List of deleted volumes.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**ListDriveHardwareRequest**(*force*)
Bases: *solidfire.common.model.DataObject*

ListDriveHardware returns all the drives connected to a node. Use this method on individual nodes to return drive hardware information or use this method on the cluster master node MVIP to see information for all the drives on all nodes. Note: The "securitySupported": true line of the method response does not imply that the drives are capable of encryption; only that the security status can be queried. If you have a node type with a model number ending in "-NE", commands to enable security features on these drives will fail. See the EnableEncryptionAtRest method for more information.

Parameters **force** (*bool*) – [required] To run this command, the force parameter must be set to true.

force = <type 'bool'>

class solidfire.models.**ListDriveHardwareResult**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (NodeDriveHardware) – [required]

nodes = <class 'solidfire.models.NodeDriveHardware[]'>

class solidfire.models.**ListDriveStatsRequest**(*drives=None*)
Bases: *solidfire.common.model.DataObject*

ListDriveStats enables you to retrieve high-level activity measurements for multiple drives in the cluster. By default, this method returns statistics for all drives in the cluster, and these measurements are cumulative from the addition of the drive to the cluster. Some values this method returns are specific to block drives, and some are specific to metadata drives.

Parameters **drives** (*int*) – Optional list of DriveIDs for which to return drive statistics. If you omit this parameter, measurements for all drives are returned.

drives = <type 'int[]'>

class solidfire.models.**ListDriveStatsResult**(*drive_stats*, *errors*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **drive_stats** (DriveStats) – [required] List of drive activity information for each drive.
- **errors** (*dict*) – [required] If there are errors retrieving information about a drive, this list contains the driveID and associated error message. Always present, and empty if there are no errors.

drive_stats = <class 'solidfire.models.DriveStats[]'>

errors = <type 'dict[]'>

class solidfire.models.**ListDrivesResult**(*drives*)
Bases: *solidfire.common.model.DataObject*

Parameters **drives** (DriveInfo) – [required] Information for the drives that are connected to the cluster.

drives = <class 'solidfire.models.DriveInfo[]'>

class solidfire.models.**ListEventsRequest**(*max_events=None*, *start_event_id=None*, *end_event_id=None*, *event_type=None*)
Bases: *solidfire.common.model.DataObject*

ListEvents returns events detected on the cluster, sorted from oldest to newest.

Parameters

- **max_events** (*int*) – Specifies the maximum number of events to return.

- **start_event_id** (*int*) – Identifies the beginning of a range of events to return.
- **end_event_id** (*int*) – Identifies the end of a range of events to return.
- **event_type** (*str*) –

end_event_id = <type 'int'>

event_type = <type 'str'>

max_events = <type 'int'>

start_event_id = <type 'int'>

class solidfire.models.**ListEventsResult**(*event_queue_type*, *events*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **event_queue_type** (*str*) – [required]
- **events** (EventInfo) – [required]

event_queue_type = <type 'str'>

events = <class 'solidfire.models.EventInfo[]'>

class solidfire.models.**ListFibreChannelPortInfoResult**(*fibre_channel_port_info*)
Bases: *solidfire.common.model.DataObject*

ListFibreChannelPortInfoResult is used to return information about the Fibre Channel ports.

Parameters **fibre_channel_port_info** (*dict*) – [required] Used to return information about the Fibre Channel ports.

fibre_channel_port_info = <type 'dict'>

class solidfire.models.**ListFibreChannelSessionsResult**(*sessions*)
Bases: *solidfire.common.model.DataObject*

Used to return information about the Fibre Channel sessions.

Parameters **sessions** (FibreChannelSession) – [required] A list of FibreChannelSession objects with information about the Fibre Channel session.

sessions = <class 'solidfire.models.FibreChannelSession[]'>

class solidfire.models.**ListGroupSnapshotsRequest**(*volumes=None*, *group_snapshot_id=None*)
Bases: *solidfire.common.model.DataObject*

ListGroupSnapshots enables you to get information about all group snapshots that have been created.

Parameters

- **volumes** (*int*) – An array of unique volume IDs to query. If you do not specify this parameter, all group snapshots on the cluster are included.
- **group_snapshot_id** (*int*) – Retrieves information for a specific group snapshot ID.

group_snapshot_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.models.**ListGroupSnapshotsResult**(*group_snapshots*)
Bases: *solidfire.common.model.DataObject*

Parameters **group_snapshots** (GroupSnapshot) – [required] List of Group Snapshots.

group_snapshots = <class 'solidfire.models.GroupSnapshot[]'>

class solidfire.models.**ListISCSISessionsResult**(*sessions*)
Bases: *solidfire.common.model.DataObject*

Parameters **sessions** (ISCSISession) – [required]

sessions = <class 'solidfire.models.ISCSISession[]'>

class solidfire.models.**ListInitiatorsRequest**(*start_initiator_id=None*, *limit=None*, *initiators=None*)
Bases: *solidfire.common.model.DataObject*

ListInitiators enables you to list initiator IQNs or World Wide Port Names (WWPNs).

Parameters

- **start_initiator_id** (*int*) – The initiator ID at which to begin the listing. You can supply this parameter or the "initiators" parameter, but not both.
- **limit** (*int*) – The maximum number of initiator objects to return.
- **initiators** (*int*) – A list of initiator IDs to retrieve. You can provide a value for this parameter or the "startInitiatorID" parameter, but not both.

initiators = <type 'int[]'>

limit = <type 'int'>

start_initiator_id = <type 'int'>

class solidfire.models.**ListInitiatorsResult**(*initiators*)
Bases: *solidfire.common.model.DataObject*

Parameters **initiators** (Initiator) – [required] List of the initiator information.

initiators = <class 'solidfire.models.Initiator[]'>

class solidfire.models.**ListNetworkInterfacesResult**(*interfaces*)
Bases: *solidfire.common.model.DataObject*

Parameters **interfaces** (NetworkInterface) – [required]

interfaces = <class 'solidfire.models.NetworkInterface[]'>

class solidfire.models.**ListNodeFibreChannelPortInfoResult**(*fibre_channel_ports*)
Bases: *solidfire.common.model.DataObject*

List of fibre channel port info results grouped by node.

Parameters **fibre_channel_ports** (FibreChannelPortInfo) – [required] List of all physical Fibre Channel ports.

fibre_channel_ports = <class 'solidfire.models.FibreChannelPortInfo[]'>

class solidfire.models.**ListNodeStatsResult**(*node_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **node_stats** (NodeStatsNodes) – [required] Node activity information for all nodes.

node_stats = <class 'solidfire.models.NodeStatsNodes'>

class solidfire.models.**ListPendingActiveNodesResult**(*pending_active_nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **pending_active_nodes** (PendingActiveNode) – [required] List of objects detailing information about all PendingActive nodes in the system.

pending_active_nodes = <class 'solidfire.models.PendingActiveNode[]'>

class solidfire.models.**ListPendingNodesResult**(*pending_nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **pending_nodes** (PendingNode) – [required]

pending_nodes = <class 'solidfire.models.PendingNode[]'>

class solidfire.models.**ListProtocolEndpointsRequest**(*protocol_endpoint_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListProtocolEndpoints enables you to retrieve information about all protocol endpoints in the cluster. Protocol endpoints govern access to their associated virtual volume storage containers.

Parameters **protocol_endpoint_ids** (*UUID*) – A list of protocol endpoint IDs for which to retrieve information. If you omit this parameter, the method returns information about all protocol endpoints.

protocol_endpoint_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListProtocolEndpointsResult**(*protocol_endpoints*)
Bases: *solidfire.common.model.DataObject*

Parameters **protocol_endpoints** (ProtocolEndpoint) – [required]

protocol_endpoints = <class 'solidfire.models.ProtocolEndpoint[]'>

class solidfire.models.**ListQoSPoliciesResult**(*qos_policies*)
Bases: *solidfire.common.model.DataObject*

Parameters **qos_policies** (QoSPolicy) – [required] A list of details about each QoS policy.

qos_policies = <class 'solidfire.models.QoSPolicy[]'>

class solidfire.models.**ListSchedulesResult**(*schedules*)
Bases: *solidfire.common.model.DataObject*

Parameters **schedules** (Schedule) – [required] The list of schedules currently on the cluster.

schedules = <class 'solidfire.models.Schedule[]'>

class solidfire.models.**ListServicesResult**(*services*)
Bases: *solidfire.common.model.DataObject*

Parameters **services** (DetailedService) – [required]

services = <class 'solidfire.models.DetailedService[]'>

class solidfire.models.**ListSnapshotsRequest**(*volume_id=None*, *snapshot_id=None*)
Bases: *solidfire.common.model.DataObject*

ListSnapshots enables you to return the attributes of each snapshot taken on the volume. Information about snapshots that reside on the target cluster is displayed on the source cluster when this method is called from the source cluster.

Parameters

- **volume_id** (*int*) – Retrieves snapshots for a volume. If volumeID is not provided, all snapshots for all volumes are returned.
- **snapshot_id** (*int*) – Retrieves information for a specific snapshot ID.

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**ListSnapshotsResult**(*snapshots*)
Bases: *solidfire.common.model.DataObject*

Parameters **snapshots** (Snapshot) – [required] Information about each snapshot for each volume. If volumeID is not provided, all snapshots for all volumes is returned. Snapshots that are in a group will be returned with a "groupID". Snapshots that are enabled for replication.

snapshots = <class 'solidfire.models.Snapshot[]'>

class solidfire.models.**ListStorageContainersRequest**(*storage_container_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListStorageContainers enables you to retrieve information about all virtual volume storage containers known to the system.

Parameters **storage_container_ids** (*UUID*) – A list of storage container IDs for which to retrieve information. If you omit this parameter, the method returns information about all storage containers in the system.

storage_container_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListStorageContainersResult**(*storage_containers*)
Bases: *solidfire.common.model.DataObject*

Parameters **storage_containers** (StorageContainer) – [required]

storage_containers = <class 'solidfire.models.StorageContainer[]'>

class solidfire.models.**ListSyncJobsResult**(*sync_jobs*)
Bases: *solidfire.common.model.DataObject*

Parameters **sync_jobs** (SyncJob) – [required]

sync_jobs = <class 'solidfire.models.SyncJob[]'>

class solidfire.models.**ListTestsResult**(*tests*)
Bases: *solidfire.common.model.DataObject*

Parameters **tests** (*str*) – [required] List of tests that can be performed on the node.

tests = <type 'str'>

class solidfire.models.**ListUtilitiesResult**(*utilities*)
Bases: *solidfire.common.model.DataObject*

Parameters **utilities** (*str*) – [required] List of utilities currently available to run on the node.

utilities = <type 'str'>

class solidfire.models.**ListVirtualNetworksRequest**(*virtual_network_id=None*, *virtual_network_tag=None*, *virtual_network_ids=None*, *virtual_network_tags=None*)
Bases: *solidfire.common.model.DataObject*

ListVirtualNetworks enables you to list all configured virtual networks for the cluster. You can use this method to verify the virtual network settings in the cluster. There are no required parameters for this method. However, to filter the results, you can pass one or more VirtualNetworkID or VirtualNetworkTag values.

Parameters

- **virtual_network_id** (*int*) – Network ID to filter the list for a single virtual network.
- **virtual_network_tag** (*int*) – Network tag to filter the list for a single virtual network.

- **virtual_network_ids** (*int*) – Network IDs to include in the list.
- **virtual_network_tags** (*int*) – Network tag to include in the list.

virtual_network_id = <type 'int'>

virtual_network_ids = <type 'int[]'>

virtual_network_tag = <type 'int'>

virtual_network_tags = <type 'int[]'>

class solidfire.models.**ListVirtualNetworksResult**(*virtual_networks*)
Bases: *solidfire.common.model.DataObject*

Parameters **virtual_networks** (VirtualNetwork) – [required] Object containing virtual network IP addresses.

virtual_networks = <class 'solidfire.models.VirtualNetwork[]'>

class solidfire.models.**ListVirtualVolumeBindingsRequest**(*virtual_volume_binding_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListVirtualVolumeBindings returns a list of all virtual volumes in the cluster that are bound to protocol endpoints.

Parameters **virtual_volume_binding_ids** (*int*) – A list of virtual volume binding IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume bindings.

virtual_volume_binding_ids = <type 'int[]'>

class solidfire.models.**ListVirtualVolumeBindingsResult**(*bindings*)
Bases: *solidfire.common.model.DataObject*

Parameters **bindings** (VirtualVolumeBinding) – [required] Describes the VVol <-> Host binding.

bindings = <class 'solidfire.models.VirtualVolumeBinding[]'>

class solidfire.models.**ListVirtualVolumeHostsRequest**(*virtual_volume_host_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListVirtualVolumeHosts returns a list of all virtual volume hosts known to the cluster. A virtual volume host is a VMware ESX host that has initiated a session with the VASA API provider.

Parameters **virtual_volume_host_ids** (*UUID*) – A list of virtual volume host IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume hosts.

virtual_volume_host_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListVirtualVolumeHostsResult**(*hosts*)
Bases: *solidfire.common.model.DataObject*

Parameters **hosts** (VirtualVolumeHost) – [required] List of known ESX hosts.

hosts = <class 'solidfire.models.VirtualVolumeHost[]'>

class solidfire.models.**ListVirtualVolumeTasksRequest**(*virtual_volume_task_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListVirtualVolumeTasks returns a list of virtual volume tasks in the system.

Parameters **virtual_volume_task_ids** (*UUID*) – A list of virtual volume task IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume tasks.

virtual_volume_task_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListVirtualVolumeTasksResult**(*tasks*)

Bases: *solidfire.common.model.DataObject*

Parameters **tasks** (VirtualVolumeTask) – [required] List of VVol Async Tasks.

tasks = <class 'solidfire.models.VirtualVolumeTask[]'>

class solidfire.models.**ListVirtualVolumesRequest**(*details=None*, *limit=None*, *recursive=None*, *start_virtual_volume_id=None*, *virtual_volume_ids=None*)

Bases: *solidfire.common.model.DataObject*

ListVirtualVolumes enables you to list the virtual volumes currently in the system. You can use this method to list all virtual volumes, or only list a subset.

Parameters

- **details** (*bool*) – Specifies the level of detail about each virtual volume that is returned. Possible values are: true: Include more details about each virtual volume in the response. false: Include the standard level of detail about each virtual volume in the response.
- **limit** (*int*) – The maximum number of virtual volumes to list.
- **recursive** (*bool*) – Specifies whether to include information about the children of each virtual volume in the response. Possible values are: true: Include information about the children of each virtual volume in the response. false: Do not include information about the children of each virtual volume in the response.
- **start_virtual_volume_id** (*UUID*) – The ID of the virtual volume at which to begin the list.
- **virtual_volume_ids** (*UUID*) – A list of virtual volume IDs for which to retrieve information. If you specify this parameter, the method returns information about only these virtual volumes.

details = <type 'bool'>

limit = <type 'int'>

recursive = <type 'bool'>

start_virtual_volume_id = <class 'uuid.UUID'>

virtual_volume_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListVirtualVolumesResult**(*virtual_volumes*, *next_virtual_volume_id=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_volumes** (VirtualVolumeInfo) – [required]
- **next_virtual_volume_id** (*UUID*) –

next_virtual_volume_id = <class 'uuid.UUID'>

virtual_volumes = <class 'solidfire.models.VirtualVolumeInfo[]'>

class solidfire.models.**ListVolumeAccessGroupsRequest**(*start_volume_access_group_id=None*, *limit=None*, *volume_access_groups=None*)

Bases: *solidfire.common.model.DataObject*

ListVolumeAccessGroups enables you to return information about the volume access groups that are currently in the system.

Parameters

- **start_volume_access_group_id** (*int*) – The volume access group ID at which to begin the listing. If unspecified, there is no lower limit (implicitly 0).
- **limit** (*int*) – The maximum number of results to return. This can be useful for paging.
- **volume_access_groups** (*int*) – The list of ids of the volume access groups you wish to list

limit = <type 'int'>

start_volume_access_group_id = <type 'int'>

volume_access_groups = <type 'int[]'>

class solidfire.models.**ListVolumeAccessGroupsResult**(*volume_access_groups*, *volume_access_groups_not_found=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **volume_access_groups** (VolumeAccessGroup) – [required] A list of objects describing each volume access group.
- **volume_access_groups_not_found** (*int*) – A list of volume access groups not found by the system. Present if you used the "volumeAccessGroups" parameter and the system was unable to find one or more volume access groups that you specified.

volume_access_groups = <class 'solidfire.models.VolumeAccessGroup[]'>

volume_access_groups_not_found = <type 'int[]'>

class solidfire.models.**ListVolumeStatsByAccountRequest**(*accounts=None*, *include_virtual_volumes=None*)

Bases: *solidfire.common.model.DataObject*

ListVolumeStatsByAccount returns high-level activity measurements for every account. Values are summed from all the volumes owned by the account.

Parameters

- **accounts** (*int*) – One or more account ids by which to filter the result.
- **include_virtual_volumes** (*bool*) – Includes virtual volumes in the response by default. To exclude virtual volumes, set to false.

accounts = <type 'int[]'>

include_virtual_volumes = <type 'bool'>

class solidfire.models.**ListVolumeStatsByAccountResult**(*volume_stats*)

Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VolumeStats) – [required] List of account activity information. Note: The volumeID member is 0 for each entry, as the values represent the summation of all volumes owned by the account.

volume_stats = <class 'solidfire.models.VolumeStats[]'>

class solidfire.models.**ListVolumeStatsByVirtualVolumeRequest**(*virtual_volume_ids=None*)

Bases: *solidfire.common.model.DataObject*

ListVolumeStatsByVirtualVolume enables you to list volume statistics for any volumes in the system that are associated with virtual volumes. Statistics are cumulative from the creation of the volume.

Parameters **virtual_volume_ids** (*UUID*) – A list of one or more virtual volume IDs for which to retrieve information. If you specify this parameter, the method returns information about only these virtual volumes.

virtual_volume_ids = <class 'uuid.UUID[]'>

class solidfire.models.**ListVolumeStatsByVirtualVolumeResult**(*volume_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VirtualVolumeStats) – [required]

volume_stats = <class 'solidfire.models.VirtualVolumeStats[]'>

class solidfire.models.**ListVolumeStatsByVolumeAccessGroupRequest**(*volume_access_groups=None*, *include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

ListVolumeStatsByVolumeAccessGroup enables you to get total activity measurements for all of the volumes that are a member of the specified volume access group(s).

Parameters

- **volume_access_groups** (*int*) – An array of VolumeAccessGroupIDs for which volume activity is returned. If omitted, statistics for all volume access groups are returned.
- **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

include_virtual_volumes = <type 'bool'>

volume_access_groups = <type 'int[]'>

class solidfire.models.**ListVolumeStatsByVolumeAccessGroupResult**(*volume_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VolumeStats) – [required] List of account activity information. Note: The volumeID member is 0 for each entry, as the values represent the summation of all volumes owned by the account.

volume_stats = <class 'solidfire.models.VolumeStats[]'>

class solidfire.models.**ListVolumeStatsByVolumeRequest**(*include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

ListVolumeStatsByVolume returns high-level activity measurements for every volume, by volume. Values are cumulative from the creation of the volume.

Parameters **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

include_virtual_volumes = <type 'bool'>

class solidfire.models.**ListVolumeStatsByVolumeResult**(*volume_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VolumeStats) – [required] List of account activity information.

volume_stats = <class 'solidfire.models.VolumeStats[]'>

class solidfire.models.**ListVolumeStatsRequest**(*volume_ids=None*)
Bases: *solidfire.common.model.DataObject*

ListVolumeStats returns high-level activity measurements for a single volume, list of volumes, or all volumes (if you omit the volumeIDs parameter). Measurement values are cumulative from the creation of the volume.

Parameters **volume_ids** (*int*) – A list of volume IDs of volumes from which to retrieve activity information.

volume_ids = <type 'int[]'>

class solidfire.models.**ListVolumeStatsResult**(*volume_stats*)
Bases: *solidfire.common.model.DataObject*

Parameters **volume_stats** (VolumeStats) – [required] List of volume activity information.

volume_stats = <class 'solidfire.models.VolumeStats[]'>

class solidfire.models.**ListVolumesForAccountRequest**(*account_id*, *start_volume_id=None*, *limit=None*, *include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

ListVolumesForAccount returns the list of active and (pending) deleted volumes for an account.

Parameters

- **account_id** (*int*) – [required] Returns all volumes owned by this AccountID.
- **start_volume_id** (*int*) – The ID of the first volume to list. This can be useful for paging results. By default, this starts at the lowest VolumeID.
- **limit** (*int*) – The maximum number of volumes to return from the API.
- **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

account_id = <type 'int'>

include_virtual_volumes = <type 'bool'>

limit = <type 'int'>

start_volume_id = <type 'int'>

class solidfire.models.**ListVolumesForAccountResult**(*volumes*)
Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] List of volumes.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**ListVolumesRequest**(*start_volume_id=None*, *limit=None*, *volume_status=None*, *accounts=None*, *is_paired=None*, *volume_ids=None*, *volume_name=None*, *include_virtual_volumes=None*)
Bases: *solidfire.common.model.DataObject*

The ListVolumes method enables you to retrieve a list of volumes that are in a cluster. You can specify the volumes you want to return in the list by using the available parameters.

Parameters

- **start_volume_id** (*int*) – Only volumes with an ID greater than or equal to this value are returned. Mutually exclusive with the volumeIDs parameter.

- **limit** (*int*) – Specifies the maximum number of volume results that are returned. Mutually exclusive with the volumeIDs parameter.
- **volume_status** (*str*) – Only volumes with a status equal to the status value are returned. Possible values are: creating snapshotting active deleted
- **accounts** (*int*) – Returns only the volumes owned by the accounts you specify here. Mutually exclusive with the volumeIDs parameter.
- **is_paired** (*bool*) – Returns volumes that are paired or not paired. Possible values are: true: Returns all paired volumes. false: Returns all volumes that are not paired.
- **volume_ids** (*int*) – A list of volume IDs. If you supply this parameter, other parameters operate only on this set of volumes. Mutually exclusive with the accounts, startVolumeID, and limit parameters.
- **volume_name** (*str*) – Only volume object information matching the volume name is returned.
- **include_virtual_volumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

accounts = <type 'int[]'>

include_virtual_volumes = <type 'bool'>

is_paired = <type 'bool'>

limit = <type 'int'>

start_volume_id = <type 'int'>

volume_ids = <type 'int[]'>

volume_name = <type 'str'>

volume_status = <type 'str'>

class solidfire.models.**ListVolumesResult**(*volumes*)

Bases: *solidfire.common.model.DataObject*

Parameters **volumes** (Volume) – [required] List of volumes.

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**LoggingServer**(*host*, *port*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **host** (*str*) – [required] Hostname or IP address of the log server.
- **port** (*int*) – [required] Port number that the log server is listening on.

host = <type 'str'>

port = <type 'int'>

class solidfire.models.**LoginBanner**(*banner*, *enabled*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **banner** (*str*) – [required] The current text of the Terms of Use banner. This value can contain text even when the banner is disabled.

- **enabled** (*bool*) – [required] The status of the Terms of Use banner. Possible values: true: The Terms of Use banner is displayed upon web interface login. false: The Terms of Use banner is not displayed upon web interface login.

banner = <type 'str'>

enabled = <type 'bool'>

class solidfire.models.**LoginSessionInfo**(*timeout*)
Bases: *solidfire.common.model.DataObject*

Parameters **timeout** (*str*) – [required] The time, in minutes, when this session will timeout and expire. Formatted in H:mm:ss. For example: 1:30:00, 20:00, 5:00. All leading zeros and colons are removed regardless of the format the timeout was entered.

timeout = <type 'str'>

class solidfire.models.**LunAssignment**(*volume_id*, *lun*)
Bases: *solidfire.common.model.DataObject*

VolumeID and Lun assignment.

Parameters

- **volume_id** (*int*) – [required] The volume ID assigned to the Lun.
- **lun** (*int*) – [required] Correct LUN values are 0 - 16383. An exception will be seen if an incorrect LUN value is passed.

lun = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**MetadataHosts**(*dead_secondaries*, *live_secondaries*, *primary*)
Bases: *solidfire.common.model.DataObject*

The volume services on which the volume metadata resides.

Parameters

- **dead_secondaries** (*int*) – [required] Secondary metadata (slice) services that are in a dead state.
- **live_secondaries** (*int*) – [required] Secondary metadata (slice) services that are currently in a "live" state.
- **primary** (*int*) – [required] The primary metadata (slice) services hosting the volume.

dead_secondaries = <type 'int[]'>

live_secondaries = <type 'int[]'>

primary = <type 'int'>

class solidfire.models.**ModifyAccountRequest**(*account_id*, *username=None*, *status=None*, *initiator_secret=None*, *target_secret=None*, *attributes=None*)
Bases: *solidfire.common.model.DataObject*

ModifyAccount enables you to modify an existing account. When you lock an account, any existing connections from that account are immediately terminated. When you change an account's CHAP settings, any existing connections remain active, and the new CHAP settings are used on subsequent connections or reconnections. To clear an account's attributes, specify {} for the attributes parameter.

Parameters

- **account_id** (*int*) – [required] Specifies the AccountID for the account to be modified.

- **username** (*str*) – Specifies the username associated with the account. (Might be 1 to 64 characters in length).
- **status** (*str*) – Sets the status for the account. Possible values are: active: The account is active and connections are allowed. locked: The account is locked and connections are refused.
- **initiator_secret** (CHAPSecret) – Specifies the CHAP secret to use for the initiator. This secret must be 12-16 characters in length and should be impenetrable. The initiator CHAP secret must be unique and cannot be the same as the target CHAP secret.
- **target_secret** (CHAPSecret) – Specifies the CHAP secret to use for the target (mutual CHAP authentication). This secret must be 12-16 characters in length and should be impenetrable. The target CHAP secret must be unique and cannot be the same as the initiator CHAP secret.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

account_id = <type 'int'>

attributes = <type 'dict'>

initiator_secret = <class 'solidfire.models.CHAPSecret'>

status = <type 'str'>

target_secret = <class 'solidfire.models.CHAPSecret'>

username = <type 'str'>

class solidfire.models.**ModifyAccountResult**(*account*)

Bases: *solidfire.common.model.DataObject*

Parameters **account** (Account) – [required]

account = <class 'solidfire.models.Account'>

class solidfire.models.**ModifyBackupTargetRequest**(*backup_target_id*, *name=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

ModifyBackupTarget enables you to change attributes of a backup target.

Parameters

- **backup_target_id** (*int*) – [required] The unique target ID for the target to modify.
- **name** (*str*) – The new name for the backup target.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

backup_target_id = <type 'int'>

name = <type 'str'>

class solidfire.models.**ModifyBackupTargetResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**ModifyClusterAdminRequest**(*cluster_admin_id*, *password=None*, *access=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use ModifyClusterAdmin to change the settings for a cluster admin or LDAP cluster admin. You cannot change access for the administrator cluster admin account.

Parameters

- **cluster_admin_id** (*int*) – [required] ClusterAdminID for the cluster admin or LDAP cluster admin to modify.
- **password** (*str*) – Password used to authenticate this cluster admin.
- **access** (*str*) – Controls which methods this cluster admin can use. For more details, see Access Control in the Element API Reference Guide.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

access = <type 'str[]'>

attributes = <type 'dict'>

cluster_admin_id = <type 'int'>

password = <type 'str'>

class solidfire.models.**ModifyClusterAdminResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**ModifyClusterFullThresholdRequest**(*stage2_aware_threshold=None*, *stage3_block_threshold_percent=None*, *max_metadata_over_provision_factor=None*)

Bases: *solidfire.common.model.DataObject*

You can use ModifyClusterFullThreshold to change the level at which the system generates an event when the storage cluster approaches a certain capacity utilization. You can use the threshold setting to indicate the acceptable amount of utilized block storage before the system generates a warning. For example, if you want to be alerted when the system reaches 3% below the "Error" level block storage utilization, enter a value of "3" for the stage3BlockThresholdPercent parameter. If this level is reached, the system sends an alert to the Event Log in the Cluster Management Console.

Parameters

- **stage2_aware_threshold** (*int*) – The number of nodes of capacity remaining in the cluster before the system triggers a capacity notification.
- **stage3_block_threshold_percent** (*int*) – The percentage of block storage utilization below the "Error" threshold that causes the system to trigger a cluster "Warning" alert.
- **max_metadata_over_provision_factor** (*int*) – A value representative of the number of times metadata space can be overprovisioned relative to the amount of space available. For example, if there was enough metadata space to store 100 TiB of volumes and this number was set to 5, then 500 TiB worth of volumes can be created.

max_metadata_over_provision_factor = <type 'int'>

stage2_aware_threshold = <type 'int'>

stage3_block_threshold_percent = <type 'int'>

class solidfire.models.**ModifyClusterFullThresholdResult**(*block_fullness*, *fullness*, *max_metadata_over_provision_factor*, *metadata_fullness*, *slice_reserve_used_threshold_pct*, *stage2_aware_threshold*, *stage2_block_threshold_bytes*, *stage3_block_threshold_bytes*, *stage3_block_threshold_percent*, *stage3_low_threshold*, *stage4_critical_threshold*, *stage4_block_threshold_bytes*, *stage5_block_threshold_bytes*, *sum_total_cluster_bytes*, *sum_total_metadata_cluster_bytes*, *sum_used_cluster_bytes*, *sum_used_metadata_cluster_bytes*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **block_fullness** (*str*) – [required] Current computed level of block fullness of the cluster. Possible values: stage1Happy: No alerts or error conditions. stage2Aware: 3 nodes of capacity available. stage3Low: 2 nodes of capacity available. stage4Critical: 1 node of capacity available. No new volumes or clones can be created. stage5CompletelyConsumed: Completely consumed. Cluster is read-only, iSCSI connection is maintained but all writes are suspended.
- **fullness** (*str*) – [required] Reflects the highest level of fullness between "blockFullness" and "metadataFullness".
- **max_metadata_over_provision_factor** (*int*) – [required] A value representative of the number of times metadata space can be over provisioned relative to the amount of space available. For example, if there was enough metadata space to store 100 TiB of volumes and this number was set to 5, then 500 TiB worth of volumes could be created.
- **metadata_fullness** (*str*) – [required] Current computed level of metadata fullness of the cluster.
- **slice_reserve_used_threshold_pct** (*int*) – [required] Error condition; message sent to "Alerts" if the reserved slice utilization is greater than the sliceReserveUsedThresholdPct value returned.
- **stage2_aware_threshold** (*int*) – [required] Awareness condition: Value that is set for "Stage 2" cluster threshold level.
- **stage2_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage2 condition will exist.
- **stage3_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage3 condition will exist.
- **stage3_block_threshold_percent** (*int*) – [required] The percent value set for stage3. At this percent full, a warning will be posted in the Alerts log.
- **stage3_low_threshold** (*int*) – [required] Error condition; message sent to "Alerts" that capacity on a cluster is getting low.
- **stage4_critical_threshold** (*int*) – [required] Error condition; message sent to "Alerts" that capacity on a cluster is critically low.

- **stage4_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage4 condition will exist.
- **stage5_block_threshold_bytes** (*int*) – [required] Number of bytes being used by the cluster at which a stage5 condition will exist.
- **sum_total_cluster_bytes** (*int*) – [required] Physical capacity of the cluster measured in bytes.
- **sum_total_metadata_cluster_bytes** (*int*) – [required] Total amount of space that can be used to store metadata.
- **sum_used_cluster_bytes** (*int*) – [required] Number of bytes used on the cluster.
- **sum_used_metadata_cluster_bytes** (*int*) – [required] Amount of space used on volume drives to store metadata.

block_fullness = <type 'str'>

fullness = <type 'str'>

max_metadata_over_provision_factor = <type 'int'>

metadata_fullness = <type 'str'>

slice_reserve_used_threshold_pct = <type 'int'>

stage2_aware_threshold = <type 'int'>

stage2_block_threshold_bytes = <type 'int'>

stage3_block_threshold_bytes = <type 'int'>

stage3_block_threshold_percent = <type 'int'>

stage3_low_threshold = <type 'int'>

stage4_block_threshold_bytes = <type 'int'>

stage4_critical_threshold = <type 'int'>

stage5_block_threshold_bytes = <type 'int'>

sum_total_cluster_bytes = <type 'int'>

sum_total_metadata_cluster_bytes = <type 'int'>

sum_used_cluster_bytes = <type 'int'>

sum_used_metadata_cluster_bytes = <type 'int'>

class solidfire.models.**ModifyGroupSnapshotRequest**(*group_snapshot_id*, *expiration_time=None*, *enable_remote_replication=None*, *snap_mirror_label=None*)

Bases: *solidfire.common.model.DataObject*

ModifyGroupSnapshot enables you to change the attributes of a group of snapshots. You can also use this method to enable snapshots created on the Read/Write (source) volume to be remotely replicated to a target SolidFire storage system.

Parameters

- **group_snapshot_id** (*int*) – [required] Specifies the ID of the group of snapshots.
- **expiration_time** (*str*) – Sets the time when the snapshot should be removed. If unspecified, the current time is used.

- **enable_remote_replication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **snap_mirror_label** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

enable_remote_replication = <type 'bool'>

expiration_time = <type 'str'>

group_snapshot_id = <type 'int'>

snap_mirror_label = <type 'str'>

class solidfire.models.**ModifyGroupSnapshotResult**(*group_snapshot*)
Bases: *solidfire.common.model.DataObject*

Parameters **group_snapshot** (GroupSnapshot) – [required]

group_snapshot = <class 'solidfire.models.GroupSnapshot'>

class solidfire.models.**ModifyInitiator**(*initiator_id*, *alias=None*, *volume_access_group_id=None*, *attributes=None*)
Bases: *solidfire.common.model.DataObject*

Object containing characteristics of each initiator to modify

Parameters

- **initiator_id** (*int*) – [required] (Required) The numeric ID of the initiator to modify. (Integer)
- **alias** (*str*) – (Optional) A new friendly name to assign to the initiator. (String)
- **volume_access_group_id** (*int*) – (Optional) The ID of the volume access group to which the newly created initiator should be added. If the initiator was previously in a different volume access group, it is removed from the old volume access group. If this key is present but null, the initiator is removed from its current volume access group, but not placed in any new volume access group. (Integer)
- **attributes** (*dict*) – (Optional) A new set of JSON attributes assigned to this initiator. (JSON Object)

alias = <type 'str'>

attributes = <type 'dict'>

initiator_id = <type 'int'>

volume_access_group_id = <type 'int'>

class solidfire.models.**ModifyInitiatorsRequest**(*initiators*)
Bases: *solidfire.common.model.DataObject*

ModifyInitiators enables you to change the attributes of one or more existing initiators. You cannot change the name of an existing initiator. If you need to change the name of an initiator, delete it first with DeleteInitiators and create a new one with CreateInitiators. If ModifyInitiators fails to change one of the initiators provided in the parameter, the method returns an error and does not modify any initiators (no partial completion is possible).

Parameters **initiators** (ModifyInitiator) – [required] A list of objects containing characteristics of each initiator to modify. Values are: initiatorID: (Required) The ID of the initiator to modify. (Integer) alias: (Optional) A new friendly name to assign to the initiator. (String) attributes: (Optional) A new set of JSON attributes to assign to the initiator. (JSON Object) volumeAccessGroupID: (Optional) The ID of the volume access group into to which the initiator

should be added. If the initiator was previously in a different volume access group, it is removed from the old volume access group. If this key is present but null, the initiator is removed from its current volume access group, but not placed in any new volume access group. (Integer)

initiators = <class 'solidfire.models.ModifyInitiator[]'>

class solidfire.models.**ModifyInitiatorsResult**(*initiators*)
Bases: *solidfire.common.model.DataObject*

Parameters **initiators** (Initiator) – [required] List of objects containing details about the modified initiators

initiators = <class 'solidfire.models.Initiator[]'>

class solidfire.models.**ModifyQoSPolicyRequest**(*qos_policy_id*, *name=None*, *qos=None*)
Bases: *solidfire.common.model.DataObject*

You can use the ModifyQoSPolicy method to modify an existing QoSPolicy on the system.

Parameters

- **qos_policy_id** (*int*) – [required] The ID of the policy to be modified.
- **name** (*str*) – If supplied, the name of the QoS Policy (e.g. gold, platinum, silver) is changed to this value.
- **qos** (QoS) – If supplied, the QoS settings for this policy are changed to these sttings. You can supply partial QoS values and only change some of the QoS settings.

name = <type 'str'>

qos = <class 'solidfire.models.QoS'>

qos_policy_id = <type 'int'>

class solidfire.models.**ModifyQoSPolicyResult**(*qos_policy*)
Bases: *solidfire.common.model.DataObject*

Parameters **qos_policy** (QoSPolicy) – [required] Details of the newly modified QoSPolicy object.

qos_policy = <class 'solidfire.models.QoSPolicy'>

class solidfire.models.**ModifyScheduleRequest**(*schedule*)
Bases: *solidfire.common.model.DataObject*

ModifySchedule enables you to change the intervals at which a scheduled snapshot occurs. This allows for adjustment to the snapshot frequency and retention.

Parameters **schedule** (Schedule) – [required] The "Schedule" object will be used to modify an existing schedule. The ScheduleID property is required. Frequency property must be of type that inherits from Frequency. Valid types are: DaysOfMonthFrequency DaysOrWeekFrequency TimeIntervalFrequency

schedule = <class 'solidfire.models.Schedule'>

class solidfire.models.**ModifyScheduleResult**(*schedule=None*)
Bases: *solidfire.common.model.DataObject*

Parameters **schedule** (Schedule) –

schedule = <class 'solidfire.models.Schedule'>

class solidfire.models.**ModifySnapshotRequest**(*snapshot_id*, *expiration_time=None*, *enable_remote_replication=None*, *snap_mirror_label=None*)

Bases: *solidfire.common.model.DataObject*

ModifySnapshot enables you to change the attributes currently assigned to a snapshot. You can use this method to enable snapshots created on the Read/Write (source) volume to be remotely replicated to a target SolidFire storage system.

Parameters

- **snapshot_id** (*int*) – [required] Specifies the ID of the snapshot.
- **expiration_time** (*str*) – Sets the time when the snapshot should be removed.
- **enable_remote_replication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **snap_mirror_label** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

enable_remote_replication = <type 'bool'>

expiration_time = <type 'str'>

snap_mirror_label = <type 'str'>

snapshot_id = <type 'int'>

class solidfire.models.**ModifySnapshotResult**(*snapshot=None*)

Bases: *solidfire.common.model.DataObject*

Parameters **snapshot** (Snapshot) –

snapshot = <class 'solidfire.models.Snapshot'>

class solidfire.models.**ModifyStorageContainerRequest**(*storage_container_id*, *initiator_secret=None*, *target_secret=None*)

Bases: *solidfire.common.model.DataObject*

ModifyStorageContainer enables you to make changes to an existing virtual volume storage container.

Parameters

- **storage_container_id** (*UUID*) – [required] The unique ID of the virtual volume storage container to modify.
- **initiator_secret** (*str*) – The new secret for CHAP authentication for the initiator.
- **target_secret** (*str*) – The new secret for CHAP authentication for the target.

initiator_secret = <type 'str'>

storage_container_id = <class 'uuid.UUID'>

target_secret = <type 'str'>

class solidfire.models.**ModifyStorageContainerResult**(*storage_container*)

Bases: *solidfire.common.model.DataObject*

Parameters **storage_container** (StorageContainer) – [required]

storage_container = <class 'solidfire.models.StorageContainer'>

class solidfire.models.**ModifyVirtualNetworkRequest**(*virtual_network_id=None*, *virtual_network_tag=None*, *name=None*, *address_blocks=None*, *netmask=None*, *svip=None*, *gateway=None*, *namespace=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use ModifyVirtualNetwork to change the attributes of an existing virtual network. This method enables you to add or remove address blocks, change the netmask, or modify the name or description of the virtual network. You can also use it to enable or disable namespaces, as well as add or remove a gateway if namespaces are enabled on the virtual network. Note: This method requires either the VirtualNetworkID or the VirtualNetworkTag as a parameter, but not both. Caution: Enabling or disabling the Routable Storage VLANs functionality for an existing virtual network by changing the "namespace" parameter disrupts any traffic handled by the virtual network. NetApp strongly recommends changing the "namespace" parameter only during a scheduled maintenance window.

Parameters

- **virtual_network_id** (*int*) – The unique identifier of the virtual network to modify. This is the virtual network ID assigned by the cluster. Note: This parameter is optional but either virtualNetworkID or virtualNetworkTag must be specified with this API method.
- **virtual_network_tag** (*int*) – The network tag that identifies the virtual network to modify. Note: This parameter is optional but either virtualNetworkID or virtualNetworkTag must be specified with this API method.
- **name** (*str*) – The new name for the virtual network.
- **address_blocks** (AddressBlockParams) – The new addressBlock to set for this virtual network. This might contain new address blocks to add to the existing object or omit unused address blocks that need to be removed. Alternatively, you can extend or reduce the size of existing address blocks. You can only increase the size of the starting addressBlocks for a virtual network object; you can never decrease it. Attributes for this parameter are: start: The start of the IP address range. (String) size: The number of IP addresses to include in the block. (Integer)
- **netmask** (*str*) – New network mask for this virtual network.
- **svip** (*str*) – The storage virtual IP address for this virtual network. The svip for a virtual network cannot be changed. You must create a new virtual network to use a different svip address.
- **gateway** (*str*) – The IP address of a gateway of the virtual network. This parameter is only valid if the "namespace" parameter is set to true.
- **namespace** (*bool*) – When set to true, enables Routable Storage VLANs functionality by recreating the virtual network and configuring a namespace to contain it. When set to false, disables the VRF functionality for the virtual network. Changing this value disrupts traffic running through this virtual network.
- **attributes** (*dict*) – A new list of name-value pairs in JSON object format.

address_blocks = <class 'solidfire.models.AddressBlockParams[]'>

attributes = <type 'dict'>

gateway = <type 'str'>

name = <type 'str'>

namespace = <type 'bool'>

netmask = <type 'str'>

svip = <type 'str'>

virtual_network_id = <type 'int'>

virtual_network_tag = <type 'int'>

class solidfire.models.**ModifyVolumeAccessGroupLunAssignmentsRequest**(*volume_access_group_id*, *lun_assignments*)

Bases: *solidfire.common.model.DataObject*

The ModifyVolumeAccessGroupLunAssignments method enables you to define custom LUN assignments for specific volumes. This method changes only LUN values set on the lunAssignments parameter in the volume access group. All other LUN assignments remain unchanged. LUN assignment values must be unique for volumes in a volume access group. You cannot define duplicate LUN values within a volume access group. However, you can use the same LUN values again in different volume access groups. Note: Correct LUN values are 0 through 16383. The system generates an exception if you pass a LUN value outside of this range. None of the specified LUN assignments are modified if there is an exception. Caution: If you change a LUN assignment for a volume with active I/O, the I/O can be disrupted. You might need to change the server configuration before changing volume LUN assignments.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group for which the LUN assignments will be modified.
- **lun_assignments** (LunAssignment) – [required] The volume IDs with new assigned LUN values.

lun_assignments = <class 'solidfire.models.LunAssignment[]'>

volume_access_group_id = <type 'int'>

class solidfire.models.**ModifyVolumeAccessGroupLunAssignmentsResult**(*volume_access_group_lun_assignm*

Bases: *solidfire.common.model.DataObject*

Parameters **volume_access_group_lun_assignments** (VolumeAccessGroupLunAssignments) – [required]

volume_access_group_lun_assignments = <class 'solidfire.models.VolumeAccessGroupLunAss

class solidfire.models.**ModifyVolumeAccessGroupRequest**(*volume_access_group_id*, *virtual_network_id=None*, *virtual_network_tags=None*, *name=None*, *initiators=None*, *volumes=None*, *delete_orphan_initiators=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use ModifyVolumeAccessGroup to update initiators and add or remove volumes from a volume access group. If a specified initiator or volume is a duplicate of what currently exists, the volume access group is left as-is. If you do not specify a value for volumes or initiators, the current list of initiators and volumes is not changed.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group to modify.

- **virtual_network_id** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **virtual_network_tags** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **name** (*str*) – The new name for this volume access group. Not required to be unique, but recommended.
- **initiators** (*str*) – List of initiators to include in the volume access group. If unspecified, the access group's configured initiators are not modified.
- **volumes** (*int*) – List of volumes to initially include in the volume access group. If unspecified, the access group's volumes are not modified.
- **delete_orphan_initiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

delete_orphan_initiators = <type 'bool'>

initiators = <type 'str[]'>

name = <type 'str'>

virtual_network_id = <type 'int[]'>

virtual_network_tags = <type 'int[]'>

volume_access_group_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.models.**ModifyVolumeAccessGroupResult**(*volume_access_group*)

Bases: *solidfire.common.model.DataObject*

Parameters **volume_access_group** (VolumeAccessGroup) – [required] An object containing information about the newly modified volume access group.

volume_access_group = <class 'solidfire.models.VolumeAccessGroup'>

class solidfire.models.**ModifyVolumePairRequest**(*volume_id*, *paused_manual=None*, *mode=None*, *pause_limit=None*)

Bases: *solidfire.common.model.DataObject*

ModifyVolumePair enables you to pause or restart replication between a pair of volumes.

Parameters

- **volume_id** (*int*) – [required] The ID of the volume to be modified.
- **paused_manual** (*bool*) – Specifies whether to pause or restart volume replication process. Valid values are: true: Pauses volume replication false: Restarts volume replication
- **mode** (*str*) – Specifies the volume replication mode. Possible values are: Async: Writes are acknowledged when they complete locally. The cluster does not wait for writes to be replicated to the target cluster. Sync: The source acknowledges the write when the data is stored locally and on the remote cluster. SnapshotsOnly: Only snapshots created on the source cluster are replicated. Active writes from the source volume are not replicated.
- **pause_limit** (*int*) – Internal use only.

mode = <type 'str'>

pause_limit = <type 'int'>

paused_manual = <type 'bool'>

volume_id = <type 'int'>

class solidfire.models.**ModifyVolumePairResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**ModifyVolumeRequest**(*volume_id*, *account_id=None*, *access=None*, *qos=None*, *total_size=None*, *attributes=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*)
Bases: *solidfire.common.model.DataObject*

ModifyVolume enables you to modify settings on an existing volume. You can make modifications to one volume at a time and changes take place immediately. If you do not specify QoS values when you modify a volume, they remain the same as before the modification. You can retrieve default QoS values for a newly created volume by running the GetDefaultQoS method. When you need to increase the size of a volume that is being replicated, do so in the following order to prevent replication errors: 1. Increase the size of the "Replication Target" volume. 2. Increase the size of the source or "Read / Write" volume. NetApp recommends that both the target and source volumes are the same size. Note: If you change the "access" status to locked or target, all existing iSCSI connections are terminated.

Parameters

- **volume_id** (*int*) – [required] VolumeID for the volume to be modified.
- **account_id** (*int*) – AccountID to which the volume is reassigned. If unspecified, the previous account name is used.
- **access** (*str*) – Specifies the access allowed for the volume. Possible values are: readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. If not specified, the access value does not change. replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked. If a value is not specified, the access value does not change.
- **qos** (*QoS*) – New QoS settings for this volume. If not specified, the QoS settings are not changed.
- **total_size** (*int*) – New size of the volume in bytes. 1000000000 is equal to 1GB. Size is rounded up to the nearest 1MB. This parameter can only be used to increase the size of a volume.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **associate_with_qos_policy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.
- **qos_policy_id** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. The volume will not maintain any association with the policy; this is an alternate way to apply QoS settings to the volume. This parameter and the qos parameter cannot be specified at the same time.

access = <type 'str'>

account_id = <type 'int'>

associate_with_qos_policy = <type 'bool'>

attributes = <type 'dict'>

qos = <class 'solidfire.models.QoS'>

qos_policy_id = <type 'int'>

total_size = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**ModifyVolumeResult**(*volume=None*)

Bases: *solidfire.common.model.DataObject*

Parameters **volume** (Volume) – Object containing information about the newly modified volume.

volume = <class 'solidfire.models.Volume'>

class solidfire.models.**ModifyVolumesRequest**(*volume_ids*, *account_id=None*, *access=None*, *qos=None*, *total_size=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

ModifyVolumes allows you to configure up to 500 existing volumes at one time. Changes take place immediately. If ModifyVolumes fails to modify any of the specified volumes, none of the specified volumes are changed. If you do not specify QoS values when you modify volumes, the QoS values for each volume remain unchanged. You can retrieve default QoS values for a newly created volume by running the GetDefaultQoS method. When you need to increase the size of volumes that are being replicated, do so in the following order to prevent replication errors:

Increase the size of the "Replication Target" volume. Increase the size of the source or "Read / Write" volume.

Recommend that both the target and source volumes be the same size. NOTE: If you change access status to locked or replicationTarget all existing iSCSI connections are terminated.

Parameters

- **volume_ids** (*int*) – [required] A list of volumeIDs for the volumes to be modified.
- **account_id** (*int*) – AccountID to which the volume is reassigned. If none is specified, the previous account name is used.
- **access** (*str*) – Access allowed for the volume. Possible values:readOnly: Only read operations are allowed.readWrite: Reads and writes are allowed.locked: No reads or writes are allowed.If not specified, the access value does not change.replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked.If a value is not specified, the access value does not change.
- **qos** (QoS) – New quality of service settings for this volume.If not specified, the QoS settings are not changed.
- **total_size** (*int*) – New size of the volume in bytes. 1000000000 is equal to 1GB. Size is rounded up to the nearest 1MB in size. This parameter can only be used to increase the size of a volume.
- **associate_with_qos_policy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.
- **qos_policy_id** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. This parameter is mutually exclusive with the qos parameter.

- **attributes** (*dict*) – List of name/value pairs in JSON object format.

access = <type 'str'>

account_id = <type 'int'>

associate_with_qos_policy = <type 'bool'>

attributes = <type 'dict'>

qos = <class 'solidfire.models.QoS'>

qos_policy_id = <type 'int'>

total_size = <type 'int'>

volume_ids = <type 'int[]'>

class solidfire.models.**ModifyVolumesResult**(*volumes*, *qos=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **volumes** (Volume) – [required]
- **qos** (QoS) –

qos = <class 'solidfire.models.QoS'>

volumes = <class 'solidfire.models.Volume[]'>

class solidfire.models.**Network**(*bond10_g=None*, *bond1_g=None*, *eth0=None*, *eth1=None*, *eth2=None*, *eth3=None*, *eth4=None*, *eth5=None*, *lo=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **bond10_g** (NetworkConfig) –
- **bond1_g** (NetworkConfig) –
- **eth0** (NetworkConfig) –
- **eth1** (NetworkConfig) –
- **eth2** (NetworkConfig) –
- **eth3** (NetworkConfig) –
- **eth4** (NetworkConfig) –
- **eth5** (NetworkConfig) –
- **lo** (NetworkConfig) –

bond10_g = <class 'solidfire.models.NetworkConfig'>

bond1_g = <class 'solidfire.models.NetworkConfig'>

eth0 = <class 'solidfire.models.NetworkConfig'>

eth1 = <class 'solidfire.models.NetworkConfig'>

eth2 = <class 'solidfire.models.NetworkConfig'>

eth3 = <class 'solidfire.models.NetworkConfig'>

eth4 = <class 'solidfire.models.NetworkConfig'>

eth5 = <class 'solidfire.models.NetworkConfig'>

lo = <class 'solidfire.models.NetworkConfig'>

class solidfire.models.**NetworkConfig**(*_default=None*, *bond_master=None*, *virtual_network_tag=None*, *address=None*, *auto=None*, *bond_downdelay=None*, *bond_fail_over_mac=None*, *bond_primary_reselect=None*, *bond_lacp_rate=None*, *bond_miimon=None*, *bond_mode=None*, *bond_slaves=None*, *bond_updelay=None*, *dns_nameservers=None*, *dns_search=None*, *family=None*, *gateway=None*, *mac_address=None*, *mac_address_permanent=None*, *method=None*, *mtu=None*, *netmask=None*, *network=None*, *physical=None*, *routes=None*, *status=None*, *symmetric_route_rules=None*, *up_and_running=None*, *bond_xmit_hash_policy=None*, *bond_ad_num_ports=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **_default** (*bool*) –
- **bond_master** (*str*) –
- **virtual_network_tag** (*str*) –
- **address** (*str*) –
- **auto** (*bool*) –
- **bond_downdelay** (*str*) –
- **bond_fail_over_mac** (*str*) –
- **bond_primary_reselect** (*str*) –
- **bond_lacp_rate** (*str*) –
- **bond_miimon** (*str*) –
- **bond_mode** (*str*) –
- **bond_slaves** (*str*) –
- **bond_updelay** (*str*) –
- **dns_nameservers** (*str*) –
- **dns_search** (*str*) –
- **family** (*str*) –
- **gateway** (*str*) –
- **mac_address** (*str*) –
- **mac_address_permanent** (*str*) –
- **method** (*str*) –
- **mtu** (*str*) –
- **netmask** (*str*) –
- **network** (*str*) –
- **physical** (PhysicalAdapter) –

- **routes** (*dict*) –
- **status** (*str*) –
- **symmetric_route_rules** (*str*) –
- **up_and_running** (*bool*) –
- **bond_xmit_hash_policy** (*str*) –
- **bond_ad_num_ports** (*str*) –

address = <type 'str'>

auto = <type 'bool'>

bond_ad_num_ports = <type 'str'>

bond_downdelay = <type 'str'>

bond_fail_over_mac = <type 'str'>

bond_lacp_rate = <type 'str'>

bond_master = <type 'str'>

bond_miimon = <type 'str'>

bond_mode = <type 'str'>

bond_primary_reselect = <type 'str'>

bond_slaves = <type 'str'>

bond_updelay = <type 'str'>

bond_xmit_hash_policy = <type 'str'>

dns_nameservers = <type 'str'>

dns_search = <type 'str'>

family = <type 'str'>

gateway = <type 'str'>

mac_address = <type 'str'>

mac_address_permanent = <type 'str'>

method = <type 'str'>

mtu = <type 'str'>

netmask = <type 'str'>

network = <type 'str'>

physical = <class 'solidfire.models.PhysicalAdapter'>

routes = <type 'dict[]'>

status = <type 'str'>

symmetric_route_rules = <type 'str[]'>

up_and_running = <type 'bool'>

virtual_network_tag = <type 'str'>

class solidfire.models.**NetworkConfigParams**(*_default=None*, *bond_master=None*, *virtual_network_tag=None*, *address=None*, *auto=None*, *bond_downdelay=None*, *bond_fail_over_mac=None*, *bond_primary_reselect=None*, *bond_lacp_rate=None*, *bond_miimon=None*, *bond_mode=None*, *bond_slaves=None*, *bond_updelay=None*, *dns_nameservers=None*, *dns_search=None*, *family=None*, *gateway=None*, *mac_address=None*, *mac_address_permanent=None*, *method=None*, *mtu=None*, *netmask=None*, *network=None*, *physical=None*, *routes=None*, *status=None*, *symmetric_route_rules=None*, *up_and_running=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **_default** (*bool*) –
- **bond_master** (*str*) –
- **virtual_network_tag** (*str*) –
- **address** (*str*) –
- **auto** (*bool*) –
- **bond_downdelay** (*str*) –
- **bond_fail_over_mac** (*str*) –
- **bond_primary_reselect** (*str*) –
- **bond_lacp_rate** (*str*) –
- **bond_miimon** (*str*) –
- **bond_mode** (*str*) –
- **bond_slaves** (*str*) –
- **bond_updelay** (*str*) –
- **dns_nameservers** (*str*) –
- **dns_search** (*str*) –
- **family** (*str*) –
- **gateway** (*str*) –
- **mac_address** (*str*) –
- **mac_address_permanent** (*str*) –
- **method** (*str*) –
- **mtu** (*str*) –
- **netmask** (*str*) –
- **network** (*str*) –
- **physical** (PhysicalAdapter) –

- **routes** (*dict*) –
- **status** (*str*) –
- **symmetric_route_rules** (*str*) –
- **up_and_running** (*bool*) –

address = <type 'str'>

auto = <type 'bool'>

bond_downdelay = <type 'str'>

bond_fail_over_mac = <type 'str'>

bond_lacp_rate = <type 'str'>

bond_master = <type 'str'>

bond_miimon = <type 'str'>

bond_mode = <type 'str'>

bond_primary_reselect = <type 'str'>

bond_slaves = <type 'str'>

bond_updelay = <type 'str'>

dns_nameservers = <type 'str'>

dns_search = <type 'str'>

family = <type 'str'>

gateway = <type 'str'>

mac_address = <type 'str'>

mac_address_permanent = <type 'str'>

method = <type 'str'>

mtu = <type 'str'>

netmask = <type 'str'>

network = <type 'str'>

physical = <class 'solidfire.models.PhysicalAdapter'>

routes = <type 'dict[]'>

status = <type 'str'>

symmetric_route_rules = <type 'str[]'>

up_and_running = <type 'bool'>

virtual_network_tag = <type 'str'>

class solidfire.models.**NetworkInterface**(*address*, *broadcast*, *mac_address*, *mtu*, *name*, *netmask*, *status*, *type*, *virtual_network_tag*, *namespace=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **address** (*str*) – [required] IP address of the network.

- **broadcast** (*str*) – [required] Address of NTP broadcast.
- **mac_address** (*str*) – [required] Address used to configure the interface.
- **mtu** (*int*) – [required] Packet size on the network interface.
- **name** (*str*) – [required] Name of the network interface.
- **netmask** (*str*) – [required] Netmask for the network interface.
- **status** (*str*) – [required] Status of the network.
- **type** (*str*) – [required] The type of network, ie, BondMaster.
- **virtual_network_tag** (*int*) – [required] Virtual Network Tag if on virtual network.
- **namespace** (*bool*) –

address = <type 'str'>

broadcast = <type 'str'>

mac_address = <type 'str'>

mtu = <type 'int'>

name = <type 'str'>

namespace = <type 'bool'>

netmask = <type 'str'>

status = <type 'str'>

type = <type 'str'>

virtual_network_tag = <type 'int'>

class solidfire.models.**NetworkParams**(*bond10_g=None*, *bond1_g=None*, *eth0=None*, *eth1=None*, *eth2=None*, *eth3=None*, *lo=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **bond10_g** (NetworkConfigParams) –
- **bond1_g** (NetworkConfigParams) –
- **eth0** (NetworkConfigParams) –
- **eth1** (NetworkConfigParams) –
- **eth2** (NetworkConfigParams) –
- **eth3** (NetworkConfigParams) –
- **lo** (NetworkConfigParams) –

bond10_g = <class 'solidfire.models.NetworkConfigParams'>

bond1_g = <class 'solidfire.models.NetworkConfigParams'>

eth0 = <class 'solidfire.models.NetworkConfigParams'>

eth1 = <class 'solidfire.models.NetworkConfigParams'>

eth2 = <class 'solidfire.models.NetworkConfigParams'>

eth3 = <class 'solidfire.models.NetworkConfigParams'>

lo = <class 'solidfire.models.NetworkConfigParams'>

class solidfire.models.**NewDrive**(*drive_id*, *type=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **drive_id** (*int*) – [required] A unique identifier for this drive.
- **type** (*str*) – block or slice

drive_id = <type 'int'>

type = <type 'str'>

class solidfire.models.**Node**(*node_id*, *associated_master_service_id*, *associated_fservice_id*, *name*, *platform_info*, *software_version*, *cip*, *cipi*, *mip*, *mipi*, *sip*, *sipi*, *uuid*, *virtual_networks*, *attributes*, *fibre_channel_target_port_group=None*, *node_slot=None*)

Bases: *solidfire.common.model.DataObject*

A node refers to an individual machine in a cluster. Each active node hosts a master service, which is responsible for managing the drives and other services on its node. After a node is made active, its drives will become available for addition to the cluster.

Parameters

- **node_id** (*int*) – [required] The unique identifier for this node.
- **associated_master_service_id** (*int*) – [required] The master service responsible for controlling other services on this node.
- **associated_fservice_id** (*int*) – [required]
- **fibre_channel_target_port_group** (*int*) –
- **name** (*str*) – [required]
- **platform_info** (Platform) – [required] Information about the platform this node is.
- **software_version** (*str*) – [required] The version of SolidFire software this node is currently running.
- **cip** (*str*) – [required] IP address used for both intra- and inter-cluster communication.
- **cipi** (*str*) – [required] The machine's name for the "cip" interface.
- **mip** (*str*) – [required] IP address used for cluster management (hosting the API and web site).
- **mipi** (*str*) – [required] The machine's name for the "mip" interface.
- **sip** (*str*) – [required] IP address used for iSCSI traffic.
- **sipi** (*str*) – [required] The machine's name for the "sip" interface.
- **uuid** (*UUID*) – [required] UUID of node.
- **virtual_networks** (VirtualNetworkAddress) – [required]
- **attributes** (*dict*) – [required]
- **node_slot** (*str*) –

associated_fservice_id = <type 'int'>

associated_master_service_id = <type 'int'>

attributes = <type 'dict'>

cip = <type 'str'>

cipi = <type 'str'>

fibre_channel_target_port_group = <type 'int'>

mip = <type 'str'>

mipi = <type 'str'>

name = <type 'str'>

node_id = <type 'int'>

node_slot = <type 'str'>

platform_info = <class 'solidfire.models.Platform'>

sip = <type 'str'>

sipi = <type 'str'>

software_version = <type 'str'>

uuid = <class 'uuid.UUID'>

virtual_networks = <class 'solidfire.models.VirtualNetworkAddress[]'>

class solidfire.models.**NodeDriveHardware**(*node_id*, *result*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) – [required]
- **result** (DrivesHardware) – [required]

node_id = <type 'int'>

result = <class 'solidfire.models.DrivesHardware'>

class solidfire.models.**NodeStateInfo**(*cluster*, *state*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster** (*str*) – [required] Name of the cluster.
- **state** (*str*) – [required] Available: Node has not been configured with a cluster name.
Pending: Node is pending for a specific named cluster and can be added.
Active: Node is active and a member of a cluster and may not be added to another cluster.

cluster = <type 'str'>

state = <type 'str'>

class solidfire.models.**NodeStateResult**(*node_id*, *result=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **node_id** (*int*) – [required] ID of the node.
- **result** (NodeStateInfo) – NodeStateInfo object.

node_id = <type 'int'>

result = <class 'solidfire.models.NodeStateInfo'>

class solidfire.models.**NodeStatsInfo**(*c_bytes_in*, *c_bytes_out*, *count*, *cpu*, *cpu_total*, *m_bytes_in*, *m_bytes_out*, *network_utilization_cluster*, *network_utilization_storage*, *node_id*, *read_ops*, *read_latency_usec_total*, *s_bytes_in*, *s_bytes_out*, *timestamp*, *used_memory*, *write_latency_usec_total*, *write_ops*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **c_bytes_in** (*int*) – [required] Bytes in on the cluster interface.
- **c_bytes_out** (*int*) – [required] Bytes out on the cluster interface.
- **count** (*int*) – [required]
- **cpu** (*int*) – [required] CPU Usage %
- **cpu_total** (*int*) – [required] CPU Total
- **m_bytes_in** (*int*) – [required] Bytes in on the management interface.
- **m_bytes_out** (*int*) – [required] Bytes out on the management interface.
- **network_utilization_cluster** (*int*) – [required] Network interface utilization (in %) for the cluster network interface.
- **network_utilization_storage** (*int*) – [required] Network interface utilization (in %) for the storage network interface.
- **node_id** (*int*) – [required]
- **read_ops** (*int*) – [required] Read Operations.
- **read_latency_usec_total** (*int*) – [required]
- **s_bytes_in** (*int*) – [required] Bytes in on the storage interface.
- **s_bytes_out** (*int*) – [required] Bytes out on the storage interface.
- **timestamp** (*str*) – [required] Current time in UTC format ISO 8691 date string.
- **used_memory** (*int*) – [required] Total memory usage in bytes.
- **write_latency_usec_total** (*int*) – [required]
- **write_ops** (*int*) – [required] Write Operations

c_bytes_in = <type 'int'>

c_bytes_out = <type 'int'>

count = <type 'int'>

cpu = <type 'int'>

cpu_total = <type 'int'>

m_bytes_in = <type 'int'>

m_bytes_out = <type 'int'>

network_utilization_cluster = <type 'int'>

network_utilization_storage = <type 'int'>

node_id = <type 'int'>

read_latency_usec_total = <type 'int'>

read_ops = <type 'int'>

s_bytes_in = <type 'int'>

s_bytes_out = <type 'int'>

timestamp = <type 'str'>

used_memory = <type 'int'>

write_latency_usec_total = <type 'int'>

write_ops = <type 'int'>

class solidfire.models.**NodeStatsNodes**(*nodes*)

Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (NodeStatsInfo) – [required] Node activity information for a single node.

nodes = <class 'solidfire.models.NodeStatsInfo[]'>

class solidfire.models.**NodeWaitingToJoin**(*version*, *compatible*, *name=None*, *node_id=None*, *pending_node_id=None*, *mip=None*, *cip=None*, *sip=None*, *chassis_type=None*, *hostname=None*, *node_type=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **name** (*str*) –
- **version** (*str*) – [required]
- **node_id** (*int*) –
- **pending_node_id** (*int*) –
- **mip** (*str*) –
- **cip** (*str*) –
- **sip** (*str*) –
- **compatible** (*bool*) – [required]
- **chassis_type** (*str*) –
- **hostname** (*str*) –
- **node_type** (*str*) –

chassis_type = <type 'str'>

cip = <type 'str'>

compatible = <type 'bool'>

hostname = <type 'str'>

mip = <type 'str'>

name = <type 'str'>

node_id = <type 'int'>

node_type = <type 'str'>

pending_node_id = <type 'int'>

sip = <type 'str'>

version = <type 'str'>

class solidfire.models.**Origin**(*signature*, *contract_date*, *contract_name*, *contract_quantity*, *contract_type*, *integrator*, *location*, *organization*, *type*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **signature** (Signature) – [required]
- **contract_date** (*str*) – [required]
- **contract_name** (*str*) – [required]
- **contract_quantity** (*int*) – [required]
- **contract_type** (*str*) – [required]
- **integrator** (*str*) – [required]
- **location** (*str*) – [required]
- **organization** (*str*) – [required]
- **type** (*str*) – [required]

contract_date = <type 'str'>

contract_name = <type 'str'>

contract_quantity = <type 'int'>

contract_type = <type 'str'>

integrator = <type 'str'>

location = <type 'str'>

organization = <type 'str'>

signature = <class 'solidfire.models.Signature'>

type = <type 'str'>

class solidfire.models.**PairedCluster**(*cluster_name*, *cluster_pair_id*, *cluster_pair_uuid*, *latency*, *mvip*, *status*, *version*, *cluster_uuid=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster_name** (*str*) – [required] Name of the other cluster in the pair
- **cluster_pair_id** (*int*) – [required] Unique ID given to each cluster in the pair.
- **cluster_pair_uuid** (*UUID*) – [required] Universally unique identifier.
- **latency** (*int*) – [required] Number, in milliseconds, of latency between clusters.
- **mvip** (*str*) – [required] IP of the management connection for paired clusters.
- **status** (*str*) – [required] Can be one of the following: Connected Misconfigured Disconnected
- **version** (*str*) – [required] The Element OS version of the other cluster in the pair.
- **cluster_uuid** (*str*) – The cluster UUID

cluster_name = <type 'str'>

cluster_pair_id = <type 'int'>

cluster_pair_uuid = <class 'uuid.UUID'>

cluster_uuid = <type 'str'>

latency = <type 'int'>

mvip = <type 'str'>

status = <type 'str'>

version = <type 'str'>

class solidfire.models.**PendingActiveNode**(*active_node_key*, *assigned_node_id*, *async_handle*, *cip*, *mip*, *pending_node_id*, *platform_info*, *sip*, *software_version*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **active_node_key** (*str*) – [required]
- **assigned_node_id** (*int*) – [required]
- **async_handle** (*int*) – [required]
- **cip** (*str*) – [required]
- **mip** (*str*) – [required]
- **pending_node_id** (*int*) – [required]
- **platform_info** (Platform) – [required]
- **sip** (*str*) – [required]
- **software_version** (*str*) – [required]

active_node_key = <type 'str'>

assigned_node_id = <type 'int'>

async_handle = <type 'int'>

cip = <type 'str'>

mip = <type 'str'>

pending_node_id = <type 'int'>

platform_info = <class 'solidfire.models.Platform'>

sip = <type 'str'>

software_version = <type 'str'>

class solidfire.models.**PendingNode**(*pending_node_id*, *assigned_node_id*, *name*, *compatible*, *platform_info*, *cip*, *cipi*, *mip*, *mipi*, *sip*, *sipi*, *software_version*, *uuid*, *node_slot=None*)

Bases: *solidfire.common.model.DataObject*

A "pending node" is one that has not yet joined the cluster. It can be added to a cluster using the AddNode method.

Parameters

- **pending_node_id** (*int*) – [required]
- **assigned_node_id** (*int*) – [required]

- **name** (*str*) – [required] The host name for this node.
- **compatible** (*bool*) – [required]
- **platform_info** (Platform) – [required] Information about the platform this node is.
- **cip** (*str*) – [required] IP address used for both intra- and inter-cluster communication.
- **cipi** (*str*) – [required] The machine's name for the "cip" interface.
- **mip** (*str*) – [required] IP address used for cluster management (hosting the API and web site).
- **mipi** (*str*) – [required] The machine's name for the "mip" interface.
- **sip** (*str*) – [required] IP address used for iSCSI traffic.
- **sipi** (*str*) – [required] The machine's name for the "sip" interface.
- **software_version** (*str*) – [required] The version of SolidFire software this node is currently running.
- **uuid** (*UUID*) – [required] UUID of node.
- **node_slot** (*str*) – UUID of node.

assigned_node_id = <type 'int'>

cip = <type 'str'>

cipi = <type 'str'>

compatible = <type 'bool'>

mip = <type 'str'>

mipi = <type 'str'>

name = <type 'str'>

node_slot = <type 'str'>

pending_node_id = <type 'int'>

platform_info = <class 'solidfire.models.Platform'>

sip = <type 'str'>

sipi = <type 'str'>

software_version = <type 'str'>

uuid = <class 'uuid.UUID'>

class solidfire.models.**PendingOperation**(*pending*, *operation*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **pending** (*bool*) – [required] true: operation is still in progress. false: operation is no integerer in progress.
- **operation** (*str*) – [required] Name of operation that is in progress or has completed.

operation = <type 'str'>

pending = <type 'bool'>

class solidfire.models.**PhysicalAdapter**(*address=None*, *mac_address=None*, *mac_address_permanent=None*, *mtu=None*, *netmask=None*, *network=None*, *up_and_running=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **address** (*str*) –
- **mac_address** (*str*) –
- **mac_address_permanent** (*str*) –
- **mtu** (*str*) –
- **netmask** (*str*) –
- **network** (*str*) –
- **up_and_running** (*bool*) –

address = <type 'str'>

mac_address = <type 'str'>

mac_address_permanent = <type 'str'>

mtu = <type 'str'>

netmask = <type 'str'>

network = <type 'str'>

up_and_running = <type 'bool'>

class solidfire.models.**Platform**(*node_type*, *chassis_type*, *cpu_model*, *node_memory_gb*, *platform_config_version=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **node_type** (*str*) – [required] SolidFire's name for this platform.
- **chassis_type** (*str*) – [required] Name of the chassis (example: "R620").
- **cpu_model** (*str*) – [required] The model of CPU used on this platform.
- **node_memory_gb** (*int*) – [required] The amount of memory on this platform in GiB.
- **platform_config_version** (*str*) –

chassis_type = <type 'str'>

cpu_model = <type 'str'>

node_memory_gb = <type 'int'>

node_type = <type 'str'>

platform_config_version = <type 'str'>

class solidfire.models.**ProtocolEndpoint**(*protocol_endpoint_id*, *protocol_endpoint_state*, *provider_type*, *primary_provider_id*, *secondary_provider_id*, *scsi_naadevice_id*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **protocol_endpoint_id** (*UUID*) – [required]
- **protocol_endpoint_state** (*str*) – [required]
- **provider_type** (*str*) – [required]
- **primary_provider_id** (*int*) – [required]
- **secondary_provider_id** (*int*) – [required]
- **scsi_naadevice_id** (*str*) – [required]

primary_provider_id = <type 'int'>

protocol_endpoint_id = <class 'uuid.UUID'>

protocol_endpoint_state = <type 'str'>

provider_type = <type 'str'>

scsi_naadevice_id = <type 'str'>

secondary_provider_id = <type 'int'>

class solidfire.models.**PurgeDeletedVolumeRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

PurgeDeletedVolume immediately and permanently purges a volume that has been deleted. You must delete a volume using DeleteVolume before it can be purged. Volumes are purged automatically after a period of time, so usage of this method is not typically required.

Parameters **volume_id** (*int*) – [required] The ID of the volume to be purged.

volume_id = <type 'int'>

class solidfire.models.**PurgeDeletedVolumeResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**PurgeDeletedVolumesRequest**(*volume_ids=None*, *account_ids=None*, *volume_access_group_ids=None*)
Bases: *solidfire.common.model.DataObject*

PurgeDeletedVolumes immediately and permanently purges volumes that have been deleted. You can use this method to purge up to 500 volumes at one time. You must delete volumes using DeleteVolumes before they can be purged. Volumes are purged by the system automatically after a period of time, so usage of this method is not typically required.

Parameters

- **volume_ids** (*int*) – A list of volumeIDs of volumes to be purged from the system.
- **account_ids** (*int*) – A list of accountIDs. All of the volumes from all of the specified accounts are purged from the system.
- **volume_access_group_ids** (*int*) – A list of volumeAccessGroupIDs. All of the volumes from all of the specified Volume Access Groups are purged from the system.

account_ids = <type 'int[]'>

volume_access_group_ids = <type 'int[]'>

volume_ids = <type 'int[]'>

class solidfire.models.**PurgeDeletedVolumesResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**QoS**(*min_iops=None*, *max_iops=None*, *burst_iops=None*, *burst_time=None*, *curve=None*)

Bases: *solidfire.common.model.DataObject*

Quality of Service (QoS) values are used on SolidFire volumes to provision performance expectations. Minimum, maximum and burst QoS values can be set within the ranges specified in the QoS table below.

Volumes created without specified QoS values are created with the Default values listed below. Default values can be found by running the GetDefaultQoS method.

minIOPS Min: 100/50 (v7.0/v8.0), Default: 100, Max: 15,000 maxIOPS Min: 100/50 (v7.0/v8.0), Default: 15,000, Max: 100,000 burstIOPS Min: 100/50 (v7.0/v8.0), Default: 15,000, Max: 100,000

Parameters

- **min_iops** (*int*) – Desired minimum 4KB IOPS to guarantee. The allowed IOPS will only drop below this level if all volumes have been capped at their minimum IOPS value and there is still insufficient performance capacity.
- **max_iops** (*int*) – Desired maximum 4KB IOPS allowed over an extended period of time.
- **burst_iops** (*int*) – Maximum "peak" 4KB IOPS allowed for short periods of time. Allows for bursts of I/O activity over the normal max IOPS value.
- **burst_time** (*int*) – The length of time burst IOPS is allowed. The value returned is represented in time units of seconds. Note: this value is calculated by the system based on IOPS set for QoS.
- **curve** (*dict*) – The curve is a set of key-value pairs. The keys are I/O sizes in bytes. The values represent the cost performing an IOP at a specific I/O size. The curve is calculated relative to a 4096 byte operation set at 100 IOPS.

burst_iops = <type 'int'>

burst_time = <type 'int'>

curve = <type 'dict'>

max_iops = <type 'int'>

min_iops = <type 'int'>

class solidfire.models.**QoSPolicy**(*qos_policy_id*, *name*, *volume_ids*, *qos*)

Bases: *solidfire.common.model.DataObject*

The QoSPolicy object contains information about a QoS policy on the cluster.

Parameters

- **qos_policy_id** (*int*) – [required] A unique integer identifier for the QoSPolicy auto-assigned by the SolidFire cluster.
- **name** (*str*) – [required] The name of the QoS policy. For example: gold, platinum, or silver.
- **volume_ids** (*int*) – [required] A list of volumes associated with this policy.
- **qos** (VolumeQOS) – [required] Quality of service settings for this volume.

name = <type 'str'>

qos = <class 'solidfire.models.VolumeQOS'>

qos_policy_id = <type 'int'>

volume_ids = <type 'int[]'>

class solidfire.models.**RemoteReplication**(*mode*, *pause_limit*, *remote_service_id*, *resume_details*, *snapshot_replication*, *state*, *state_details*)

Bases: *solidfire.common.model.DataObject*

Details on the volume replication.

Parameters

- **mode** (*str*) – [required] Volume replication mode. Possible values: Async: Writes are acknowledged when they complete locally. The cluster does not wait for writes to be replicated to the target cluster. Sync: Source acknowledges write when the data is stored locally and on the remote cluster. SnapshotsOnly: Only snapshots created on the source cluster will be replicated. Active writes from the source volume will not be replicated.
- **pause_limit** (*int*) – [required] The number of occurring write ops before auto-pausing, on a per volume pair level.
- **remote_service_id** (*int*) – [required] The remote slice service ID.
- **resume_details** (*str*) – [required] Reserved for future use.
- **snapshot_replication** (SnapshotReplication) – [required] The details of snapshot replication.
- **state** (*str*) – [required] The state of the volume replication.
- **state_details** (*str*) – [required] Reserved for future use.

mode = <type 'str'>

pause_limit = <type 'int'>

remote_service_id = <type 'int'>

resume_details = <type 'str'>

snapshot_replication = <class 'solidfire.models.SnapshotReplication'>

state = <type 'str'>

state_details = <type 'str'>

class solidfire.models.**RemoveAccountRequest**(*account_id*)

Bases: *solidfire.common.model.DataObject*

RemoveAccount enables you to remove an existing account. You must delete and purge all volumes associated with the account using DeleteVolume before you can remove the account. If volumes on the account are still pending deletion, you cannot use RemoveAccount to remove the account.

Parameters **account_id** (*int*) – [required] Specifies the AccountID for the account to be removed.

account_id = <type 'int'>

class solidfire.models.**RemoveAccountResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveBackupTargetRequest**(*backup_target_id*)

Bases: *solidfire.common.model.DataObject*

RemoveBackupTarget allows you to delete backup targets.

Parameters **backup_target_id** (*int*) – [required] The unique target ID of the target to remove.

backup_target_id = <type 'int'>

class solidfire.models.**RemoveBackupTargetResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveClusterAdminRequest**(*cluster_admin_id*)
Bases: *solidfire.common.model.DataObject*

You can use RemoveClusterAdmin to remove a Cluster Admin. You cannot remove the administrator cluster admin account.

Parameters **cluster_admin_id** (*int*) – [required] ClusterAdminID for the cluster admin to remove.

cluster_admin_id = <type 'int'>

class solidfire.models.**RemoveClusterAdminResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveClusterPairRequest**(*cluster_pair_id*)
Bases: *solidfire.common.model.DataObject*

You can use the RemoveClusterPair method to close the open connections between two paired clusters. Note: Before you remove a cluster pair, you must first remove all volume pairing to the clusters with the "RemoveVolumePair" API method.

Parameters **cluster_pair_id** (*int*) – [required] Unique identifier used to pair two clusters.

cluster_pair_id = <type 'int'>

class solidfire.models.**RemoveClusterPairResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveDrivesRequest**(*drives*, *force_during_upgrade=None*)
Bases: *solidfire.common.model.DataObject*

You can use RemoveDrives to proactively remove drives that are part of the cluster. You might want to use this method when reducing cluster capacity or preparing to replace drives nearing the end of their service life. Any data on the drives is removed and migrated to other drives in the cluster before the drive is removed from the cluster. This is an asynchronous method. Depending on the total capacity of the drives being removed, it might take several minutes to migrate all of the data. Use the GetAsyncResult method to check the status of the remove operation. When removing multiple drives, use a single RemoveDrives method call rather than multiple individual methods with a single drive each. This reduces the amount of data balancing that must occur to even stabilize the storage load on the cluster. You can also remove drives with a "failed" status using RemoveDrives. When you remove a drive with a "failed" status it is not returned to an "available" or active status. The drive is unavailable for use in the cluster. Use the ListDrives method to obtain the driveIDs for the drives you want to remove.

Parameters

- **drives** (*int*) – [required] List of driveIDs to remove from the cluster.
- **force_during_upgrade** (*bool*) – If you want to remove a drive during upgrade, this must be set to true.

drives = <type 'int[]'>

force_during_upgrade = <type 'bool'>

class solidfire.models.**RemoveInitiatorsFromVolumeAccessGroupRequest**(*volume_access_group_id*, *initiators*, *delete_orphan_initiators=None*)
Bases: *solidfire.common.model.DataObject*

RemoveInitiatorsFromVolumeAccessGroup enables you to remove initiators from a specified volume access group.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group from which the initiators are removed.
- **initiators** (*str*) – [required] The list of initiators to remove from the volume access group.
- **delete_orphan_initiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.

delete_orphan_initiators = <type 'bool'>

initiators = <type 'str[]'>

volume_access_group_id = <type 'int'>

class solidfire.models.**RemoveNodeSSLCertificateResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveNodesRequest**(*nodes*)
Bases: *solidfire.common.model.DataObject*

You can use RemoveNodes to remove one or more nodes that should no longer participate in the cluster. Before removing a node, you must remove all drives the node contains using the RemoveDrives method. You cannot remove a node until the RemoveDrives process has completed and all data has been migrated away from the node. After you remove a node, it registers itself as a pending node. You can add the node again or shut it down (shutting the node down removes it from the Pending Node list).

Parameters **nodes** (*int*) – [required] List of NodeIDs for the nodes to be removed.

nodes = <type 'int[]'>

class solidfire.models.**RemoveNodesResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveSSLCertificateResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveVirtualNetworkRequest**(*virtual_network_id=None*, *virtual_network_tag=None*)
Bases: *solidfire.common.model.DataObject*

RemoveVirtualNetwork enables you to remove a previously added virtual network. Note: This method requires either the virtualNetworkID or the virtualNetworkTag as a parameter, but not both.

Parameters

- **virtual_network_id** (*int*) – Network ID that identifies the virtual network to remove.
- **virtual_network_tag** (*int*) – Network tag that identifies the virtual network to remove.

virtual_network_id = <type 'int'>

virtual_network_tag = <type 'int'>

class solidfire.models.**RemoveVirtualNetworkResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveVolumePairRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

RemoveVolumePair enables you to remove the remote pairing between two volumes. Use this method on both the source and target volumes that are paired together. When you remove the volume pairing information, data is no longer replicated to or from the volume.

Parameters **volume_id** (*int*) – [required] The ID of the volume on which to stop the replication process.

volume_id = <type 'int'>

class solidfire.models.**RemoveVolumePairResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RemoveVolumesFromVolumeAccessGroupRequest**(*volume_access_group_id*, *volumes*)
Bases: *solidfire.common.model.DataObject*

The RemoveVolumeFromVolumeAccessGroup method enables you to remove volumes from a volume access group.

Parameters

- **volume_access_group_id** (*int*) – [required] The ID of the volume access group to remove volumes from.
- **volumes** (*int*) – [required] The ID of the volume access group to remove volumes from.

volume_access_group_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.models.**ResetDriveDetails**(*drive*, *return_code*, *stderr*, *stdout*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **drive** (*str*) – [required] Drive name
- **return_code** (*int*) – [required]
- **stderr** (*str*) – [required]
- **stdout** (*str*) – [required]

drive = <type 'str'>

return_code = <type 'int'>

stderr = <type 'str'>

stdout = <type 'str'>

class solidfire.models.**ResetDrivesDetails**(*drives*)
Bases: *solidfire.common.model.DataObject*

Parameters **drives** (ResetDriveDetails) – [required] Details of a single drive that is being reset.

drives = <class 'solidfire.models.ResetDriveDetails[]'>

class solidfire.models.**ResetDrivesRequest**(*drives*, *force*)
Bases: *solidfire.common.model.DataObject*

ResetDrives enables you to proactively initialize drives and remove all data currently residing on a drive. The drive can then be reused in an existing node or used in an upgraded node. This method requires the force parameter to be included in the method call.

Parameters

- **drives** (*str*) – [required] List of device names (not driveIDs) to reset.
- **force** (*bool*) – [required] Required parameter to successfully reset a drive.

drives = <type 'str'>

force = <type 'bool'>

class solidfire.models.**ResetDrivesResult**(*details*, *duration=None*, *result=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (ResetDrivesDetails) – [required] Details of drives that are being reset.
- **duration** (*str*) –
- **result** (*str*) –

details = <class 'solidfire.models.ResetDrivesDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**ResetNodeDetails**(*rtfi_info*)
Bases: *solidfire.common.model.DataObject*

Parameters **rtfi_info** (RtfiInfo) – [required]

rtfi_info = <class 'solidfire.models.RtfiInfo'>

class solidfire.models.**ResetNodeRequest**(*build*, *force*, *reboot=None*, *options=None*)
Bases: *solidfire.common.model.DataObject*

The ResetNode API method enables you to reset a node to the factory settings. All data, packages (software upgrades, and so on), configurations, and log files are deleted from the node when you call this method. However, network settings for the node are preserved during this operation. Nodes that are participating in a cluster cannot be reset to the factory settings. The ResetNode API can only be used on nodes that are in an "Available" state. It cannot be used on nodes that are "Active" in a cluster, or in a "Pending" state. Caution: This method clears any data that is on the node. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters

- **build** (*str*) – [required] Specifies the URL to a remote Element software image to which the node will be reset.
- **force** (*bool*) – [required] Required parameter to successfully reset the node.
- **reboot** (*bool*) – Set to true if you want to reboot the node.
- **options** (*str*) – Used to enter specifications for running the reset operation. Available options: 'edebug': '', 'sf_auto': '0', 'sf_bond_mode': 'ActivePassive', 'sf_check_hardware': '0', 'sf_disable_otpw': '0', 'sf_fa_host': '', 'sf_hostname': 'SF-FA18', 'sf_inplace': '1', 'sf_inplace_die_action': 'kexec', 'sf_inplace_safe': '0', 'sf_keep_cluster_config': '0', 'sf_keep_data': '0', 'sf_keep_hostname': '0', 'sf_keep_network_config': '0', 'sf_keep_paths': '/var/log/hardware.xml 'sf_max_archives': '5', 'sf_nvram_size': '', 'sf_oldroot':

'', 'sf_postinst_erase_root_drive': '0', 'sf_root_drive': '', 'sf_rtfi_cleanup_state': '', 'sf_secure_erase': '1', 'sf_secure_erase_retries': '5', 'sf_slice_size': '', 'sf_ssh_key': '1', 'sf_ssh_root': '1', 'sf_start_rtfi': '1', 'sf_status_httpserver': '1', 'sf_status_httpserver_stop_delay': '5m', 'sf_status_inject_failure': '', 'sf_status_json': '0', 'sf_support_host': 'sfsupport.solidfire.com', 'sf_test_hardware': '0', 'sf_upgrade': '0', 'sf_upgrade_firmware': '0', 'sf_upload_logs_url': ''

build = <type 'str'>

force = <type 'bool'>

options = <type 'str'>

reboot = <type 'bool'>

class solidfire.models.**ResetNodeResult**(*details=None*, *duration=None*, *result=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (ResetNodeDetails) –
- **duration** (*str*) –
- **result** (*str*) –

details = <class 'solidfire.models.ResetNodeDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**RestartNetworkingRequest**(*force*)

Bases: *solidfire.common.model.DataObject*

The RestartNetworking API method enables you to restart the networking services on a node. Warning: This method restarts all networking services on a node, causing temporary loss of networking connectivity. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **force** (*bool*) – [required] Required parameter to successfully reset the node.

force = <type 'bool'>

class solidfire.models.**RestartServicesRequest**(*force*, *service=None*, *action=None*)

Bases: *solidfire.common.model.DataObject*

The RestartServices API method enables you to restart the services on a node. Caution: This method causes temporary node services interruption. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters

- **force** (*bool*) – [required] Required parameter to successfully restart services on a node.
- **service** (*str*) – Service name to be restarted.
- **action** (*str*) – Action to perform on the service (start, stop, restart).

action = <type 'str'>

force = <type 'bool'>

service = <type 'str'>

class solidfire.models.**RestoreDeletedVolumeRequest**(*volume_id*)
Bases: *solidfire.common.model.DataObject*

RestoreDeletedVolume marks a deleted volume as active again. This action makes the volume immediately available for iSCSI connection.

Parameters **volume_id** (*int*) – [required] VolumeID of the deleted volume to be restored.

volume_id = <type 'int'>

class solidfire.models.**RestoreDeletedVolumeResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**RollbackToGroupSnapshotRequest**(*group_snapshot_id*, *save_current_state*, *name=None*, *attributes=None*)
Bases: *solidfire.common.model.DataObject*

RollbackToGroupSnapshot enables you to roll back all individual volumes in a snapshot group to each volume's individual snapshot. Note: Rolling back to a group snapshot creates a temporary snapshot of each volume within the group snapshot. Snapshots are allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5.

Parameters

- **group_snapshot_id** (*int*) – [required] Specifies the unique ID of the group snapshot.
- **save_current_state** (*bool*) – [required] Specifies whether to save an active volume image or delete it. Values are: true: The previous active volume image is kept. false: (default) The previous active volume image is deleted.
- **name** (*str*) – Name for the group snapshot of the volume's current state that is created if "saveCurrentState" is set to true. If you do not give a name, the name of the snapshots (group and individual volume) are set to a timestamp of the time that the rollback occurred.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

group_snapshot_id = <type 'int'>

name = <type 'str'>

save_current_state = <type 'bool'>

class solidfire.models.**RollbackToGroupSnapshotResult**(*group_snapshot=None*, *group_snapshot_id=None*, *members=None*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **group_snapshot** (GroupSnapshot) –
- **group_snapshot_id** (*int*) – Unique ID of the new group snapshot.
- **members** (GroupSnapshotMembers) – List of checksum, volumeIDs and snapshotIDs for each member of the group.

group_snapshot = <class 'solidfire.models.GroupSnapshot'>

group_snapshot_id = <type 'int'>

members = <class 'solidfire.models.GroupSnapshotMembers[]'>

class solidfire.models.**RollbackToSnapshotRequest**(*volume_id*, *snapshot_id*, *save_current_state*, *name=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

RollbackToSnapshot enables you to make an existing snapshot of the "active" volume image. This method creates a new snapshot from an existing snapshot. The new snapshot becomes "active" and the existing snapshot is preserved until you delete it. The previously "active" snapshot is deleted unless you set the parameter saveCurrentState to true. Note: Creating a snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5.

Parameters

- **volume_id** (*int*) – [required] VolumeID for the volume.
- **snapshot_id** (*int*) – [required] The ID of a previously created snapshot on the given volume.
- **save_current_state** (*bool*) – [required] Specifies whether to save an active volume image or delete it. Values are: true: The previous active volume image is kept. false: (default) The previous active volume image is deleted.
- **name** (*str*) – Name for the snapshot. If unspecified, the name of the snapshot being rolled back to is used with "- copy" appended to the end of the name.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

attributes = <type 'dict'>

name = <type 'str'>

save_current_state = <type 'bool'>

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**RollbackToSnapshotResult**(*snapshot=None*, *snapshot_id=None*, *checksum=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **snapshot** (Snapshot) –
- **snapshot_id** (*int*) – ID of the newly-created snapshot.
- **checksum** (*str*) – A string that represents the correct digits in the stored snapshot. This checksum can be used later to compare other snapshots to detect errors in the data.

checksum = <type 'str'>

snapshot = <class 'solidfire.models.Snapshot'>

snapshot_id = <type 'int'>

class solidfire.models.**RtfiInfo**(*generation*, *build*, *status_url_all*, *status_url_current*, *mipi=None*, *status_url_logfile=None*, *generation_next=None*, *mip=None*, *options=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **mipi** (*str*) –
- **generation** (*str*) – [required]

- **status_url_logfile** (*str*) –
- **build** (*str*) – [required]
- **status_url_all** (*str*) – [required]
- **generation_next** (*int*) –
- **mip** (*str*) –
- **status_url_current** (*str*) – [required]
- **options** (*dict*) –

build = <type 'str'>

generation = <type 'str'>

generation_next = <type 'int'>

mip = <type 'str'>

mipi = <type 'str'>

options = <type 'dict'>

status_url_all = <type 'str'>

status_url_current = <type 'str'>

status_url_logfile = <type 'str'>

class solidfire.models.**Schedule**(*schedule_info*, *name*, *frequency*, *last_run_time_started=None*, *has_error=None*, *run_next_interval=None*, *last_run_status=None*, *schedule_id=None*, *paused=None*, *to_be_deleted=None*, *starting_date=None*, *recurring=None*)

Bases: *solidfire.common.model.DataObject*

Schedule is an object containing information about each schedule created to autonomously make a snapshot of a volume. The return object includes information for all schedules. If scheduleID is used to identify a specific schedule then only information for that scheduleID is returned. Schedules information is returned with the API method, see ListSchedules on the SolidFire API guide page 245.

Parameters

- **last_run_time_started** (*str*) – Indicates the last time the schedule started n ISO 8601 date string. Valid values are: Success Failed
- **has_error** (*bool*) – Indicates whether or not the schedule has errors.
- **schedule_info** (ScheduleInfo) – [required] Includes the unique name given to the schedule, the retention period for the snapshot that was created, and the volume ID of the volume from which the snapshot was created.
- **run_next_interval** (*bool*) – Indicates whether or not the schedule will run the next time the scheduler is active. When set to "true", the schedule will run the next time the scheduler is active and then reset back to "false".
- **name** (*str*) – [required] Unique name assigned to the schedule.
- **last_run_status** (*str*) – Indicates the status of the last scheduled snapshot. Valid values are: Success Failed
- **schedule_id** (*int*) – Unique ID of the schedule
- **paused** (*bool*) – Indicates whether or not the schedule is paused.
- **to_be_deleted** (*bool*) – Indicates if the schedule is marked for deletion.

- **frequency** (Frequency) – [required] Indicates the frequency of the schedule occurrence. Set this to a type that inherits from Frequency. Valid types are: DayOfWeekFrequency DayOfMonthFrequency TimeIntervalFrequency
- **starting_date** (*str*) – Indicates the date the first time the schedule began of will begin. Formatted in UTC time.
- **recurring** (*bool*) – Indicates whether or not the schedule is recurring.

frequency = <class 'solidfire.models.Frequency'>

has_error = <type 'bool'>

last_run_status = <type 'str'>

last_run_time_started = <type 'str'>

name = <type 'str'>

paused = <type 'bool'>

recurring = <type 'bool'>

run_next_interval = <type 'bool'>

schedule_id = <type 'int'>

schedule_info = <class 'solidfire.models.ScheduleInfo'>

starting_date = <type 'str'>

to_be_deleted = <type 'bool'>

class solidfire.models.**ScheduleInfo**(*snapshot_name=None*, *enable_remote_replication=None*, *volume_ids=None*, *retention=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **snapshot_name** (*str*) – The snapshot name to be used.
- **enable_remote_replication** (*bool*) – Indicates if the snapshot should be included in remote replication.
- **volume_ids** (*int*) – A list of volume IDs to be included in the group snapshot.
- **retention** (*str*) – The amount of time the snapshot will be retained in HH:mm:ss.

enable_remote_replication = <type 'bool'>

retention = <type 'str'>

snapshot_name = <type 'str'>

volume_ids = <type 'int[]'>

class solidfire.models.**SecureEraseDrivesRequest**(*drives*)

Bases: *solidfire.common.model.DataObject*

SecureEraseDrives enables you to remove any residual data from drives that have a status of "available." You might want to use this method when replacing a drive nearing the end of its service life that contained sensitive data. This method uses a Security Erase Unit command to write a predetermined pattern to the drive and resets the encryption key on the drive. This asynchronous method might take up to two minutes to complete. You can use GetAsyncResult to check on the status of the secure erase operation. You can use the ListDrives method to obtain the driveIDs for the drives you want to secure erase.

Parameters **drives** (*int*) – [required] List of driveIDs to be secure erased.

drives = <type 'int[]'>

class solidfire.models.**Service**(*service_id*, *service_type*, *node_id*, *async_result_ids*, *first_time_startup*, *ipc_port*, *iscsi_port*, *status*, *started_drive_ids*, *drive_ids*, *associated_bv=None*, *associated_ts=None*, *associated_vs=None*, *drive_id=None*, *smart_ssd_write_enabled=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **service_id** (*int*) – [required] Unique identifier for this service.
- **service_type** (*str*) – [required]
- **node_id** (*int*) – [required] The node this service resides on.
- **associated_bv** (*int*) – This service's associated bulk volume service. This will only be set if the service type is a slice service.
- **associated_ts** (*int*) – This service's associated transport service. This will only be set if the service type is a slice service.
- **associated_vs** (*int*) – This service's associated volume service. This will only be set if the service type is a slice service.
- **async_result_ids** (*int*) – [required] The list of asynchronous jobs currently running for this service.
- **drive_id** (*int*) – If this service resides on a drive, the ID of that drive.
- **first_time_startup** (*bool*) – [required] Has this service started successfully? When a new drive is added to the system, the created service will initially have a value of false here. After the service has started, this value will be set to true. This can be used to check if the service has ever started.
- **ipc_port** (*int*) – [required] The port used for intra-cluster communication. This will be in the 4000-4100 range.
- **iscsi_port** (*int*) – [required] The port used for iSCSI traffic. This will only be set if the service type is a transport service.
- **status** (*str*) – [required]
- **started_drive_ids** (*int*) – [required]
- **drive_ids** (*int*) – [required]
- **smart_ssd_write_enabled** (*bool*) –

associated_bv = <type 'int'>

associated_ts = <type 'int'>

associated_vs = <type 'int'>

async_result_ids = <type 'int[]'>

drive_id = <type 'int'>

drive_ids = <type 'int[]'>

first_time_startup = <type 'bool'>

ipc_port = <type 'int'>

iscsi_port = <type 'int'>

node_id = <type 'int'>

service_id = <type 'int'>

service_type = <type 'str'>

smart_ssd_write_enabled = <type 'bool'>

started_drive_ids = <type 'int[]'>

status = <type 'str'>

class solidfire.models.**SetClusterConfigRequest**(*cluster*)
Bases: *solidfire.common.model.DataObject*

The SetClusterConfig API method enables you to set the configuration this node uses to communicate with the cluster it is associated with. To see the states in which these objects can be modified, see Cluster Object Attributes. To display the current cluster interface settings for a node, run the GetClusterConfig API method. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **cluster** (ClusterConfig) – [required] Objects that are changed for the cluster interface settings.

cluster = <class 'solidfire.models.ClusterConfig'>

class solidfire.models.**SetClusterConfigResult**(*cluster*)
Bases: *solidfire.common.model.DataObject*

Parameters **cluster** (ClusterConfig) – [required] Settings for the cluster. All new and current settings are returned.

cluster = <class 'solidfire.models.ClusterConfig'>

class solidfire.models.**SetConfigRequest**(*config*)
Bases: *solidfire.common.model.DataObject*

The SetConfig API method enables you to set all the configuration information for the node. This includes the same information available via calls to SetClusterConfig and SetNetworkConfig in one API method. Note: This method is available only through the per-node API endpoint 5.0 or later. Caution: Changing the "bond-mode" on a node can cause a temporary loss of network connectivity. Exercise caution when using this method.

Parameters **config** (ConfigParams) – [required] Objects that you want changed for the cluster interface settings.

config = <class 'solidfire.models.ConfigParams'>

class solidfire.models.**SetConfigResult**(*config*)
Bases: *solidfire.common.model.DataObject*

Parameters **config** (Config) – [required] The new and current configuration for the node.

config = <class 'solidfire.models.Config'>

class solidfire.models.**SetDefaultQoSRequest**(*min_iops=None*, *max_iops=None*, *burst_iops=None*)
Bases: *solidfire.common.model.DataObject*

SetDefaultQoS enables you to configure the default Quality of Service (QoS) values (measured in inputs and outputs per second, or IOPS) for a volume. For more information about QoS in a SolidFire cluster, see the User Guide.

Parameters

- **min_iops** (*int*) – The minimum number of sustained IOPS provided by the cluster to a volume.

- **max_iops** (*int*) – The maximum number of sustained IOPS provided by the cluster to a volume.
- **burst_iops** (*int*) – The maximum number of IOPS allowed in a short burst scenario.

burst_iops = <type 'int'>

max_iops = <type 'int'>

min_iops = <type 'int'>

class solidfire.models.**SetDefaultQoSResult**(*min_iops*, *max_iops*, *burst_iops*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **min_iops** (*int*) – [required] The minimum number of sustained IOPS that are provided by the cluster to a volume.
- **max_iops** (*int*) – [required] The maximum number of sustained IOPS that are provided by the cluster to a volume.
- **burst_iops** (*int*) – [required] The maximum number of IOPS allowed in a short burst scenario.

burst_iops = <type 'int'>

max_iops = <type 'int'>

min_iops = <type 'int'>

class solidfire.models.**SetLoginBannerRequest**(*banner=None*, *enabled=None*)
Bases: *solidfire.common.model.DataObject*

You can use the SetLoginBanner method to set the active Terms of Use banner users see when they log on to the web interface.

Parameters

- **banner** (*str*) – The desired text of the Terms of Use banner.
- **enabled** (*bool*) – The status of the Terms of Use banner. Possible values: true: The Terms of Use banner is displayed upon web interface login. false: The Terms of Use banner is not displayed upon web interface login.

banner = <type 'str'>

enabled = <type 'bool'>

class solidfire.models.**SetLoginBannerResult**(*login_banner*)
Bases: *solidfire.common.model.DataObject*

Parameters **login_banner** (LoginBanner) – [required]

login_banner = <class 'solidfire.models.LoginBanner'>

class solidfire.models.**SetLoginSessionInfoRequest**(*timeout*)
Bases: *solidfire.common.model.DataObject*

You can use SetLoginSessionInfo to set the period of time that a session's login authentication is valid. After the log in period elapses without activity on the system, the authentication expires. New login credentials are required for continued access to the cluster after the timeout period has elapsed.

Parameters **timeout** (*str*) – [required] Cluster authentication expiration period. Formatted in HH:mm:ss. For example, 01:30:00, 00:90:00, and 00:00:5400 can be used to equal a 90 minute timeout period. The default value is 30 minutes. The minimum value is 1 minute.

timeout = <type 'str'>

class solidfire.models.**SetLoginSessionInfoResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetNetworkConfigRequest**(*network*)
Bases: *solidfire.common.model.DataObject*

The SetNetworkConfig API method enables you to set the network configuration for a node. To display the current network settings for a node, run the GetNetworkConfig API method. Note: This method is available only through the per-node API endpoint 5.0 or later. Changing the "bond-mode" on a node can cause a temporary loss of network connectivity. Exercise caution when using this method.

Parameters **network** (NetworkParams) – [required] An object containing node network settings to modify.

network = <class 'solidfire.models.NetworkParams'>

class solidfire.models.**SetNetworkConfigResult**(*network*)
Bases: *solidfire.common.model.DataObject*

Parameters **network** (Network) – [required]

network = <class 'solidfire.models.Network'>

class solidfire.models.**SetNodeSSLCertificateRequest**(*certificate*, *private_key*)
Bases: *solidfire.common.model.DataObject*

You can use the SetNodeSSLCertificate method to set a user SSL certificate and private key for the management node.

Parameters

- **certificate** (*str*) – [required] The PEM-encoded text version of the certificate.
- **private_key** (*str*) – [required] The PEM-encoded text version of the private key.

certificate = <type 'str'>

private_key = <type 'str'>

class solidfire.models.**SetNodeSSLCertificateResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetNtpInfoRequest**(*servers*, *broadcastclient=None*)
Bases: *solidfire.common.model.DataObject*

SetNtpInfo enables you to configure NTP on cluster nodes. The values you set with this interface apply to all nodes in the cluster. If an NTP broadcast server periodically broadcasts time information on your network, you can optionally configure nodes as broadcast clients. Note: NetApp recommends using NTP servers that are internal to your network, rather than the installation defaults.

Parameters

- **servers** (*str*) – [required] List of NTP servers to add to each nodes NTP configuration.
- **broadcastclient** (*bool*) – Enables every node in the cluster as a broadcast client.

broadcastclient = <type 'bool'>

servers = <type 'str[]'>

class solidfire.models.**SetNtpInfoResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetRemoteLoggingHostsRequest**(*remote_hosts*)
Bases: *solidfire.common.model.DataObject*

SetRemoteLoggingHosts enables you to configure remote logging from the nodes in the storage cluster to a centralized log server or servers. Remote logging is performed over TCP using the default port 514. This API does not add to the existing logging hosts. Rather, it replaces what currently exists with new values specified by this API method. You can use GetRemoteLoggingHosts to determine what the current logging hosts are, and then use SetRemoteLoggingHosts to set the desired list of current and new logging hosts.

Parameters **remote_hosts** (LoggingServer) – [required] A list of hosts to send log messages to.

remote_hosts = <class 'solidfire.models.LoggingServer[]'>

class solidfire.models.**SetRemoteLoggingHostsResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetSSLCertificateRequest**(*certificate*, *private_key*)
Bases: *solidfire.common.model.DataObject*

You can use the SetSSLCertificate method to set a user SSL certificate and a private key for the cluster.

Parameters

- **certificate** (*str*) – [required] The PEM-encoded text version of the certificate.
- **private_key** (*str*) – [required] The PEM-encoded text version of the private key.

certificate = <type 'str'>

private_key = <type 'str'>

class solidfire.models.**SetSSLCertificateResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetSnmpACLRequest**(*networks*, *usm_users*)
Bases: *solidfire.common.model.DataObject*

SetSnmpACL enables you to configure SNMP access permissions on the cluster nodes. The values you set with this interface apply to all nodes in the cluster, and the values that are passed replace, in whole, all values set in any previous call to SetSnmpACL. Also note that the values set with this interface replace all network or usmUsers values set with the older SetSnmpInfo.

Parameters

- **networks** (SnmpNetwork) – [required] List of networks and what type of access they have to the SNMP servers running on the cluster nodes. See SNMP Network Object for possible "networks" values. This parameter is required if SNMP v3 is disabled.
- **usm_users** (SnmpV3UsmUser) – [required] List of users and the type of access they have to the SNMP servers running on the cluster nodes.

networks = <class 'solidfire.models.SnmpNetwork[]'>

usm_users = <class 'solidfire.models.SnmpV3UsmUser[]'>

class solidfire.models.**SetSnmpACLResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetSnmpInfoRequest**(*networks=None*, *enabled=None*, *snmp_v3_enabled=None*, *usm_users=None*)
Bases: *solidfire.common.model.DataObject*

SetSnmpInfo enables you to configure SNMP version 2 and version 3 on cluster nodes. The values you set with this interface apply to all nodes in the cluster, and the values that are passed replace, in whole, all values set in

any previous call to SetSnmpInfo. Note: SetSnmpInfo is deprecated. Use the EnableSnmp and SetSnmpACL methods instead.

Parameters

- **networks** (SnmpNetwork) – List of networks and what type of access they have to the SNMP servers running on the cluster nodes. See the SNMP Network Object for possible "networks" values. This parameter is required only for SNMP v2.
- **enabled** (*bool*) – If set to true, SNMP is enabled on each node in the cluster.
- **snmp_v3_enabled** (*bool*) – If set to true, SNMP v3 is enabled on each node in the cluster.
- **usm_users** (SnmpV3UsmUser) – If SNMP v3 is enabled, this value must be passed in place of the networks parameter. This parameter is required only for SNMP v3.

enabled = <type 'bool'>

networks = <class 'solidfire.models.SnmpNetwork[]'>

snmp_v3_enabled = <type 'bool'>

usm_users = <class 'solidfire.models.SnmpV3UsmUser[]'>

class solidfire.models.**SetSnmpInfoResult**

Bases: *solidfire.common.model.DataObject*

class solidfire.models.**SetSnmpTrapInfoRequest**(*trap_recipients*, *cluster_fault_traps_enabled*, *cluster_fault_resolved_traps_enabled*, *cluster_event_traps_enabled*)

Bases: *solidfire.common.model.DataObject*

You can use SetSnmpTrapInfo to enable and disable the generation of cluster SNMP notifications (traps) and to specify the set of network host computers that receive the notifications. The values you pass with each SetSnmpTrapInfo method call replace all values set in any previous call to SetSnmpTrapInfo.

Parameters

- **trap_recipients** (SnmpTrapRecipient) – [required] List of hosts that are to receive the traps generated by the Cluster Master. At least one object is required if any one of the trap types is enabled.
- **cluster_fault_traps_enabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterFaultNotification is sent to the configured list of trap recipients when a cluster fault is logged. The default value is false.
- **cluster_fault_resolved_traps_enabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterFaultResolvedNotification is sent to the configured list of trap recipients when a cluster fault is resolved. The default value is false.
- **cluster_event_traps_enabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterEventNotification is sent to the configured list of trap recipients when a cluster event is logged. The default value is false.

cluster_event_traps_enabled = <type 'bool'>

cluster_fault_resolved_traps_enabled = <type 'bool'>

cluster_fault_traps_enabled = <type 'bool'>

trap_recipients = <class 'solidfire.models.SnmpTrapRecipient[]'>

class solidfire.models.**SetSnmpTrapInfoResult**
Bases: *solidfire.common.model.DataObject*

class solidfire.models.**ShutdownRequest**(*nodes*, *option=None*)
Bases: *solidfire.common.model.DataObject*

The Shutdown API method enables you to restart or shutdown a node that has not yet been added to a cluster. To use this method, log in to the MIP for the pending node, and enter the "shutdown" method with either the "restart" or "halt" options.

Parameters

- **nodes** (*int*) – [required] List of NodeIDs for the nodes to be shutdown.
- **option** (*str*) – Specifies the action to take for the node shutdown. Possible values are: restart: Restarts the node. halt: Shuts down the node.

nodes = <type 'int[]'>

option = <type 'str'>

class solidfire.models.**ShutdownResult**(*failed*, *successful*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **failed** (*int*) – [required]
- **successful** (*int*) – [required]

failed = <type 'int[]'>

successful = <type 'int[]'>

class solidfire.models.**Signature**(*data*, *pubkey*, *version*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **data** (*str*) – [required]
- **pubkey** (*str*) – [required]
- **version** (*int*) – [required]

data = <type 'str'>

pubkey = <type 'str'>

version = <type 'int'>

class solidfire.models.**Snapshot**(*snapshot_id*, *volume_id*, *name*, *checksum*, *enable_remote_replication*, *expiration_reason*, *status*, *snapshot_uuid*, *total_size*, *group_snapshot_uuid*, *create_time*, *instance_create_time*, *volume_name*, *instance_snapshot_uuid*, *attributes*, *expiration_time=None*, *remote_statuses=None*, *group_id=None*, *virtual_volume_id=None*, *snap_mirror_label=None*)
Bases: *solidfire.common.model.DataObject*

Snapshots is an object containing information about each snapshot made for a volume. The return object includes information for the active snapshot as well as each snapshot created for the volume.

Parameters

- **snapshot_id** (*int*) – [required] Unique ID for this snapshot.

- **volume_id** (*int*) – [required] The volume this snapshot was taken of.
- **name** (*str*) – [required] A name for this snapshot. It is not necessarily unique.
- **checksum** (*str*) – [required] A string that represents the correct digits in the stored snapshot. This checksum can be used later to compare other snapshots to detect errors in the data.
- **enable_remote_replication** (*bool*) – [required] Identifies if snapshot is enabled for remote replication.
- **expiration_reason** (*str*) – [required] Indicates how the snapshot expiration was set. Possible values: api: expiration time was set by using the API. none: there is no expiration time set. test: expiration time was set for testing.
- **expiration_time** (*str*) – The time at which this snapshot will expire and be purged from the cluster.
- **remote_statuses** (SnapshotRemoteStatus) – Current remote status of the snapshot remoteStatus: Possible values: Present: Snapshot exists on a remote cluster Not Present: Snapshot does not exist on remote cluster Syncing: This is a target cluster and it is currently replicating the snapshot Deleted: This is a target cluster, the snapshot has been deleted, and it still exists on the source. volumePairUUID: universal identifier of the volume pair
- **status** (*str*) – [required] Current status of the snapshot Preparing: A snapshot that is being prepared for use and is not yet writable. Done: A snapshot that has finished being prepared and is now usable. Active: This snapshot is the active branch.
- **snapshot_uuid** (*UUID*) – [required] Universal Unique ID of an existing snapshot.
- **total_size** (*int*) – [required] Total size of this snapshot in bytes. It is equivalent to totalSize of the volume when this snapshot was taken.
- **group_id** (*int*) – If present, the ID of the group this snapshot is a part of. If not present, this snapshot is not part of a group.
- **group_snapshot_uuid** (*UUID*) – [required] The current "members" results contains information about each snapshot in the group. Each of these members will have a "uuid" parameter for the snapshot's UUID.
- **create_time** (*str*) – [required] The time this snapshot was taken.
- **instance_create_time** (*str*) – [required]
- **volume_name** (*str*) – [required]
- **instance_snapshot_uuid** (*UUID*) – [required]
- **virtual_volume_id** (*UUID*) – The ID of the virtual volume with which the snapshot is associated.
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.
- **snap_mirror_label** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

```
attributes = <type 'dict'>

checksum = <type 'str'>

create_time = <type 'str'>

enable_remote_replication = <type 'bool'>
```

expiration_reason = <type 'str'>

expiration_time = <type 'str'>

group_id = <type 'int'>

group_snapshot_uuid = <class 'uuid.UUID'>

instance_create_time = <type 'str'>

instance_snapshot_uuid = <class 'uuid.UUID'>

name = <type 'str'>

remote_statuses = <class 'solidfire.models.SnapshotRemoteStatus[]'>

snap_mirror_label = <type 'str'>

snapshot_id = <type 'int'>

snapshot_uuid = <class 'uuid.UUID'>

status = <type 'str'>

total_size = <type 'int'>

virtual_volume_id = <class 'uuid.UUID'>

volume_id = <type 'int'>

volume_name = <type 'str'>

class solidfire.models.**SnapshotRemoteStatus**(*remote_status*, *volume_pair_uuid*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **remote_status** (*str*) – [required]
- **volume_pair_uuid** (*UUID*) – [required] The snapshot is done and is writable (the active branch of the slice).

remote_status = <type 'str'>

volume_pair_uuid = <class 'uuid.UUID'>

class solidfire.models.**SnapshotReplication**(*state*, *state_details*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **state** (*str*) – [required] The state of the snapshot replication.
- **state_details** (*str*) – [required] Reserved for future use.

state = <type 'str'>

state_details = <type 'str'>

class solidfire.models.**SnmpNetwork**(*access*, *cidr*, *community*, *network*)

Bases: *solidfire.common.model.DataObject*

The SNMP network object contains information about SNMP configuration for the cluster nodes. SNMP v3 is supported on SolidFire clusters.

Parameters

- **access** (*str*) – [required] ro: read-only access.* rw: for read-write access. rosys: for read-only access to a restricted set of system information *SolidFire recommends that all networks other than the default "localhost" be set to "ro" access, because all SolidFire MIB objects are read-only.
- **cidr** (*int*) – [required] A CIDR network mask. This network mask must be an integer greater than or equal to 0, and less than or equal to 32. It must also not be equal to 31.
- **community** (*str*) – [required] SNMP community string.
- **network** (*str*) – [required] This parameter ainteger with the cidr variable is used to control which network the access and community string apply to. The special value of "default" is used to specify an entry that applies to all networks. The cidr mask is ignored when network value is either a host name or default.

access = <type 'str'>

cidr = <type 'int'>

community = <type 'str'>

network = <type 'str'>

class solidfire.models.**SnmpSendTestTrapsResult**(*status*)

Bases: *solidfire.common.model.DataObject*

Parameters **status** (*str*) – [required]

status = <type 'str'>

class solidfire.models.**SnmpTrapRecipient**(*host*, *community*, *port*)

Bases: *solidfire.common.model.DataObject*

Host that is to receive the traps generated by the cluster.

Parameters

- **host** (*str*) – [required] The IP address or host name of the target network management station.
- **community** (*str*) – [required] SNMP community string.
- **port** (*int*) – [required] The UDP port number on the host where the trap is to be sent. Valid range is 1 - 65535. 0 (zero) is not a valid port number. Default is 162.

community = <type 'str'>

host = <type 'str'>

port = <type 'int'>

class solidfire.models.**SnmpV3UsmUser**(*access*, *name*, *password*, *passphrase*, *sec_level*)

Bases: *solidfire.common.model.DataObject*

The SNMP v3 usmUser object is used with the API method SetSnmpInfo to configure SNMP on the cluster.

Parameters

- **access** (*str*) – [required] rouser: read-only access.* rwuser: for read-write access. rosys: for read-only access to a restricted set of system information *SolidFire recommends that all USM users be set to "rouser" access, because all SolidFire MIB objects are read-only.
- **name** (*str*) – [required] The name of the user. Must contain at least one character, but no more than 32 characters. Blank spaces are not allowed.

- **password** (*str*) – [required] The password of the user. Must be between 8 and 255 characters integer (inclusive). Blank spaces are not allowed. Required if "secLevel" is "auth" or "priv."
- **passphrase** (*str*) – [required] The passphrase of the user. Must be between 8 and 255 characters integer (inclusive). Blank spaces are not allowed. Required if "secLevel" is "priv."
- **sec_level** (*str*) – [required] noauth: No password or passphrase is required. auth: A password is required for user access. priv: A password and passphrase is required for user access.

access = <type 'str'>

name = <type 'str'>

passphrase = <type 'str'>

password = <type 'str'>

sec_level = <type 'str'>

class solidfire.models.**SoftwareVersionInfo**(*current_version*, *node_id*, *package_name*, *pending_version*, *start_time*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **current_version** (*str*) – [required]
- **node_id** (*int*) – [required]
- **package_name** (*str*) – [required]
- **pending_version** (*str*) – [required]
- **start_time** (*str*) – [required]

current_version = <type 'str'>

node_id = <type 'int'>

package_name = <type 'str'>

pending_version = <type 'str'>

start_time = <type 'str'>

class solidfire.models.**StartBulkVolumeReadRequest**(*volume_id*, *format*, *snapshot_id=None*, *script=None*, *script_parameters=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

StartBulkVolumeRead enables you to initialize a bulk volume read session on a specified volume. Only two bulk volume processes can run simultaneously on a volume. When you initialize the session, data is read from a SolidFire storage volume for the purposes of storing the data on an external backup source. The external data is accessed by a web server running on an SF-series node. Communications and server interaction information for external data access is passed by a script running on the storage system. At the start of a bulk volume read operation, a snapshot of the volume is made and the snapshot is deleted when the read is complete. You can also read a snapshot of the volume by entering the ID of the snapshot as a parameter. When you read a previous snapshot, the system does not create a new snapshot of the volume or delete the previous snapshot when the read completes. Note: This process creates a new snapshot if the ID of an existing snapshot is not provided. Snapshots can be created if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5.

Parameters

- **volume_id** (*int*) – [required] The ID of the volume to be read.
- **format** (*str*) – [required] The format of the volume data. It can be either of the following formats: uncompressed: Every byte of the volume is returned without any compression. native: Opaque data is returned that is smaller and more efficiently stored and written on a subsequent bulk volume write.
- **snapshot_id** (*int*) – The ID of a previously created snapshot used for bulk volume reads. If no ID is entered, a snapshot of the current active volume image is made.
- **script** (*str*) – The executable name of a script. If unspecified, the key and URL is necessary to access SF-series nodes. The script is run on the primary node and the key and URL is returned to the script so the local web server can be contacted.
- **script_parameters** (*dict*) – JSON parameters to pass to the script.
- **attributes** (*dict*) – JSON attributes for the bulk volume job.

attributes = <type 'dict'>

format = <type 'str'>

script = <type 'str'>

script_parameters = <type 'dict'>

snapshot_id = <type 'int'>

volume_id = <type 'int'>

class solidfire.models.**StartBulkVolumeReadResult**(*async_handle*, *key*, *url*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **async_handle** (*int*) – [required] ID of the async process to be checked for completion.
- **key** (*str*) – [required] Opaque key uniquely identifying the session.
- **url** (*str*) – [required] URL to access the node's web server

async_handle = <type 'int'>

key = <type 'str'>

url = <type 'str'>

class solidfire.models.**StartBulkVolumeWriteRequest**(*volume_id*, *format*, *script=None*, *script_parameters=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

StartBulkVolumeWrite enables you to initialize a bulk volume write session on a specified volume. Only two bulk volume processes can run simultaneously on a volume. When you initialize the write session, data is written to a SolidFire storage volume from an external backup source. The external data is accessed by a web server running on an SF-series node. Communications and server interaction information for external data access is passed by a script running on the storage system.

Parameters

- **volume_id** (*int*) – [required] The ID of the volume to be written to.
- **format** (*str*) – [required] The format of the volume data. It can be either of the following formats: uncompressed: Every byte of the volume is returned without any compression.

native: Opaque data is returned that is smaller and more efficiently stored and written on a subsequent bulk volume write.

- **script** (*str*) – The executable name of a script. If unspecified, the key and URL are necessary to access SF-series nodes. The script runs on the primary node and the key and URL is returned to the script, so the local web server can be contacted.
- **script_parameters** (*dict*) – JSON parameters to pass to the script.
- **attributes** (*dict*) – JSON attributes for the bulk volume job.

attributes = <type 'dict'>

format = <type 'str'>

script = <type 'str'>

script_parameters = <type 'dict'>

volume_id = <type 'int'>

class solidfire.models.**StartBulkVolumeWriteResult**(*async_handle*, *key*, *url*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **async_handle** (*int*) – [required] ID of the async process to be checked for completion.
- **key** (*str*) – [required] Opaque key uniquely identifying the session.
- **url** (*str*) – [required] URL to access the node's web server

async_handle = <type 'int'>

key = <type 'str'>

url = <type 'str'>

class solidfire.models.**StartClusterPairingResult**(*cluster_pairing_key*, *cluster_pair_id*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **cluster_pairing_key** (*str*) – [required] A string of characters that is used by the "CompleteClusterPairing" API method.
- **cluster_pair_id** (*int*) – [required] Unique identifier for the cluster pair.

cluster_pair_id = <type 'int'>

cluster_pairing_key = <type 'str'>

class solidfire.models.**StartVolumePairingRequest**(*volume_id*, *mode=None*)

Bases: *solidfire.common.model.DataObject*

StartVolumePairing enables you to create an encoded key from a volume that is used to pair with another volume. The key that this method creates is used in the CompleteVolumePairing API method to establish a volume pairing.

Parameters

- **volume_id** (*int*) – [required] The ID of the volume on which to start the pairing process.
- **mode** (*str*) – The mode of the volume on which to start the pairing process. The mode can only be set if the volume is the source volume. Possible values are: Async: (default if no mode parameter specified) Writes are acknowledged when they complete locally. The

cluster does not wait for writes to be replicated to the target cluster. Sync: Source acknowledges write when the data is stored locally and on the remote cluster. SnapshotsOnly: Only snapshots created on the source cluster will be replicated. Active writes from the source volume are not replicated.

mode = <type 'str'>

volume_id = <type 'int'>

class solidfire.models.**StartVolumePairingResult**(*volume_pairing_key*)

Bases: *solidfire.common.model.DataObject*

Parameters **volume_pairing_key** (*str*) – [required] A string of characters that is used by the "CompleteVolumePairing" API method.

volume_pairing_key = <type 'str'>

class solidfire.models.**StorageContainer**(*name*, *storage_container_id*, *account_id*, *protocol_endpoint_type*, *initiator_secret*, *target_secret*, *status*, *virtual_volumes=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **name** (*str*) – [required]
- **storage_container_id** (*UUID*) – [required]
- **account_id** (*int*) – [required]
- **protocol_endpoint_type** (*str*) – [required]
- **initiator_secret** (*str*) – [required]
- **target_secret** (*str*) – [required]
- **status** (*str*) – [required]
- **virtual_volumes** (*UUID*) –

account_id = <type 'int'>

initiator_secret = <type 'str'>

name = <type 'str'>

protocol_endpoint_type = <type 'str'>

status = <type 'str'>

storage_container_id = <class 'uuid.UUID'>

target_secret = <type 'str'>

virtual_volumes = <class 'uuid.UUID[]'>

class solidfire.models.**SupportBundleDetails**(*bundle_name*, *extra_args*, *files*, *url*, *output*, *timeout_sec*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **bundle_name** (*str*) – [required] The name specified in the 'CreateSupportBundle API method. If no name was specified, 'supportbundle' will be used.
- **extra_args** (*str*) – [required] The arguments passed with this method.
- **files** (*str*) – [required] A list of the support bundle files that were created.

- **url** (*str*) – [required] The URL that you can use to download the bundle file(s). Should correspond 1:1 with files list.
- **output** (*str*) – [required] The commands that were run and their output, with newlines replaced by HTML
 elements.
- **timeout_sec** (*int*) – [required] The timeout specified for the support bundle creation process.

bundle_name = <type 'str'>

extra_args = <type 'str'>

files = <type 'str[]'>

output = <type 'str'>

timeout_sec = <type 'int'>

url = <type 'str[]'>

class solidfire.models.**SyncJob**(*bytes_per_second*, *current_bytes*, *dst_service_id*, *elapsed_time*, *percent_complete*, *slice_id*, *src_service_id*, *total_bytes*, *type*, *clone_id*, *dst_volume_id*, *node_id*, *snapshot_id*, *src_volume_id*, *blocks_per_second*, *stage*, *remaining_time=None*, *group_clone_id=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **bytes_per_second** (*float*) – [required]
- **current_bytes** (*int*) – [required]
- **dst_service_id** (*int*) – [required]
- **elapsed_time** (*float*) – [required]
- **percent_complete** (*float*) – [required]
- **remaining_time** (*int*) –
- **slice_id** (*int*) – [required]
- **src_service_id** (*int*) – [required]
- **total_bytes** (*int*) – [required]
- **type** (*str*) – [required]
- **clone_id** (*int*) – [required]
- **dst_volume_id** (*int*) – [required]
- **node_id** (*int*) – [required]
- **snapshot_id** (*int*) – [required]
- **src_volume_id** (*int*) – [required]
- **blocks_per_second** (*float*) – [required]
- **stage** (*str*) – [required]
- **group_clone_id** (*int*) –

blocks_per_second = <type 'float'>

bytes_per_second = <type 'float'>

clone_id = <type 'int'>

current_bytes = <type 'int'>

dst_service_id = <type 'int'>

dst_volume_id = <type 'int'>

elapsed_time = <type 'float'>

group_clone_id = <type 'int'>

node_id = <type 'int'>

percent_complete = <type 'float'>

remaining_time = <type 'int'>

slice_id = <type 'int'>

snapshot_id = <type 'int'>

src_service_id = <type 'int'>

src_volume_id = <type 'int'>

stage = <type 'str'>

total_bytes = <type 'int'>

type = <type 'str'>

class solidfire.models.**TestConnectEnsembleDetails**(*nodes*)
Bases: *solidfire.common.model.DataObject*

Parameters **nodes** (*dict*) – [required] A list of each ensemble node in the test and the results of the tests.

nodes = <type 'dict'>

class solidfire.models.**TestConnectEnsembleRequest**(*ensemble=None*)
Bases: *solidfire.common.model.DataObject*

The TestConnectEnsemble API method enables you to verify connectivity with a specified database ensemble. By default, it uses the ensemble for the cluster that the node is associated with. Alternatively, you can provide a different ensemble to test connectivity with. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **ensemble** (*str*) – Uses a comma-separated list of ensemble node cluster IP addresses to test connectivity. This parameter is optional.

ensemble = <type 'str'>

class solidfire.models.**TestConnectEnsembleResult**(*details*, *duration*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (TestConnectEnsembleDetails) – [required]
- **duration** (*str*) – [required] The length of time required to run the test.
- **result** (*str*) – [required] The results of the entire test

details = <class 'solidfire.models.TestConnectEnsembleDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**TestConnectMvipDetails**(*ping_bytes*, *mvip*, *connected*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **ping_bytes** (*dict*) – [required] Details of the ping tests with 56 Bytes and 1500 Bytes.
- **mvip** (*str*) – [required] The MVIP tested against.
- **connected** (*bool*) – [required] Whether the test could connect to the MVIP.

connected = <type 'bool'>

mvip = <type 'str'>

ping_bytes = <type 'dict'>

class solidfire.models.**TestConnectMvipRequest**(*mvip=None*)
Bases: *solidfire.common.model.DataObject*

The TestConnectMvip API method enables you to test the management connection to the cluster. The test pings the MVIP and executes a simple API method to verify connectivity. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **mvip** (*str*) – If specified, tests the management connection of a different MVIP. You do not need to use this value when testing the connection to the target cluster. This parameter is optional.

mvip = <type 'str'>

class solidfire.models.**TestConnectMvipResult**(*details*, *duration*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (TestConnectMvipDetails) – [required] Information about the test operation
- **duration** (*str*) – [required] The length of time required to run the test.
- **result** (*str*) – [required] The results of the entire test

details = <class 'solidfire.models.TestConnectMvipDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**TestConnectSvipDetails**(*ping_bytes*, *svip*, *connected*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **ping_bytes** (*dict*) – [required] Details of the ping tests with 56 Bytes and 1500 Bytes.
- **svip** (*str*) – [required] The SVIP tested against.
- **connected** (*bool*) – [required] Whether the test could connect to the MVIP.

connected = <type 'bool'>

ping_bytes = <type 'dict'>

svip = <type 'str'>

class solidfire.models.**TestConnectSvipRequest**(*svip=None*)
Bases: *solidfire.common.model.DataObject*

The TestConnectSvip API method enables you to test the storage connection to the cluster. The test pings the SVIP using ICMP packets, and when successful, connects as an iSCSI initiator. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters **svip** (*str*) – If specified, tests the storage connection of a different SVIP. You do not need to use this value when testing the connection to the target cluster. This parameter is optional.

svip = <type 'str'>

class solidfire.models.**TestConnectSvipResult**(*details*, *duration*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (TestConnectSvipDetails) – [required] Information about the test operation
- **duration** (*str*) – [required] The length of time required to run the test.
- **result** (*str*) – [required] The results of the entire test

details = <class 'solidfire.models.TestConnectSvipDetails'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**TestDrivesRequest**(*minutes=None*, *force=None*)
Bases: *solidfire.common.model.DataObject*

You can use the TestDrives API method to run a hardware validation on all drives on the node. This method detects hardware failures on the drives (if present) and reports them in the results of the validation tests. You can only use the TestDrives method on nodes that are not "active" in a cluster. Note: This test takes approximately 10 minutes. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters

- **minutes** (*int*) – Specifies the number of minutes to run the test.
- **force** (*bool*) – Required parameter to successfully test the drives on the node.

force = <type 'bool'>

minutes = <type 'int'>

class solidfire.models.**TestDrivesResult**(*details*, *duration*, *result*)
Bases: *solidfire.common.model.DataObject*

Parameters

- **details** (*str*) – [required]
- **duration** (*str*) – [required]
- **result** (*str*) – [required]

details = <type 'str'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**TestLdapAuthenticationRequest**(*username*, *password*, *ldap_configuration=None*)
Bases: *solidfire.common.model.DataObject*

The TestLdapAuthentication method enables you to validate the currently enabled LDAP authentication settings. If the configuration is correct, the API call returns the group membership of the tested user.

Parameters

- **username** (*str*) – [required] The username to be tested.
- **password** (*str*) – [required] The password for the username to be tested.
- **ldap_configuration** (LdapConfiguration) – An ldapConfiguration object to be tested. If specified, the API call tests the provided configuration even if LDAP authentication is disabled.

ldap_configuration = <class 'solidfire.models.LdapConfiguration'>

password = <type 'str'>

username = <type 'str'>

class solidfire.models.**TestLdapAuthenticationResult**(*groups*, *user_dn*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **groups** (*str*) – [required] List of LDAP groups that the tested user is a member of.
- **user_dn** (*str*) – [required] The tested user's full LDAP distinguished name.

groups = <type 'str[]'>

user_dn = <type 'str'>

class solidfire.models.**TestPingRequest**(*attempts=None*, *hosts=None*, *total_timeout_sec=None*, *packet_size=None*, *ping_timeout_msec=None*, *prohibit_fragmentation=None*)

Bases: *solidfire.common.model.DataObject*

You can use the TestPing API method to validate the connection to all the nodes in a cluster on both 1G and 10G interfaces by using ICMP packets. The test uses the appropriate MTU sizes for each packet based on the MTU settings in the network configuration. Note: This method is available only through the per-node API endpoint 5.0 or later.

Parameters

- **attempts** (*int*) – Specifies the number of times the system should repeat the test ping. The default value is 5.
- **hosts** (*str*) – Specifies a comma-separated list of addresses or hostnames of devices to ping.
- **total_timeout_sec** (*int*) – Specifies the length of time the ping should wait for a system response before issuing the next ping attempt or ending the process.
- **packet_size** (*int*) – Specifies the number of bytes to send in the ICMP packet that is sent to each IP. The number must be less than the maximum MTU specified in the network configuration.
- **ping_timeout_msec** (*int*) – Specifies the number of milliseconds to wait for each individual ping response. The default value is 500 ms.
- **prohibit_fragmentation** (*bool*) – Specifies that the Do not Fragment (DF) flag is enabled for the ICMP packets.

attempts = <type 'int'>

hosts = <type 'str'>

packet_size = <type 'int'>

ping_timeout_msec = <type 'int'>

prohibit_fragmentation = <type 'bool'>

total_timeout_sec = <type 'int'>

class solidfire.models.**TestPingResult**(*result*, *duration*, *details*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **result** (*str*) – [required] Result of the ping test.
- **duration** (*str*) – [required] The total duration of the ping test.
- **details** (*dict*) – [required] List of each IP the node was able to communicate with.

details = <type 'dict'>

duration = <type 'str'>

result = <type 'str'>

class solidfire.models.**UpdateBulkVolumeStatusRequest**(*key*, *status*, *percent_complete=None*, *message=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

You can use UpdateBulkVolumeStatus in a script to update the status of a bulk volume job that you started with the StartBulkVolumeRead or StartBulkVolumeWrite methods.

Parameters

- **key** (*str*) – [required] The key assigned during initialization of a StartBulkVolumeRead or StartBulkVolumeWrite session.
- **status** (*str*) – [required] The status of the given bulk volume job. The system sets the status. Possible values are: running: Jobs that are still active. complete: Jobs that are done. failed: Jobs that failed.
- **percent_complete** (*str*) – The completed progress of the bulk volume job as a percentage value.
- **message** (*str*) – The message returned indicating the status of the bulk volume job after the job is complete.
- **attributes** (*dict*) – JSON attributes; updates what is on the bulk volume job.

attributes = <type 'dict'>

key = <type 'str'>

message = <type 'str'>

percent_complete = <type 'str'>

status = <type 'str'>

class solidfire.models.**UpdateBulkVolumeStatusResult**(*status*, *url*, *attributes*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **status** (*str*) – [required] Status of the session requested. Returned status: preparing active done failed
- **url** (*str*) – [required] The URL to access the node's web server provided only if the session is still active.
- **attributes** (*dict*) – [required] Returns attributes that were specified in the BulkVolumeStatusUpdate method. Values are returned if they have changed or not.

attributes = <type 'dict'>

status = <type 'str'>

url = <type 'str'>

class solidfire.models.**VirtualNetwork**(*virtual_network_id*, *virtual_network_tag*, *address_blocks*, *name*, *netmask*, *svip*, *gateway=None*, *namespace=None*, *attributes=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_network_id** (*int*) – [required] SolidFire unique identifier for a virtual network.
- **virtual_network_tag** (*int*) – [required] VLAN Tag identifier.
- **address_blocks** (AddressBlock) – [required] Range of address blocks currently assigned to the virtual network. available: Binary string in "1"s and "0"s. 1 equals the IP is available and 0 equals the IP is not available. The string is read from right to left with the digit to the far right being the first IP address in the list of addressBlocks. size: the size of this block of addresses. start: first IP address in the block.
- **name** (*str*) – [required] The name assigned to the virtual network.
- **netmask** (*str*) – [required] IP address of the netmask for the virtual network.
- **svip** (*str*) – [required] Storage IP address for the virtual network.
- **gateway** (*str*) –
- **namespace** (*bool*) –
- **attributes** (*dict*) – List of Name/Value pairs in JSON object format.

address_blocks = <class 'solidfire.models.AddressBlock[]'>

attributes = <type 'dict'>

gateway = <type 'str'>

name = <type 'str'>

namespace = <type 'bool'>

netmask = <type 'str'>

svip = <type 'str'>

virtual_network_id = <type 'int'>

virtual_network_tag = <type 'int'>

class solidfire.models.**VirtualNetworkAddress**(*virtual_network_id*, *address*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_network_id** (*int*) – [required] SolidFire unique identifier for a virtual network.
- **address** (*str*) – [required] Virtual Network Address.

address = <type 'str'>

virtual_network_id = <type 'int'>

class solidfire.models.**VirtualVolumeBinding**(*protocol_endpoint_id*, *protocol_endpoint_in_band_id*, *protocol_endpoint_type*, *virtual_volume_binding_id*, *virtual_volume_host_id*, *virtual_volume_id*, *virtual_volume_secondary_id*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **protocol_endpoint_id** (*UUID*) – [required] The unique ID of the protocol endpoint.
- **protocol_endpoint_in_band_id** (*str*) – [required] The scsiNAADeviceID of the protocol endpoint. For more information, see protocolEndpoint.
- **protocol_endpoint_type** (*str*) – [required] The type of protocol endpoint. SCSI is the only value returned for the protocol endpoint type.
- **virtual_volume_binding_id** (*int*) – [required] The unique ID of the virtual volume binding object.
- **virtual_volume_host_id** (*UUID*) – [required] The unique ID of the virtual volume host.
- **virtual_volume_id** (*UUID*) – [required] The unique ID of the virtual volume.
- **virtual_volume_secondary_id** (*str*) – [required] The secondary ID of the virtual volume.

protocol_endpoint_id = <class 'uuid.UUID'>

protocol_endpoint_in_band_id = <type 'str'>

protocol_endpoint_type = <type 'str'>

virtual_volume_binding_id = <type 'int'>

virtual_volume_host_id = <class 'uuid.UUID'>

virtual_volume_id = <class 'uuid.UUID'>

virtual_volume_secondary_id = <type 'str'>

class solidfire.models.**VirtualVolumeHost**(*virtual_volume_host_id*, *cluster_id*, *visible_protocol_endpoint_ids*, *bindings*, *initiator_names*, *host_address*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_volume_host_id** (*UUID*) – [required]
- **cluster_id** (*UUID*) – [required]
- **visible_protocol_endpoint_ids** (*UUID*) – [required]
- **bindings** (*int*) – [required]

- **initiator_names** (*str*) – [required]
- **host_address** (*str*) – [required]

bindings = <type 'int[]'>

cluster_id = <class 'uuid.UUID'>

host_address = <type 'str'>

initiator_names = <type 'str[]'>

virtual_volume_host_id = <class 'uuid.UUID'>

visible_protocol_endpoint_ids = <class 'uuid.UUID[]'>

class solidfire.models.**VirtualVolumeInfo**(*virtual_volume_id*, *parent_virtual_volume_id*, *storage_container*, *volume_id*, *snapshot_id*, *virtual_volume_type*, *status*, *bindings*, *children*, *metadata*, *snapshot_info=None*, *volume_info=None*, *descendants=None*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_volume_id** (*UUID*) – [required]
- **parent_virtual_volume_id** (*UUID*) – [required]
- **storage_container** (StorageContainer) – [required]
- **volume_id** (*int*) – [required]
- **snapshot_id** (*int*) – [required]
- **virtual_volume_type** (*str*) – [required]
- **status** (*str*) – [required]
- **bindings** (*int*) – [required]
- **children** (*UUID*) – [required]
- **metadata** (*dict*) – [required]
- **snapshot_info** (Snapshot) –
- **volume_info** (Volume) –
- **descendants** (*int*) –

bindings = <type 'int[]'>

children = <class 'uuid.UUID[]'>

descendants = <type 'int[]'>

metadata = <type 'dict'>

parent_virtual_volume_id = <class 'uuid.UUID'>

snapshot_id = <type 'int'>

snapshot_info = <class 'solidfire.models.Snapshot'>

status = <type 'str'>

storage_container = <class 'solidfire.models.StorageContainer'>

virtual_volume_id = <class 'uuid.UUID'>

virtual_volume_type = <type 'str'>

volume_id = <type 'int'>

volume_info = <class 'solidfire.models.Volume'>

class solidfire.models.**VirtualVolumeStats**(*account_id*, *non_zero_blocks*, *read_bytes*, *read_ops*, *timestamp*, *unaligned_reads*, *unaligned_writes*, *volume_access_groups*, *volume_id*, *volume_size*, *write_bytes*, *write_ops*, *zero_blocks*, *actual_iops=None*, *async_delay=None*, *average_iopsize=None*, *burst_iopscredit=None*, *client_queue_depth=None*, *desired_metadata_hosts=None*, *latency_usec=None*, *metadata_hosts=None*, *read_latency_usec=None*, *throttle=None*, *total_latency_usec=None*, *volume_utilization=None*, *write_latency_usec=None*, *write_bytes_last_sample=None*, *sample_period_msec=None*, *read_bytes_last_sample=None*, *read_ops_last_sample=None*, *write_ops_last_sample=None*, *virtual_volume_id=None*)

Bases: *solidfire.common.model.DataObject*

Contains statistical data for an individual volume.

Parameters

- **account_id** (*int*) – [required] AccountID of the volume owner.
- **actual_iops** (*int*) – Current actual IOPS to the volume in the last 500 milliseconds.
- **async_delay** (*str*) – The length of time since the volume was last synced with the remote cluster. If the volume is not paired, this is null. Note: A target volume in an active replication state always has an async delay of 0 (zero). Target volumes are system-aware during replication and assume async delay is accurate at all times.
- **average_iopsize** (*int*) – Average size in bytes of recent I/O to the volume in the last 500 milliseconds.
- **burst_iopscredit** (*int*) – The total number of IOP credits available to the user. When users are not using up to the max IOPS, credits are accrued.
- **client_queue_depth** (*int*) – The number of outstanding read and write operations to the cluster.
- **desired_metadata_hosts** (MetadataHosts) – The volume services being migrated to if the volume metadata is getting migrated between volume services. A "null" value means the volume is not migrating.
- **latency_usec** (*int*) – The observed latency time, in microseconds, to complete operations to a volume. A "0" (zero) value means there is no I/O to the volume.
- **metadata_hosts** (MetadataHosts) – The volume services on which the volume metadata resides.
- **non_zero_blocks** (*int*) – [required] The number of 4KiB blocks with data after the last garbage collection operation has completed.

- **read_bytes** (*int*) – [required] Total bytes read by clients.
- **read_latency_usec** (*int*) – The average time, in microseconds, to complete read operations.
- **read_ops** (*int*) – [required] Total read operations.
- **throttle** (*float*) – A floating value between 0 and 1 that represents how much the system is throttling clients below their max IOPS because of re-replication of data, transient errors and snapshots taken.
- **timestamp** (*str*) – [required] The current time in UTC.
- **total_latency_usec** (*int*) – The average time, in microseconds, to complete read and write operations to a volume.
- **unaligned_reads** (*int*) – [required] For 512e volumes, the number of read operations that were not on a 4k sector boundary. High numbers of unaligned reads may indicate improper partition alignment.
- **unaligned_writes** (*int*) – [required] For 512e volumes, the number of write operations that were not on a 4k sector boundary. High numbers of unaligned writes may indicate improper partition alignment.
- **volume_access_groups** (*int*) – [required] List of volume access group(s) to which a volume beintegers.
- **volume_id** (*int*) – [required] Volume ID of the volume.
- **volume_size** (*int*) – [required] Total provisioned capacity in bytes.
- **volume_utilization** (*float*) – A floating value that describes how much the client is using the volume. Values: 0 = Client is not using the volume 1 = Client is using their max >1 = Client is using their burst
- **write_bytes** (*int*) – [required] Total bytes written by clients.
- **write_latency_usec** (*int*) – The average time, in microseconds, to complete write operations.
- **write_ops** (*int*) – [required] Total write operations occurring on the volume.
- **zero_blocks** (*int*) – [required] Total number of 4KiB blocks without data after the last round of garbage collection operation has completed.
- **write_bytes_last_sample** (*int*) – The total number of bytes written to the volume during the last sample period.
- **sample_period_msec** (*int*) – The length of the sample period in milliseconds.
- **read_bytes_last_sample** (*int*) – The total number of bytes read from the volume during the last sample period.
- **read_ops_last_sample** (*int*) – The total number of read operations durin gth elast sample period.
- **write_ops_last_sample** (*int*) – The total number of write operations during the last sample period.
- **virtual_volume_id** (*UUID*) – If the volume of interest is associated with a virtual volume, this is the virtual volume ID.

account_id = <type 'int'>

actual_iops = <type 'int'>

async_delay = <type 'str'>

average_iopsize = <type 'int'>

burst_iopscredit = <type 'int'>

client_queue_depth = <type 'int'>

desired_metadata_hosts = <class 'solidfire.models.MetadataHosts'>

latency_usec = <type 'int'>

metadata_hosts = <class 'solidfire.models.MetadataHosts'>

non_zero_blocks = <type 'int'>

read_bytes = <type 'int'>

read_bytes_last_sample = <type 'int'>

read_latency_usec = <type 'int'>

read_ops = <type 'int'>

read_ops_last_sample = <type 'int'>

sample_period_msec = <type 'int'>

throttle = <type 'float'>

timestamp = <type 'str'>

total_latency_usec = <type 'int'>

unaligned_reads = <type 'int'>

unaligned_writes = <type 'int'>

virtual_volume_id = <class 'uuid.UUID'>

volume_access_groups = <type 'int[]'>

volume_id = <type 'int'>

volume_size = <type 'int'>

volume_utilization = <type 'float'>

write_bytes = <type 'int'>

write_bytes_last_sample = <type 'int'>

write_latency_usec = <type 'int'>

write_ops = <type 'int'>

write_ops_last_sample = <type 'int'>

zero_blocks = <type 'int'>

class solidfire.models.**VirtualVolumeTask**(*virtual_volume_task_id*, *virtualvolume_id*, *clone_virtual_volume_id*, *status*, *operation*, *virtual_volume_host_id*, *parent_metadata*, *parent_total_size*, *parent_used_size*, *cancelled*)

Bases: *solidfire.common.model.DataObject*

Parameters

- **virtual_volume_task_id** (*UUID*) – [required]
- **virtualvolume_id** (*UUID*) – [required]

- **clone_virtual_volume_id** (*UUID*) – [required]
- **status** (*str*) – [required]
- **operation** (*str*) – [required]
- **virtual_volume_host_id** (*UUID*) – [required]
- **parent_metadata** (*dict*) – [required]
- **parent_total_size** (*int*) – [required]
- **parent_used_size** (*int*) – [required]
- **cancelled** (*bool*) – [required]

cancelled = <type 'bool'>

clone_virtual_volume_id = <class 'uuid.UUID'>

operation = <type 'str'>

parent_metadata = <type 'dict'>

parent_total_size = <type 'int'>

parent_used_size = <type 'int'>

status = <type 'str'>

virtual_volume_host_id = <class 'uuid.UUID'>

virtual_volume_task_id = <class 'uuid.UUID'>

virtualvolume_id = <class 'uuid.UUID'>

class solidfire.models.**Volume**(*volume_id*, *name*, *account_id*, *create_time*, *volume_consistency_group_uuid*, *volume_uuid*, *enable_snap_mirror_replication*, *status*, *access*, *enable512e*, *scsi_euidevice_id*, *scsi_naadevice_id*, *qos*, *volume_access_groups*, *volume_pairs*, *slice_count*, *total_size*, *block_size*, *attributes*, *iqn=None*, *qos_policy_id=None*, *delete_time=None*, *purge_time=None*, *last_access_time=None*, *last_access_time_io=None*, *virtual_volume_id=None*)

Bases: *solidfire.common.model.DataObject*

Volumes Info is an object containing information about a volume. The return objects only include "configured" information about the volume and not runtime or usage information. Information about paired volumes will also be returned.

Parameters

- **volume_id** (*int*) – [required] Unique VolumeID for the volume.
- **name** (*str*) – [required] Name of the volume as provided at creation time.
- **account_id** (*int*) – [required] Unique AccountID for the account.
- **create_time** (*str*) – [required] UTC formatted time the volume was created.
- **volume_consistency_group_uuid** (*UUID*) – [required]
- **volume_uuid** (*UUID*) – [required]
- **enable_snap_mirror_replication** (*bool*) – [required]
- **status** (*str*) – [required] Current status of the volume init: A volume that is being initialized and is not ready for connections. active: An active volume ready for connections.

- **access** (*str*) – [required] Access allowed for the volume readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. replicationTarget: Designated as a target volume in a replicated volume pair.
- **enable512e** (*bool*) – [required] If "true", the volume provides 512 byte sector emulation.
- **iqn** (*str*) – Volume iSCSI Qualified Name.
- **scsi_euidevice_id** (*str*) – [required] Globally unique SCSI device identifier for the volume in EUI-64 based 16-byte format.
- **scsi_naadevice_id** (*str*) – [required] Globally unique SCSI device identifier for the volume in NAA IEEE Registered Extended format.
- **qos** (VolumeQOS) – [required] Quality of service settings for this volume.
- **qos_policy_id** (*int*) – The QoS policy ID associated with the volume. The value is null if the volume is not associated with a policy.
- **volume_access_groups** (*int*) – [required] List of volume access groups to which a volume beintegers.
- **volume_pairs** (VolumePair) – [required] Information about a paired volume. Available only if a volume is paired. @see VolumePairs for return values.
- **delete_time** (*str*) – The time this volume was deleted. If this has no value, the volume has not yet been deleted.
- **purge_time** (*str*) – The time this volume will be purged from the system. If this has no value, the volume has not yet been deleted (and is not scheduled for purging).
- **last_access_time** (*str*) – The last time any access to this volume occurred. If this has no value, the last access time is not known.
- **last_access_time_io** (*str*) – The last time I/O access to this volume occurred. If this has no value, the last I/O access time is not known.
- **slice_count** (*int*) – [required] The number of slices backing this volume. In the current software, this value will always be 1.
- **total_size** (*int*) – [required] Total size of this volume in bytes.
- **block_size** (*int*) – [required] Size of the blocks on the volume.
- **virtual_volume_id** (*UUID*) – Virtual volume ID this volume backs.
- **attributes** (*dict*) – [required] List of Name/Value pairs in JSON object format.

```
access = <type 'str'>
account_id = <type 'int'>
attributes = <type 'dict'>
block_size = <type 'int'>
create_time = <type 'str'>
delete_time = <type 'str'>
enable512e = <type 'bool'>
enable_snap_mirror_replication = <type 'bool'>
iqn = <type 'str'>
```

last_access_time = <type 'str'>

last_access_time_io = <type 'str'>

name = <type 'str'>

purge_time = <type 'str'>

qos = <class 'solidfire.models.VolumeQOS'>

qos_policy_id = <type 'int'>

scsi_euidevice_id = <type 'str'>

scsi_naadevice_id = <type 'str'>

slice_count = <type 'int'>

status = <type 'str'>

total_size = <type 'int'>

virtual_volume_id = <class 'uuid.UUID'>

volume_access_groups = <type 'int[]'>

volume_consistency_group_uuid = <class 'uuid.UUID'>

volume_id = <type 'int'>

volume_pairs = <class 'solidfire.models.VolumePair[]'>

volume_uuid = <class 'uuid.UUID'>

class solidfire.models.**VolumeAccessGroup**(*deleted_volumes*, *volume_access_group_id*, *name*, *initiator_ids*, *initiators*, *volumes*, *attributes*)

Bases: *solidfire.common.model.DataObject*

A volume access group is a useful way of grouping volumes and initiators together for ease of management.

Volume Access Group Limits:

- A volume access group can contain up to sixty-four initiator IQNs.
- An initiator can only beinteger to only one volume access group.
- A volume access group can contain up to two thousand volumes.
- Each volume access group can beinteger to a maximum of four other volume access groups.

Parameters

- **deleted_volumes** (*int*) – [required] A list of deleted volumes that have yet to be purged from the VAG.
- **volume_access_group_id** (*int*) – [required] Unique ID for this volume access group.
- **name** (*str*) – [required] Name of the volume access group.
- **initiator_ids** (*int*) – [required] A list of IDs of initiators that are mapped to the VAG.
- **initiators** (*str*) – [required] List of unique initiator names beintegering to the volume access group.
- **volumes** (*int*) – [required] List of volumes beintegering to the volume access group.
- **attributes** (*dict*) – [required] List of name/value pairs

attributes = <type 'dict'>

deleted_volumes = <type 'int[]'>

initiator_ids = <type 'int[]'>

initiators = <type 'str[]'>

name = <type 'str'>

volume_access_group_id = <type 'int'>

volumes = <type 'int[]'>

class solidfire.models.**VolumeAccessGroupLunAssignments**(*volume_access_group_id*, *lun_assignments*, *deleted_lun_assignments*)

Bases: *solidfire.common.model.DataObject*

VolumeAccessGroup ID and Lun to be assigned to all volumes within it.

Parameters

- **volume_access_group_id** (*int*) – [required] Unique volume access group ID for which the LUN assignments will be modified.
- **lun_assignments** (LunAssignment) – [required] The volume IDs with assigned LUN values.
- **deleted_lun_assignments** (LunAssignment) – [required] The volume IDs with deleted LUN values.

deleted_lun_assignments = <class 'solidfire.models.LunAssignment[]'>

lun_assignments = <class 'solidfire.models.LunAssignment[]'>

volume_access_group_id = <type 'int'>

class solidfire.models.**VolumePair**(*cluster_pair_id*, *remote_volume_id*, *remote_slice_id*, *remote_volume_name*, *volume_pair_uuid*, *remote_replication*)

Bases: *solidfire.common.model.DataObject*

The Volume Pair Info is an object containing information about a volume that is paired on a remote cluster. If the volume is not paired, this object is null.

Parameters

- **cluster_pair_id** (*int*) – [required] The remote cluster a volume is paired with.
- **remote_volume_id** (*int*) – [required] The VolumeID on the remote cluster a volume is paired with.
- **remote_slice_id** (*int*) – [required] The SliceID on the remote cluster a volume is paired with.
- **remote_volume_name** (*str*) – [required] The last-observed name of the volume on the remote cluster a volume is paired with.
- **volume_pair_uuid** (*UUID*) – [required] A UUID in canonical form.
- **remote_replication** (RemoteReplication) – [required] Details about the replication configuration for this volume pair.

cluster_pair_id = <type 'int'>

remote_replication = <class 'solidfire.models.RemoteReplication'>

remote_slice_id = <type 'int'>

remote_volume_id = <type 'int'>

remote_volume_name = <type 'str'>

volume_pair_uuid = <class 'uuid.UUID'>

class solidfire.models.**VolumeQOS**(*min_iops*, *max_iops*, *burst_iops*, *burst_time*, *curve*)
Bases: *solidfire.common.model.DataObject*

Quality of Service (QoS) Result values are used on SolidFire volumes to provision performance expectations.

Parameters

- **min_iops** (*int*) – [required] Desired minimum 4KB IOPS to guarantee. The allowed IOPS will only drop below this level if all volumes have been capped at their min IOPS value and there is still insufficient performance capacity.
- **max_iops** (*int*) – [required] Desired maximum 4KB IOPS allowed over an extended period of time.
- **burst_iops** (*int*) – [required] Maximum "peak" 4KB IOPS allowed for short periods of time. Allows for bursts of I/O activity over the normal max IOPS value.
- **burst_time** (*int*) – [required] The length of time burst IOPS is allowed. The value returned is represented in time units of seconds. Note: this value is calculated by the system based on IOPS set for QoS.
- **curve** (*dict*) – [required] The curve is a set of key-value pairs. The keys are I/O sizes in bytes. The values represent the cost performing an IOP at a specific I/O size. The curve is calculated relative to a 4096 byte operation set at 100 IOPS.

burst_iops = <type 'int'>

burst_time = <type 'int'>

curve = <type 'dict'>

max_iops = <type 'int'>

min_iops = <type 'int'>

class solidfire.models.**VolumeStats**(*account_id*, *non_zero_blocks*, *read_bytes*, *read_ops*, *timestamp*, *unaligned_reads*, *unaligned_writes*, *volume_access_groups*, *volume_id*, *volume_size*, *write_bytes*, *write_ops*, *zero_blocks*, *actual_iops=None*, *average_iopsize=None*, *burst_iopscredit=None*, *client_queue_depth=None*, *latency_usec=None*, *async_delay=None*, *metadata_hosts=None*, *desired_metadata_hosts=None*, *read_latency_usec=None*, *throttle=None*, *total_latency_usec=None*, *volume_utilization=None*, *write_latency_usec=None*, *write_bytes_last_sample=None*, *sample_period_msec=None*, *read_bytes_last_sample=None*, *read_ops_last_sample=None*, *write_ops_last_sample=None*)
Bases: *solidfire.common.model.DataObject*

Contains statistical data for an individual volume.

Parameters

- **account_id** (*int*) – [required] AccountID of the volume owner.

- **actual_iops** (*int*) – Current actual IOPS to the volume in the last 500 milliseconds.
- **average_iopsize** (*int*) – Average size in bytes of recent I/O to the volume in the last 500 milliseconds.
- **burst_iopscredit** (*int*) – The total number of IOP credits available to the user. When users are not using up to the max IOPS, credits are accrued.
- **client_queue_depth** (*int*) – The number of outstanding read and write operations to the cluster.
- **latency_usec** (*int*) – The observed latency time, in microseconds, to complete operations to a volume. A "0" (zero) value means there is no I/O to the volume.
- **async_delay** (*int*) –
- **metadata_hosts** (MetadataHosts) – The volume services on which the volume metadata resides.
- **desired_metadata_hosts** (MetadataHosts) –
- **non_zero_blocks** (*int*) – [required] The number of 4KiB blocks with data after the last garbage collection operation has completed.
- **read_bytes** (*int*) – [required] Total bytes read by clients.
- **read_latency_usec** (*int*) – The average time, in microseconds, to complete read operations.
- **read_ops** (*int*) – [required] Total read operations.
- **throttle** (*float*) – A floating value between 0 and 1 that represents how much the system is throttling clients below their max IOPS because of re-replication of data, transient errors and snapshots taken.
- **timestamp** (*str*) – [required] The current time in UTC.
- **total_latency_usec** (*int*) – The average time, in microseconds, to complete read and write operations to a volume.
- **unaligned_reads** (*int*) – [required] For 512e volumes, the number of read operations that were not on a 4k sector boundary. High numbers of unaligned reads may indicate improper partition alignment.
- **unaligned_writes** (*int*) – [required] For 512e volumes, the number of write operations that were not on a 4k sector boundary. High numbers of unaligned writes may indicate improper partition alignment.
- **volume_access_groups** (*int*) – [required] List of volume access group(s) to which a volume beintegers.
- **volume_id** (*int*) – [required] Volume ID of the volume.
- **volume_size** (*int*) – [required] Total provisioned capacity in bytes.
- **volume_utilization** (*float*) – A floating value that describes how much the client is using the volume. Values: 0 = Client is not using the volume 1 = Client is using their max >1 = Client is using their burst
- **write_bytes** (*int*) – [required] Total bytes written by clients.
- **write_latency_usec** (*int*) – The average time, in microseconds, to complete write operations.
- **write_ops** (*int*) – [required] Total write operations occurring on the volume.

- **zero_blocks** (*int*) – [required] Total number of 4KiB blocks without data after the last round of garbage collection operation has completed.
- **write_bytes_last_sample** (*int*) – The total number of bytes written to the volume during the last sample period.
- **sample_period_msec** (*int*) – The length of the sample period in milliseconds.
- **read_bytes_last_sample** (*int*) – The total number of bytes read from the volume during the last sample period.
- **read_ops_last_sample** (*int*) – The total number of read operations durin gth elast sample period.
- **write_ops_last_sample** (*int*) – The total number of write operations during the last sample period.

account_id = <type 'int'>

actual_iops = <type 'int'>

async_delay = <type 'int'>

average_iopsize = <type 'int'>

burst_iopscredit = <type 'int'>

client_queue_depth = <type 'int'>

desired_metadata_hosts = <class 'solidfire.models.MetadataHosts'>

latency_usec = <type 'int'>

metadata_hosts = <class 'solidfire.models.MetadataHosts'>

non_zero_blocks = <type 'int'>

read_bytes = <type 'int'>

read_bytes_last_sample = <type 'int'>

read_latency_usec = <type 'int'>

read_ops = <type 'int'>

read_ops_last_sample = <type 'int'>

sample_period_msec = <type 'int'>

throttle = <type 'float'>

timestamp = <type 'str'>

total_latency_usec = <type 'int'>

unaligned_reads = <type 'int'>

unaligned_writes = <type 'int'>

volume_access_groups = <type 'int[]'>

volume_id = <type 'int'>

volume_size = <type 'int'>

volume_utilization = <type 'float'>

write_bytes = <type 'int'>

write_bytes_last_sample = <type 'int'>

write_latency_usec = <type 'int'>

write_ops = <type 'int'>

write_ops_last_sample = <type 'int'>

zero_blocks = <type 'int'>

4.5 Module contents

class solidfire.**Element**(*mvip=None*, *username=None*, *password=None*, *api_version=8.0*, *verify_ssl=True*, *dispatcher=None*)

Bases: *solidfire.common.ServiceBase*

The API for controlling a SolidFire cluster.

Constructor for initializing a connection to an instance of Element OS

Parameters

- **mvip** (*str*) – the management IP (IP or hostname)
- **username** (*str*) – username use to connect to the Element OS instance.
- **password** (*str*) – authentication for username
- **api_version** (*float or str*) – specific version of Element OS to connect
- **verify_ssl** (*bool*) – disable to avoid ssl connection errors especially when using an IP instead of a hostname
- **dispatcher** – a prebuilt or custom http dispatcher

Returns a configured and tested instance of Element

add_account(*username*, *initiator_secret=None*, *target_secret=None*, *attributes=None*)

You can use AddAccount to add a new account to the system. You can create new volumes under the new account. The CHAP settings you specify for the account apply to all volumes owned by the account. :param username: [required] Specifies the username for this account. (Might be 1 to 64 characters in length). :type username: str

Parameters

- **initiatorSecret** (CHAPSecret) – The CHAP secret to use for the initiator. This secret must be 12-16 characters in length and should be impenetrable. The initiator CHAP secret must be unique and cannot be the same as the target CHAP secret. If unspecified, a random secret is created.
- **targetSecret** (CHAPSecret) – The CHAP secret to use for the target (mutual CHAP authentication). This secret must be 12-16 characters in length and should be impenetrable. The target CHAP secret must be unique and cannot be the same as the initiator CHAP secret. If unspecified, a random secret is created.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

add_cluster_admin(*username*, *password*, *access*, *accept_eula=None*, *attributes=None*)

You can use AddClusterAdmin to add a new cluster admin account. A cluster ddmin can manage the cluster using the API and management tools. Cluster admins are completely separate and unrelated to standard tenant accounts. Each cluster admin can be restricted to a subset of the API. NetApp recommends using multiple cluster admin accounts for different users and applications. You should give each cluster admin the minimal permissions necessary; this reduces the potential impact of credential compromise. You must accept the End User License Agreement (EULA) by setting the acceptEula parameter to true to add a

cluster administrator account to the system. :param username: [required] Unique username for this cluster admin. Must be between 1 and 1024 characters in length. :type username: str

Parameters

- **password** (*str*) – [required] Password used to authenticate this cluster admin.
- **access** (*str*) – [required] Controls which methods this cluster admin can use. For more details on the levels of access, see Access Control in the Element API Reference Guide.
- **acceptEula** (*bool*) – Required to indicate your acceptance of the End User License Agreement when creating this cluster. To accept the EULA, set this parameter to true.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

add_drives(*drives*, *force_during_upgrade=None*, *force_during_bin_sync=None*)

AddDrives enables you to add one or more available drives to the cluster, enabling the drives to host a portion of the cluster's data. When you add a node to the cluster or install new drives in an existing node, the new drives are marked as "available" and must be added via AddDrives before they can be utilized. Use the ListDrives method to display drives that are "available" to be added. When you add multiple drives, it is more efficient to add them in a single AddDrives method call rather than multiple individual methods with a single drive each. This reduces the amount of data balancing that must occur to stabilize the storage load on the cluster. When you add a drive, the system automatically determines the "type" of drive it should be. The method is asynchronous and returns immediately. However, it can take some time for the data in the cluster to be rebalanced using the newly added drives. As the new drives are syncing on the system, you can use the ListSyncJobs method to see how the drives are being rebalanced and the progress of adding the new drive. You can also use the GetAsyncResult method to query the method's returned asyncHandle. :param drives: [required] Returns information about each drive to be added to the cluster. Possible values are: driveID: The ID of the drive to add. (Integer) type: (Optional) The type of drive to add. Valid values are "slice" or "block". If omitted, the system assigns the correct type. (String) :type drives: NewDrive

Parameters

- **forceDuringUpgrade** (*bool*) – Allows the user to force the addition of drives during an upgrade.
- **forceDuringBinSync** (*bool*) – Allows the user to force the addition of drives during a bin sync operation.

add_initiators_to_volume_access_group(*volume_access_group_id*, *initiators*)

AddInitiatorsToVolumeAccessGroup enables you to add initiators to a specified volume access group. :param volumeAccessGroupID: [required] The ID of the volume access group to modify. :type volumeAccessGroupID: int

Parameters **initiators** (*str*) – [required] The list of initiators to add to the volume access group.

add_ldap_cluster_admin(*username*, *access*, *accept_eula=None*, *attributes=None*)

AddLdapClusterAdmin enables you to add a new LDAP cluster administrator user. An LDAP cluster administrator can manage the cluster via the API and management tools. LDAP cluster admin accounts are completely separate and unrelated to standard tenant accounts. You can also use this method to add an LDAP group that has been defined in Active Directory. The access level that is given to the group is passed to the individual users in the LDAP group. :param username: [required] The distinguished user name for the new LDAP cluster admin. :type username: str

Parameters

- **access** (*str*) – [required] Controls which methods this Cluster Admin can use. For more details on the levels of access, see the Access Control appendix in the SolidFire API Reference.

- **acceptEula** (*bool*) – Accept the End User License Agreement. Set to true to add a cluster administrator account to the system. If omitted or set to false, the method call fails.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

add_nodes(*pending_nodes*, *auto_install=None*)

AddNodes enables you to add one or more new nodes to a cluster. When a node that is not configured starts up for the first time, you are prompted to configure the node. After you configure the node, it is registered as a "pending node" with the cluster. Note: It might take several seconds after adding a new node for it to start up and register its drives as available. :param pendingNodes: [required] List of pending NodeIDs for the nodes to be added. You can obtain the list of pending nodes using the ListPendingNodes method. :type pendingNodes: int

Parameters **auto_Install** – Whether these nodes should be autoinstalled

add_virtual_network(*virtual_network_tag*, *name*, *address_blocks*, *netmask*, *svip*, *gateway=None*, *namespace=None*, *attributes=None*)

You can use the AddVirtualNetwork method to add a new virtual network to a cluster configuration. When you add a virtual network, an interface for each node is created and each interface will require a virtual network IP address. The number of IP addresses you specify as a parameter for this API method must be equal to or greater than the number of nodes in the cluster. The system bulk provisions virtual network addresses and assigns them to individual nodes automatically. You do not need to assign virtual network addresses to nodes manually. Note: You can use AddVirtualNetwork only to create a new virtual network. If you want to make changes to an existing virtual network, use ModifyVirtualNetwork. Note: Virtual network parameters must be unique to each virtual network when setting the namespace parameter to false. :param virtualNetworkTag: [required] A unique virtual network (VLAN) tag. Supported values are 1 through 4094.The number zero (0) is not supported. :type virtualNetworkTag: int

Parameters

- **name** (*str*) – [required] A user-defined name for the new virtual network.
- **addressBlocks** (AddressBlockParams) – [required] Unique range of IP addresses to include in the virtual network. Attributes for this parameter are: start: The start of the IP address range. (String) size: The number of IP addresses to include in the block. (Integer)
- **netmask** (*str*) – [required] Unique network mask for the virtual network being created.
- **svip** (*str*) – [required] Unique storage IP address for the virtual network being created.
- **gateway** (*str*) – The IP address of a gateway of the virtual network. This parameter is only valid if the "namespace" parameter is set to true.
- **namespace** (*bool*) – When set to true, enables the Routable Storage VLANs functionality by creating and configuring a namespace and the virtual network contained by it.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

add_volumes_to_volume_access_group(*volume_access_group_id*, *volumes*)

AddVolumesToVolumeAccessGroup enables you to add volumes to a specified volume access group. :param volumeAccessGroupID: [required] The ID of the volume access group to which volumes are added. :type volumeAccessGroupID: int

Parameters **volumes** (*int*) – [required] The list of volumes to add to the volume access group.

cancel_clone(*clone_id*)

CancelClone enables you to stop an ongoing CloneVolume or CopyVolume process. When you cancel a group clone operation, the system completes and removes the operation's associated asyncHandle. :param cloneID: [required] The cloneID for the ongoing clone process. :type cloneID: int

cancel_group_clone(*group_clone_id*)

CancelGroupClone enables you to stop an ongoing CloneMultipleVolumes process occurring on a group of volumes. When you cancel a group clone operation, the system completes and removes the operation's associated asyncHandle. :param groupCloneID: [required] The cloneID for the ongoing clone process. :type groupCloneID: int

clear_cluster_faults(*fault_types=None*)

You can use the ClearClusterFaults method to clear information about both current and previously detected faults. Both resolved and unresolved faults can be cleared. :param faultTypes: Determines the types of faults cleared. Possible values are: current: Faults that are currently detected and have not been resolved. resolved: (Default) Faults that were previously detected and resolved. all: Both current and resolved faults are cleared. The fault status can be determined by the resolved field of the fault object. :type faultTypes: str

clone_multiple_volumes(*volumes*, *access=None*, *group_snapshot_id=None*, *new_account_id=None*)

CloneMultipleVolumes enables you to create a clone of a group of specified volumes. You can assign a consistent set of characteristics to a group of multiple volumes when they are cloned together. Before using groupSnapshotID to clone the volumes in a group snapshot, you must create the group snapshot by using the CreateGroupSnapshot API method or the Element OS Web UI. Using groupSnapshotID is optional when cloning multiple volumes. Note: Cloning multiple volumes is allowed if cluster fullness is at stage 2 or 3. Clones are not created when cluster fullness is at stage 4 or 5. :param volumes: [required] Unique ID for each volume to include in the clone. If optional parameters are not specified, the values are inherited from the source volumes. Required parameter for "volumes" array: volumeID Optional parameters for "volumes" array: access: Can be one of readOnly, readWrite, locked, or replicationTarget attributes: List of name-value pairs in JSON object format. name: New name for the clone. newAccountID: Account ID for the new volumes. newSize: New size Total size of the volume, in bytes. Size is rounded up to the nearest 1MB. :type volumes: CloneMultipleVolumeParams

Parameters

- **access** (*str*) – New default access method for the new volumes if not overridden by information passed in the volume's array.
- **groupSnapshotID** (*int*) – ID of the group snapshot to use as a basis for the clone.
- **newAccountID** (*int*) – New account ID for the volumes if not overridden by information passed in the volumes array.

clone_volume(*volume_id*, *name*, *new_account_id=None*, *new_size=None*, *access=None*, *snapshot_id=None*, *attributes=None*, *enable512e=None*)

CloneVolume enables you to create a copy of a volume. This method is asynchronous and might take a variable amount of time to complete. The cloning process begins immediately when you make the CloneVolume request and is representative of the state of the volume when the API method is issued. You can use the GetAsyncResult method to determine when the cloning process is complete and the new volume is available for connections. You can use ListSyncJobs to see the progress of creating the clone. Note: The initial attributes and QoS settings for the volume are inherited from the volume being cloned. You can change these settings with ModifyVolume. Note: Cloned volumes do not inherit volume access group memberships from the source volume. :param volumeID: [required] VolumeID for the volume to be cloned. :type volumeID: int

Parameters

- **name** (*str*) – [required] The name of the new cloned volume. Might be 1 to 64 characters in length.
- **newAccountID** (*int*) – AccountID for the owner of the new volume. If unspecified, the accountID of the owner of the volume being cloned is used.

- **newSize** (*int*) – New size of the volume, in bytes. Might be greater or less than the size of the volume being cloned. If unspecified, the volume size is not changed. Size is rounded to the nearest 1MB.
- **access** (*str*) – Specifies the level of access allowed for the new volume. Possible values are: readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. If unspecified, the level of access of the volume being cloned is used. replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked. If a value is not specified, the access value does not change.
- **snapshotID** (*int*) – ID of the snapshot that is used as the source of the clone. If no ID is provided, the current active volume is used.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **enable512e** (*bool*) – Should the volume provide 512-byte sector emulation?

complete_cluster_pairing(*cluster_pairing_key*)

You can use the CompleteClusterPairing method with the encoded key received from the StartClusterPairing method to complete the cluster pairing process. The CompleteClusterPairing method is the second step in the cluster pairing process. :param clusterPairingKey: [required] A string of characters that is returned from the "StartClusterPairing" API method. :type clusterPairingKey: str

complete_volume_pairing(*volume_pairing_key*, *volume_id*)

You can use the CompleteVolumePairing method to complete the pairing of two volumes. :param volumePairingKey: [required] The key returned from the StartVolumePairing method. :type volumePairingKey: str

Parameters **volumeID** (*int*) – [required] The ID of the volume on which to complete the pairing process.

copy_volume(*volume_id*, *dst_volume_id*, *snapshot_id=None*)

CopyVolume enables you to overwrite the data contents of an existing volume with the data contents of another volume (or snapshot). Attributes of the destination volume such as IQN, QoS settings, size, account, and volume access group membership are not changed. The destination volume must already exist and must be the same size as the source volume. NetApp strongly recommends that clients unmount the destination volume before the CopyVolume operation begins. If the destination volume is modified during the copy operation, the changes will be lost. This method is asynchronous and may take a variable amount of time to complete. You can use the GetAsyncResult method to determine when the process has finished, and ListSyncJobs to see the progress of the copy. :param volumeID: [required] VolumeID of the volume to be read from. :type volumeID: int

Parameters

- **dstVolumeID** (*int*) – [required] VolumeID of the volume to be overwritten.
- **snapshotID** (*int*) – ID of the snapshot that is used as the source of the clone. If no ID is provided, the current active volume is used.

create_backup_target(*name*, *attributes*)

CreateBackupTarget enables you to create and store backup target information so that you do not need to re-enter it each time a backup is created. :param name: [required] The name of the backup target. :type name: str

Parameters **attributes** (*dict*) – [required] List of name-value pairs in JSON object format.

create_cluster(*mvip*, *svip*, *rep_count*, *username*, *password*, *nodes*, *accept_eula=None*, *attributes=None*)

The CreateCluster method enables you to initialize the node in a cluster that has ownership of the "mvip"

and "svip" addresses. Each new cluster is initialized using the management IP (MIP) of the first node in the cluster. This method also automatically adds all the nodes being configured into the cluster. You only need to use this method once each time a new cluster is initialized. Note: You need to log in to the node that is used as the master node for the cluster. After you log in, run the GetBootstrapConfig method on the node to get the IP addresses for the rest of the nodes that you want to include in the cluster. Then, run the CreateCluster method. :param acceptEula: Required to indicate your acceptance of the End User License Agreement when creating this cluster. To accept the EULA, set this parameter to true. :type acceptEula: bool

Parameters

- **mvip** (*str*) – [required] Floating (virtual) IP address for the cluster on the management network.
- **svip** (*str*) – [required] Floating (virtual) IP address for the cluster on the storage (iSCSI) network.
- **repCount** (*int*) – [required] Number of replicas of each piece of data to store in the cluster. Valid value is "2".
- **username** (*str*) – [required] Username for the cluster admin.
- **password** (*str*) – [required] Initial password for the cluster admin account.
- **nodes** (*str*) – [required] CIP/SIP addresses of the initial set of nodes making up the cluster. This node's IP must be in the list.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

create_group_snapshot(*volumes*, *name=None*, *enable_remote_replication=None*, *retention=None*, *attributes=None*, *snap_mirror_label=None*)

CreateGroupSnapshot enables you to create a point-in-time copy of a group of volumes. You can use this snapshot later as a backup or rollback to ensure the data on the group of volumes is consistent for the point in time that you created the snapshot. Note: Creating a group snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5. :param volumes: [required] Unique ID of the volume image from which to copy. :type volumes: int

Parameters

- **name** (*str*) – Name for the group snapshot. If unspecified, the date and time the group snapshot was taken is used.
- **enableRemoteReplication** (*bool*) – Replicates the snapshot created to remote storage. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **retention** (*str*) – Specifies the amount of time for which the snapshots are retained. The format is HH:mm:ss.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **snapMirrorLabel** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

create_initiators(*initiators*)

CreateInitiators enables you to create multiple new initiator IQNs or World Wide Port Names (WWPNs) and optionally assign them aliases and attributes. When you use CreateInitiators to create new initiators, you can also add them to volume access groups. If CreateInitiators fails to create one of the initiators provided in the parameter, the method returns an error and does not create any initiators (no partial completion is possible). :param initiators: [required] A list of objects containing characteristics of each new initiator. Values are: name: (Required) The name of the initiator (IQN or WWPN) to create. (String) alias: (Optional) The friendly name to assign to this initiator. (String) attributes: (Optional) A set of JSON attributes

to assign to this initiator. (JSON Object) volumeAccessGroupID: (Optional) The ID of the volume access group into to which this newly created initiator will be added. (Integer) :type initiators: CreateInitiator

create_qos_policy(*name*, *qos*)

You can use the CreateQoSPolicy method to create a QoSPolicy object that you can later apply to a volume upon creation or modification. A QoS policy has a unique ID, a name, and QoS settings. :param name: [required] The name of the QoS policy; for example, gold, platinum, or silver. :type name: str

Parameters **qos** (QoS) – [required] The QoS settings that this policy represents.

create_schedule(*schedule*)

CreateSchedule enables you to schedule an automatic snapshot of a volume at a defined interval. You can use the created snapshot later as a backup or rollback to ensure the data on a volume or group of volumes is consistent for the point in time in which the snapshot was created. If you schedule a snapshot to run at a time period that is not divisible by 5 minutes, the snapshot runs at the next time period that is divisible by 5 minutes. For example, if you schedule a snapshot to run at 12:42:00 UTC, it runs at 12:45:00 UTC. Note: You can create snapshots if cluster fullness is at stage 1, 2 or 3. You cannot create snapshots after cluster fullness reaches stage 4 or 5. :param schedule: [required] The "Schedule" object will be used to create a new schedule. Do not set ScheduleID property, it will be ignored. Frequency property must be of type that inherits from Frequency. Valid types are: DaysOfMonthFrequency DaysOrWeekFrequency TimeIntervalFrequency :type schedule: Schedule

create_snapshot(*volume_id*, *snapshot_id=None*, *name=None*, *enable_remote_replication=None*, *retention=None*, *attributes=None*, *snap_mirror_label=None*)

CreateSnapshot enables you to create a point-in-time copy of a volume. You can create a snapshot from any volume or from an existing snapshot. If you do not provide a SnapshotID with this API method, a snapshot is created from the volume's active branch. If the volume from which the snapshot is created is being replicated to a remote cluster, the snapshot can also be replicated to the same target. Use the enableRemoteReplication parameter to enable snapshot replication. Note: Creating a snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5. :param volumeID: [required] Specifies the unique ID of the volume image from which to copy. :type volumeID: int

Parameters

- **snapshotID** (*int*) – Specifies the unique ID of a snapshot from which the new snapshot is made. The snapshotID passed must be a snapshot on the given volume.
- **name** (*str*) – Specifies a name for the snapshot. If unspecified, the date and time the snapshot was taken is used.
- **enableRemoteReplication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **retention** (*str*) – Specifies the amount of time for which the snapshot is retained. The format is HH:mm:ss.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **snapMirrorLabel** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

create_storage_container(*name*, *initiator_secret=None*, *target_secret=None*, *account_id=None*)

CreateStorageContainer enables you to create a Virtual Volume (VVol) storage container. Storage containers are associated with a SolidFire storage system account, and are used for reporting and resource allocation. Storage containers can only be associated with virtual volumes. You need at least one storage container to use the Virtual Volumes feature. :param name: [required] The name of the storage container. Follows SolidFire account naming restrictions. :type name: str

Parameters

- **initiatorSecret** (*str*) – The secret for CHAP authentication for the initiator.
- **targetSecret** (*str*) – The secret for CHAP authentication for the target.
- **accountID** (*int*) – Non-storage container account that will become a storage container.

create_support_bundle(*bundle_name=None*, *extra_args=None*, *timeout_sec=None*)

CreateSupportBundle enables you to create a support bundle file under the node's directory. After creation, the bundle is stored on the node as a tar.gz file. :param bundleName: The unique name for the support bundle. If no name is provided, "supportbundle" and the node name are used as the filename. :type bundleName: str

Parameters

- **extraArgs** (*str*) – Passed to the sf_make_support_bundle script. You should use this parameter only at the request of NetApp SolidFire Support.
- **timeoutSec** (*int*) – The number of seconds to allow the support bundle script to run before stopping. The default value is 1500 seconds.

create_volume(*name*, *account_id*, *total_size*, *enable512e*, *qos=None*, *attributes=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*)

CreateVolume enables you to create a new (empty) volume on the cluster. As soon as the volume creation is complete, the volume is available for connection via iSCSI. :param name: [required] The name of the volume access group (might be user specified). Not required to be unique, but recommended. Might be 1 to 64 characters in length. :type name: str

Parameters

- **accountID** (*int*) – [required] AccountID for the owner of this volume.
- **totalSize** (*int*) – [required] Total size of the volume, in bytes. Size is rounded up to the nearest 1MB size.
- **enable512e** (*bool*) – [required] Specifies whether 512e emulation is enabled or not. Possible values are: true: The volume provides 512-byte sector emulation. false: 512e emulation is not enabled.
- **qos** (QoS) – Initial quality of service settings for this volume. Default values are used if none are specified. Valid settings are: minIOPS maxIOPS burstIOPS You can get the default values for a volume by using the GetDefaultQoS method.
- **attributes** (*dict*) – The list of name-value pairs in JSON object format. Total attribute size must be less than 1000B, or 1KB, including JSON formatting characters.
- **associateWithQoSPolicy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.
- **qosPolicyID** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. This parameter is mutually exclusive with the qos parameter.

create_volume_access_group(*name*, *initiators=None*, *volumes=None*, *virtual_network_id=None*, *virtual_network_tags=None*, *attributes=None*)

You can use CreateVolumeAccessGroup to create a new volume access group. When you create the volume access group, you need to give it a name, and you can optionally enter initiators and volumes. After you create the group, you can add volumes and initiator IQNs. Any initiator IQN that you add to the volume access group is able to access any volume in the group without CHAP authentication. :param name:

[required] The name for this volume access group. Not required to be unique, but recommended. :type name: str

Parameters

- **initiators** (*str*) – List of initiators to include in the volume access group. If unspecified, the access group's configured initiators are not modified.
- **volumes** (*int*) – List of volumes to initially include in the volume access group. If unspecified, the access group's volumes are not modified.
- **virtualNetworkID** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **virtualNetworkTags** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

delete_all_support_bundles()

DeleteAllSupportBundles enables you to delete all support bundles generated with the CreateSupportBundle API method.

delete_group_snapshot(*group_snapshot_id*, *save_members*)

DeleteGroupSnapshot enables you to delete a group snapshot. You can use the saveMembers parameter to preserve all the snapshots that were made for the volumes in the group, but the group association is removed. :param groupSnapshotID: [required] Specifies the unique ID of the group snapshot. :type groupSnapshotID: int

Parameters **saveMembers** (*bool*) – [required] Specifies whether to preserve snapshots or delete them. Valid values are: true: Snapshots are preserved, but group association is removed. false: The group and snapshots are deleted.

delete_initiators(*initiators*)

DeleteInitiators enables you to delete one or more initiators from the system (and from any associated volumes or volume access groups). If DeleteInitiators fails to delete one of the initiators provided in the parameter, the system returns an error and does not delete any initiators (no partial completion is possible). :param initiators: [required] An array of IDs of initiators to delete. :type initiators: int

delete_qos_policy(*qos_policy_id*)

You can use the DeleteQoSPolicy method to delete a QoS policy from the system. The QoS settings for all volumes created of modified with this policy are unaffected. :param qosPolicyID: [required] The ID of the QoS policy to be deleted. :type qosPolicyID: int

delete_snapshot(*snapshot_id*)

DeleteSnapshot enables you to delete a snapshot. A snapshot that is currently the "active" snapshot cannot be deleted. You must rollback and make another snapshot "active" before the current snapshot can be deleted. For more details on rolling back snapshots, see RollbackToSnapshot. :param snapshotID: [required] The ID of the snapshot to be deleted. :type snapshotID: int

delete_storage_containers(*storage_container_ids*)

DeleteStorageContainers enables you to remove up to 2000 Virtual Volume (VVol) storage containers from the system at one time. The storage containers you remove must not contain any VVols. :param storageContainerIDs: [required] A list of IDs of the storage containers to delete. You can specify up to 2000 IDs in the list. :type storageContainerIDs: UUID

delete_volume(*volume_id*)

DeleteVolume marks an active volume for deletion. When marked, the volume is purged (permanently deleted) after the cleanup interval elapses. After making a request to delete a volume, any active iSCSI connections to the volume are immediately terminated and no further connections are allowed while the volume is in this state. A marked volume is not returned in target discovery requests. Any snapshots of a

volume that has been marked for deletion are not affected. Snapshots are kept until the volume is purged from the system. If a volume is marked for deletion and has a bulk volume read or bulk volume write operation in progress, the bulk volume read or write operation is stopped. If the volume you delete is paired with a volume, replication between the paired volumes is suspended and no data is transferred to it or from it while in a deleted state. The remote volume that the deleted volume was paired with enters into a PausedMisconfigured state and data is no longer sent to it or from the deleted volume. Until the deleted volume is purged, it can be restored and data transfers resume. If the deleted volume gets purged from the system, the volume it was paired with enters into a StoppedMisconfigured state and the volume pairing status is removed. The purged volume becomes permanently unavailable. :param volumeID: [required] The ID of the volume to be deleted. :type volumeID: int

delete_volume_access_group(*volume_access_group_id*, *delete_orphan_initiators=None*, *force=None*)

DeleteVolumeAccessGroup enables you to delete a volume access group. :param volumeAccessGroupID: [required] The ID of the volume access group to be deleted. :type volumeAccessGroupID: int

Parameters

- **deleteOrphanInitiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.
- **force** (*bool*) – Adding this flag will force the volume access group to be deleted even though it has a Virtual Network ID or Tag. true: Volume access group will be deleted. false: Default. Do not delete the volume access group if it has a Virtual Network ID or Tag.

delete_volumes(*account_ids=None*, *volume_access_group_ids=None*, *volume_ids=None*)

DeleteVolumes marks multiple (up to 500) active volumes for deletion. Once marked, the volumes are purged (permanently deleted) after the cleanup interval elapses. The cleanup interval can be set in the SetClusterSettings method. For more information on using this method, see SetClusterSettings on page 1. After making a request to delete volumes, any active iSCSI connections to the volumes are immediately terminated and no further connections are allowed while the volumes are in this state. A marked volume is not returned in target discovery requests. Any snapshots of a volume that has been marked for deletion are not affected. Snapshots are kept until the volume is purged from the system. If a volume is marked for deletion and has a bulk volume read or bulk volume write operation in progress, the bulk volume read or write operation is stopped. If the volumes you delete are paired with a volume, replication between the paired volumes is suspended and no data is transferred to them or from them while in a deleted state. The remote volumes the deleted volumes were paired with enter into a PausedMisconfigured state and data is no longer sent to them or from the deleted volumes. Until the deleted volumes are purged, they can be restored and data transfers resume. If the deleted volumes are purged from the system, the volumes they were paired with enter into a StoppedMisconfigured state and the volume pairing status is removed. The purged volumes become permanently unavailable. :param accountIDs: A list of account IDs. All volumes from these accounts are deleted from the system. :type accountIDs: int

Parameters

- **volumeAccessGroupIDs** (*int*) – A list of volume access group IDs. All of the volumes from all of the volume access groups you specify in this list are deleted from the system.
- **volumeIDs** (*int*) – The list of IDs of the volumes to delete from the system.

disable_encryption_at_rest()

The DisableEncryptionAtRest method enables you to remove the encryption that was previously applied to the cluster using the EnableEncryptionAtRest method. This disable method is asynchronous and returns a response before encryption is disabled. You can use the GetClusterInfo method to poll the system to see when the process has completed.

disable_ldap_authentication()
The DisableLdapAuthentication method enables you to disable LDAP authentication and remove all LDAP configuration settings. This method does not remove any configured cluster admin accounts (user or group). However, those cluster admin accounts will no longer be able to log in.

disable_snmp()
You can use DisableSnmp to disable SNMP on the cluster nodes.

enable_encryption_at_rest()
You can use the EnableEncryptionAtRest method to enable the Advanced Encryption Standard (AES) 256-bit encryption at rest on the cluster, so that the cluster can manage the encryption key used for the drives on each node. This feature is not enabled by default. When you enable Encryption at Rest, the cluster automatically manages encryption keys internally for the drives on each node in the cluster. Nodes do not store the keys to unlock drives and the keys are never passed over the network. Two nodes participating in a cluster are required to access the key to disable encryption on a drive. The encryption management does not affect performance or efficiency on the cluster. If an encryption-enabled drive or node is removed from the cluster with the API, Encryption at Rest is disabled and the data is not secure erased. Data can be secure erased using the SecureEraseDrives API method. Note: If you have a node type with a model number ending in "-NE", the EnableEncryptionAtRest method call fails with a response of "Encryption not allowed. Cluster detected non-encryptable node". You should only enable or disable encryption when the cluster is running and in a healthy state. You can enable or disable encryption at your discretion and as often as you need. Note: This process is asynchronous and returns a response before encryption is enabled. You can use the GetClusterInfo method to poll the system to see when the process has completed.

enable_feature(*feature*)
You can use EnableFeature to enable cluster features that are disabled by default. :param feature: [required] Indicates which feature to enable. Valid value is: vvols: Enable the NetApp SolidFire VVols cluster feature. :type feature: str

enable_ldap_authentication(*server_uris*, *auth_type=None*, *group_search_base_dn=None*, *group_search_custom_filter=None*, *group_search_type=None*, *search_bind_dn=None*, *search_bind_password=None*, *user_dntemplate=None*, *user_search_base_dn=None*, *user_search_filter=None*)
The EnableLdapAuthentication method enables you to configure an LDAP directory connection to use for LDAP authentication to a cluster. Users that are members of the LDAP directory can then log in to the storage system using their LDAP credentials. :param authType: Identifies which user authentication method to use. Must be one of the following: DirectBind SearchAndBind :type authType: str

Parameters

- **groupSearchBaseDN** (*str*) – The base DN of the tree to start the group search (will do a subtree search from here).
- **groupSearchCustomFilter** (*str*) – For use with the CustomFilter search type, an LDAP filter to use to return the DNs of a users groups. The string can have placeholder text of %USERNAME% and %USERDN% to be replaced with their username and full userDN as needed.
- **groupSearchType** (*str*) – Controls the default group search filter used, and must be one of the following: NoGroups: No group support. ActiveDirectory: Nested membership of all of a users AD groups. MemberDN: MemberDN style groups (single level).
- **searchBindDN** (*str*) – A fully qualified DN to log in with to perform an LDAP search for the user (needs read access to the LDAP directory).
- **searchBindPassword** (*str*) – The password for the searchBindDN account used for searching.

- **serverURIs** (*str*) – [required] A comma-separated list of LDAP server URIs (examples: "ldap://1.2.3.4" and ldaps://1.2.3.4:123")
- **userDNTemplate** (*str*) – A string that is used to form a fully qualified user DN. The string should have the placeholder text %USERNAME%, which is replaced with the username of the authenticating user.
- **userSearchBaseDN** (*str*) – The base DN of the tree to start the search (will do a subtree search from here).
- **userSearchFilter** (*str*) – The LDAP filter to use. The string should have the placeholder text %USERNAME% which is replaced with the username of the authenticating user. Example: (&(objectClass=person)(sAMAccountName=%USERNAME%)) will use the sAMAccountName field in Active Directory to match the username entered at cluster login.

enable_snmp(*snmp_v3_enabled*)
EnableSnmp enables you to enable SNMP on cluster nodes. When you enable SNMP, the action applies to all nodes in the cluster, and the values that are passed replace, in whole, all values set in any previous call to EnableSnmp. :param snmpV3Enabled: [required] If set to "true", then SNMP v3 is enabled on each node in the cluster. If set to "false", then SNMP v2 is enabled. :type snmpV3Enabled: bool

get_account_by_id(*account_id*)
GetAccountByID enables you to return details about a specific account, given its accountID. :param accountID: [required] Specifies the account for which details are gathered. :type accountID: int

get_account_by_name(*username*)
GetAccountByName enables you to retrieve details about a specific account, given its username. :param username: [required] Username for the account. :type username: str

get_account_efficiency(*account_id*)
GetAccountEfficiency enables you to retrieve efficiency statistics about a volume account. This method returns efficiency information only for the account you specify as a parameter. :param accountID: [required] Specifies the volume account for which efficiency statistics are returned. :type accountID: int

get_api()
You can use the GetAPI method to return a list of all the API methods and supported API endpoints that can be used in the system.

get_async_result(*async_handle*, *keep_result=None*)
You can use GetAsyncResult to retrieve the result of asynchronous method calls. Some method calls require some time to run, and might not be finished when the system sends the initial response. To obtain the status or result of the method call, use GetAsyncResult to poll the asyncHandle value returned by the method. GetAsyncResult returns the overall status of the operation (in progress, completed, or error) in a standard fashion, but the actual data returned for the operation depends on the original method call and the return data is documented with each method. :param asyncHandle: [required] A value that was returned from the original asynchronous method call. :type asyncHandle: int

Parameters **keepResult** (*bool*) – If true, GetAsyncResult does not remove the asynchronous result upon returning it, enabling future queries to that asyncHandle.

get_backup_target(*backup_target_id*)
GetBackupTarget enables you to return information about a specific backup target that you have created. :param backupTargetID: [required] The unique identifier assigned to the backup target. :type backupTargetID: int

get_bootstrap_config()
GetBootstrapConfig returns cluster and node information from the bootstrap configuration file. Use this API method on an individual node before it has been joined with a cluster. You can use the information this method returns in the cluster configuration interface when you create a cluster.

get_cluster_capacity()
You can use the GetClusterCapacity method to return the high-level capacity measurements for an entire cluster. You can use the fields returned from this method to calculate the efficiency rates that are displayed in the Element OS Web UI. You can use the following calculations in scripts to return the efficiency rates for thin provisioning, deduplication, compression, and overall efficiency.

get_cluster_config()
The GetClusterConfig API method enables you to return information about the cluster configuration this node uses to communicate with the cluster that it is a part of.

get_cluster_full_threshold()
You can use GetClusterFullThreshold to view the stages set for cluster fullness levels. This method returns all fullness metrics for the cluster. Note: When a cluster reaches the Error stage of block cluster fullness, the maximum IOPS on all volumes are reduced linearly to the volume's minimum IOPS as the cluster approaches the Critical stage. This helps prevent the cluster from reaching the Critical stage of block cluster fullness.

get_cluster_hardware_info(*type=None*)
You can use the GetClusterHardwareInfo method to retrieve the hardware status and information for all Fibre Channel nodes, iSCSI nodes and drives in the cluster. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information. :param type: Includes only a certain type of hardware information in the response. Possible values are: drives: List only drive information in the response. nodes: List only node information in the response. all: Include both drive and node information in the response. If this parameter is omitted, a type of "all" is assumed. :type type: str

get_cluster_info()
GetClusterInfo enables you to return configuration information about the cluster.

get_cluster_master_node_id()
GetClusterMasterNodeID enables you to retrieve the ID of the node that can perform cluster-wide administration tasks and holds the storage virtual IP address (SVIP) and management virtual IP address (MVIP).

get_cluster_state(*force*)
The GetClusterState API method enables you to indicate if a node is part of a cluster or not. The three states are: Available: Node has not been configured with a cluster name. Pending: Node is pending for a specific named cluster and can be added. Active: Node is an active member of a cluster and may not be added to another cluster. Note: This method is available only through the per-node API endpoint 5.0 or later. :param force: [required] To run this command, the force parameter must be set to true. :type force: bool

get_cluster_stats()
GetClusterStats enables you to retrieve high-level activity measurements for the cluster. Values returned are cumulative from the creation of the cluster.

get_cluster_version_info()
GetClusterVersionInfo enables you to retrieve information about the Element software version running on each node in the cluster. This method also returns information about nodes that are currently in the process of upgrading software.

get_complete_stats()
NetApp engineering uses the GetCompleteStats API method to troubleshoot new features. The data returned from GetCompleteStats is not documented, changes frequently, and is not guaranteed to be accurate. NetApp does not recommend using GetCompleteStats for collecting performance data or any other management integration with a SolidFire cluster.

get_config()
The GetConfig API method enables you to retrieve all configuration information for a node. This method includes the same information available in both the GetClusterConfig and GetNetworkConfig API methods. Note: This method is available only through the per-node API endpoint 5.0 or later.

get_current_cluster_admin()
GetCurrentClusterAdmin returns information for the current primary cluster administrator. The primary Cluster Admin was created when the cluster was created.

get_default_qos()
GetDefaultQoS enables you to retrieve the default QoS values for a newly created volume.

get_drive_config()
GetDriveConfig enables you to display drive information for expected slice and block drive counts as well as the number of slices and block drives that are currently connected to the node. Note: This method is available only through the per-node API endpoint 5.0 or later.

get_drive_hardware_info(*drive_id*)
GetDriveHardwareInfo returns all the hardware information for the given drive. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information. :param driveID: [required] DriveID for the drive information requested. You can get DriveIDs by using the ListDrives method. :type driveID: int

get_drive_stats(*drive_id*)
GetDriveStats returns high-level activity measurements for a single drive. Values are cumulative from the addition of the drive to the cluster. Some values are specific to block drives. You might not obtain statistical data for both block and metadata drives when you run this method. :param driveID: [required] Specifies the drive for which statistics are gathered. :type driveID: int

get_feature_status(*feature=None*)
GetFeatureStatus enables you to retrieve the status of a cluster feature. :param feature: Specifies the feature for which the status is returned. Valid value is: vvols: Retrieve status for the NetApp SolidFire VVols cluster feature. :type feature: str

get_hardware_config()
GetHardwareConfig enables you to display the hardware configuration information for a node. Note: This method is available only through the per-node API endpoint 5.0 or later.

get_hardware_info()
The GetHardwareInfo API method enables you to return hardware information and status for a single node. This generally includes details about manufacturers, vendors, versions, drives, and other associated hardware identification information.

get_ipmi_config(*chassis_type=None*)
GetIpmiConfig enables you to retrieve hardware sensor information from sensors that are in your node. :param chassisType: Displays information for each node chassis type. Valid values are: all: Returns sensor information for each chassis type. {chassis type}: Returns sensor information for a specified chassis type. :type chassisType: str

get_ipmi_info()
GetIpmiInfo enables you to display a detailed reporting of sensors (objects) for node fans, intake and exhaust temperatures, and power supplies that are monitored by the system.

get_ldap_configuration()
The GetLdapConfiguration method enables you to get the currently active LDAP configuration on the cluster.

get_limits()
GetLimits enables you to retrieve the limit values set by the API. These values might change between releases of Element OS, but do not change without an update to the system. Knowing the limit values set by the API can be useful when writing API scripts for user-facing tools. Note: The GetLimits method returns the limits for the current software version regardless of the API endpoint version used to pass the method.

get_login_banner()
You can use the GetLoginBanner method to get the currently active Terms of Use banner that users see when they log on to the web interface.

get_login_session_info()
GetLoginSessionInfo enables you to return the period of time a log in authentication session is valid for both log in shells and the TUI.

get_network_config()
The GetNetworkConfig API method enables you to display the network configuration information for a node. Note: This method is available only through the per-node API endpoint 5.0 or later.

get_node_hardware_info(*node_id*)
GetNodeHardwareInfo enables you to return all the hardware information and status for the node specified. This generally includes details about manufacturers, vendors, versions, and other associated hardware identification information. :param nodeID: [required] The ID of the node for which hardware information is being requested. Information about a Fibre Channel node is returned if a Fibre Channel node is specified. :type nodeID: int

get_node_sslcertificate()
You can use the GetNodeSSLCertificate method to retrieve the SSL certificate that is currently active on the cluster. You can use this method on both management and storage nodes.

get_node_stats(*node_id*)
GetNodeStats enables you to retrieve the high-level activity measurements for a single node. :param nodeID: [required] Specifies the node for which statistics are gathered. :type nodeID: int

get_ntp_info()
GetNtpInfo enables you to return the current network time protocol (NTP) configuration information.

get_nvram_info(*force=None*)
GetNvramInfo enables you to retrieve information from each node about the NVRAM card. :param force: Required parameter to successfully run on all nodes in the cluster. :type force: bool

get_origin()
GetOrigin enables you to retrieve the origination certificate for where the node was built. This method might return null if there is no origination certification.

get_pending_operation()
You can use GetPendingOperation to detect an operation on a node that is currently in progress. You can also use this method to report back when an operation has completed. Note: method is available only through the per-node API endpoint 5.0 or later.

get_qos_policy(*qos_policy_id*)
You can use the GetQoSPolicy method to get details about a specific QoSPolicy from the system. :param qosPolicyID: [required] The ID of the policy to be retrieved. :type qosPolicyID: int

get_raw_stats()
NetApp engineering uses the GetRawStats API method to troubleshoot new features. The data returned from GetRawStats is not documented, changes frequently, and is not guaranteed to be accurate. NetApp does not recommend using GetCompleteStats for collecting performance data or any other management integration with a SolidFire cluster.

get_remote_logging_hosts()
GetRemoteLoggingHosts enables you to retrieve the current list of log servers.

get_schedule(*schedule_id*)
You can use the GetSchedule method to retrieve information about a scheduled snapshot. You can see information about a specific schedule if there are many snapshot schedules in the system. You also retrieve information about more than one schedule with this method by specifying additional scheduleIDs in the

parameter. :param scheduleID: [required] Specifies the unique ID of the schedule or multiple schedules to display. :type scheduleID: int

get_snmp_acl()
GetSnmpACL enables you to return the current SNMP access permissions on the cluster nodes.

get_snmp_info()
GetSnmpInfo enables you to retrieve the current simple network management protocol (SNMP) configuration information. Note: GetSnmpInfo is available for Element OS 8 and prior releases. It is deprecated for versions later than Element OS 8. NetApp recommends that you migrate to the GetSnmpState and SetSnmpACL methods. See details in the Element API Reference Guide for their descriptions and usage.

get_snmp_state()
You can use GetSnmpState to return the current state of the SNMP feature.

get_snmp_trap_info()
You can use GetSnmpTrapInfo to return current SNMP trap configuration information.

get_sslcertificate()
You can use the GetSSLCertificate method to retrieve the SSL certificate that is currently active on the cluster.

get_storage_container_efficiency(*storage_container_id*)
GetStorageContainerEfficiency enables you to retrieve efficiency information about a virtual volume storage container. :param storageContainerID: [required] The ID of the storage container for which to retrieve efficiency information. :type storageContainerID: UUID

get_system_status()
GetSystemStatus enables you to return whether a reboot ir required or not.

get_virtual_volume_count()
Enables retrieval of the number of virtual volumes currently in the system.

get_volume_access_group_efficiency(*volume_access_group_id*)
GetVolumeAccessGroupEfficiency enables you to retrieve efficiency information about a volume access group. Only the volume access group you provide as the parameter in this API method is used to compute the capacity. :param volumeAccessGroupID: [required] The volume access group for which capacity is computed. :type volumeAccessGroupID: int

get_volume_access_group_lun_assignments(*volume_access_group_id*)
The GetVolumeAccessGroupLunAssignments method enables you to retrieve details on LUN mappings of a specified volume access group. :param volumeAccessGroupID: [required] The unique volume access group ID used to return information. :type volumeAccessGroupID: int

get_volume_count()
GetVolumeCount enables you to retrieve the number of volumes currently in the system.

get_volume_efficiency(*volume_id*)
GetVolumeEfficiency enables you to retrieve information about a volume. Only the volume you give as a parameter in this API method is used to compute the capacity. :param volumeID: [required] Specifies the volume for which capacity is computed. :type volumeID: int

get_volume_stats(*volume_id*)
GetVolumeStats enables you to retrieve high-level activity measurements for a single volume. Values are cumulative from the creation of the volume. :param volumeID: [required] Specifies the volume for which statistics are gathered. :type volumeID: int

invoke_sfapi(*method*, *parameters=None*)
This will invoke any API method supported by the SolidFire API for the version and port the connection is using. Returns a nested hashtable of key/value pairs that contain the result of the invoked method. :param method: [required] The name of the method to invoke. This is case sensitive. :type method: str

Parameters **parameters** (*str*) – An object, normally a dictionary or hashtable of the key/value pairs, to be passed as the params for the method being invoked.

list_accounts(*start_account_id=None*, *limit=None*, *include_storage_containers=None*)
ListAccounts returns the entire list of accounts, with optional paging support. :param startAccountID: Starting AccountID to return. If no account exists with this AccountID, the next account by AccountID order is used as the start of the list. To page through the list, pass the AccountID of the last account in the previous response + 1. :type startAccountID: int

Parameters

- **limit** (*int*) – Maximum number of AccountInfo objects to return.
- **includeStorageContainers** (*bool*) – Includes storage containers in the response by default. To exclude storage containers, set to false.

list_active_nodes()
ListActiveNodes returns the list of currently active nodes that are in the cluster.

list_active_paired_volumes(*start_volume_id=None*, *limit=None*)
ListActivePairedVolumes enables you to list all the active volumes paired with a volume. This method returns information about volumes with active and pending pairings. :param startVolumeID: The beginning of the range of active paired volumes to return. :type startVolumeID: int

Parameters **limit** (*int*) – Maximum number of active paired volumes to return.

list_active_volumes(*start_volume_id=None*, *limit=None*, *include_virtual_volumes=None*)
ListActiveVolumes enables you to return the list of active volumes currently in the system. The list of volumes is returned sorted in VolumeID order and can be returned in multiple parts (pages). :param startVolumeID: Starting VolumeID to return. If no volume exists with this VolumeID, the next volume by VolumeID order is used as the start of the list. To page through the list, pass the VolumeID of the last volume in the previous response + 1. :type startVolumeID: int

Parameters

- **limit** (*int*) – Maximum number of Volume Info objects to return. A value of 0 (zero) returns all volumes (unlimited).
- **includeVirtualVolumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

list_all_nodes()
ListAllNodes enables you to retrieve a list of active and pending nodes in the cluster.

list_async_results(*async_result_types=None*)
You can use ListAsyncResults to list the results of all currently running and completed asynchronous methods on the system. Querying asynchronous results with ListAsyncResults does not cause completed asyncHandles to expire; you can use GetAsyncResult to query any of the asyncHandles returned by ListAsyncResults. :param asyncResultTypes: An optional list of types of results. You can use this list to restrict the results to only these types of operations. Possible values are: BulkVolume: Copy operations between volumes, such as backups or restores. Clone: Volume cloning operations. DriveRemoval: Operations involving the system copying data from a drive in preparation to remove it from the cluster. RtfiPendingNode: Operations involving the system installing compatible software on a node before adding it to the cluster :type asyncResultTypes: str

list_backup_targets()
You can use ListBackupTargets to retrieve information about all backup targets that have been created.

list_bulk_volume_jobs()
ListBulkVolumeJobs enables you to retrieve information about each bulk volume read or write operation that is occurring in the system.

list_cluster_admins()
ListClusterAdmins returns the list of all cluster administrators for the cluster. There can be several cluster administrator accounts with different levels of permissions. There can be only one primary cluster administrator in the system. The primary Cluster Admin is the administrator that was created when the cluster was created. You can also create LDAP administrators when setting up an LDAP system on the cluster.

list_cluster_faults(*best_practices=None*, *fault_types=None*)
ListClusterFaults enables you to retrieve information about any faults detected on the cluster. With this method, you can retrieve both current faults as well as faults that have been resolved. The system caches faults every 30 seconds. :param bestPractices: Specifies whether to include faults triggered by suboptimal system configuration. Possible values are: true false :type bestPractices: bool

Parameters **faultTypes** (*str*) – Determines the types of faults returned. Possible values are: current: List active, unresolved faults. resolved: List faults that were previously detected and resolved. all: (Default) List both current and resolved faults. You can see the fault status in the resolved field of the Cluster Fault object.

list_cluster_pairs()
You can use the ListClusterPairs method to list all the clusters that a cluster is paired with. This method returns information about active and pending cluster pairings, such as statistics about the current pairing as well as the connectivity and latency (in milliseconds) of the cluster pairing.

list_deleted_volumes(*include_virtual_volumes=None*)
ListDeletedVolumes enables you to retrieve the list of volumes that have been marked for deletion and purged from the system. :param includeVirtualVolumes: Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false. :type includeVirtualVolumes: bool

list_drive_hardware(*force*)
ListDriveHardware returns all the drives connected to a node. Use this method on individual nodes to return drive hardware information or use this method on the cluster master node MVIP to see information for all the drives on all nodes. Note: The "securitySupported": true line of the method response does not imply that the drives are capable of encryption; only that the security status can be queried. If you have a node type with a model number ending in "-NE", commands to enable security features on these drives will fail. See the EnableEncryptionAtRest method for more information. :param force: [required] To run this command, the force parameter must be set to true. :type force: bool

list_drive_stats(*drives=None*)
ListDriveStats enables you to retrieve high-level activity measurements for multiple drives in the cluster. By default, this method returns statistics for all drives in the cluster, and these measurements are cumulative from the addition of the drive to the cluster. Some values this method returns are specific to block drives, and some are specific to metadata drives. :param drives: Optional list of DriveIDs for which to return drive statistics. If you omit this parameter, measurements for all drives are returned. :type drives: int

list_drives()
ListDrives enables you to retrieve the list of the drives that exist in the cluster's active nodes. This method returns drives that have been added as volume metadata or block drives as well as drives that have not been added and are available.

list_events(*max_events=None*, *start_event_id=None*, *end_event_id=None*, *event_type=None*)
ListEvents returns events detected on the cluster, sorted from oldest to newest. :param maxEvents: Specifies the maximum number of events to return. :type maxEvents: int

Parameters

- **startEventID** (*int*) – Identifies the beginning of a range of events to return.
- **endEventID** (*int*) – Identifies the end of a range of events to return.
- **eventType** (*str*) –

list_fibre_channel_port_info()
ListFibreChannelPortInfo enables you to retrieve information about the Fibre Channel ports on a node. The API method is intended for use on individual nodes; userid and password authentication is required for access to individual Fibre Channel nodes.

list_fibre_channel_sessions()
ListFibreChannelSessions enables you to retrieve information about the active Fibre Channel sessions on a cluster.

list_group_snapshots(*volumes=None*, *group_snapshot_id=None*)
ListGroupSnapshots enables you to get information about all group snapshots that have been created. :param volumes: An array of unique volume IDs to query. If you do not specify this parameter, all group snapshots on the cluster are included. :type volumes: int

Parameters **groupSnapshotID** (*int*) – Retrieves information for a specific group snapshot ID.

list_initiators(*start_initiator_id=None*, *limit=None*, *initiators=None*)
ListInitiators enables you to list initiator IQNs or World Wide Port Names (WWPNs). :param startInitiatorID: The initiator ID at which to begin the listing. You can supply this parameter or the "initiators" parameter, but not both. :type startInitiatorID: int

Parameters

- **limit** (*int*) – The maximum number of initiator objects to return.
- **initiators** (*int*) – A list of initiator IDs to retrieve. You can provide a value for this parameter or the "startInitiatorID" parameter, but not both.

list_iscsisessions()
You can use ListISCSISessions to return iSCSI information for volumes in the cluster.

list_network_interfaces()
ListNetworkInterfaces enables you to retrieve information about each network interface on a node. The API method is intended for use on individual nodes; userid and password authentication is required for access to individual nodes.

list_node_fibre_channel_port_info()
The ListNodeFibreChannelPortInfo API method enables you to retrieve information about the Fibre Channel ports on a node. The API method is intended for use on individual nodes; userid and password authentication is required for access to individual Fibre Channel nodes.

list_node_stats()
ListNodeStats enables you to view the high-level activity measurements for all nodes in a cluster.

list_pending_active_nodes()
ListPendingActiveNodes returns the list of nodes in the cluster that are currently in the PendingActive state, between the pending and active states. These are nodes that are currently being returned to the factory image.

list_pending_nodes()
ListPendingNodes returns a list of the currently pending nodes in the system. Pending nodes are nodes that are running and configured to join the cluster, but have not yet been added via the AddNodes API method.

list_protocol_endpoints(*protocol_endpoint_ids=None*)
ListProtocolEndpoints enables you to retrieve information about all protocol endpoints in the cluster. Protocol endpoints govern access to their associated virtual volume storage containers. :param protocolEndpointIDs: A list of protocol endpoint IDs for which to retrieve information. If you omit this parameter, the method returns information about all protocol endpoints. :type protocolEndpointIDs: UUID

list_qos_policies()
You can use the ListQoSPolicies method to list all the settings of all QoS policies on the system.

list_schedules()
ListSchedule enables you to retrieve information about all scheduled snapshots that have been created.

list_services()
You can use ListServices to return the services information for nodes, drives, current software, and other services that are running on the cluster.

list_snapshots(*volume_id=None*, *snapshot_id=None*)
ListSnapshots enables you to return the attributes of each snapshot taken on the volume. Information about snapshots that reside on the target cluster is displayed on the source cluster when this method is called from the source cluster. :param volumeID: Retrieves snapshots for a volume. If volumeID is not provided, all snapshots for all volumes are returned. :type volumeID: int

Parameters **snapshotID** (*int*) – Retrieves information for a specific snapshot ID.

list_storage_containers(*storage_container_ids=None*)
ListStorageContainers enables you to retrieve information about all virtual volume storage containers known to the system. :param storageContainerIDs: A list of storage container IDs for which to retrieve information. If you omit this parameter, the method returns information about all storage containers in the system. :type storageContainerIDs: UUID

list_sync_jobs()
ListSyncJobs enables you to return information about synchronization jobs that are running on a SolidFire cluster. The type of synchronization jobs that are returned with this method are slice, clone, and remote.

list_tests()
You can use the ListTests API method to return the tests that are available to run on a node. Note: This method is available only through the per-node API endpoint 5.0 or later.

list_utilities()
You can use the ListUtilities API method to return the operations that are available to run on a node. Note: This method is available only through the per-node API endpoint 5.0 or later.

list_virtual_networks(*virtual_network_id=None*, *virtual_network_tag=None*, *virtual_network_ids=None*, *virtual_network_tags=None*)
ListVirtualNetworks enables you to list all configured virtual networks for the cluster. You can use this method to verify the virtual network settings in the cluster. There are no required parameters for this method. However, to filter the results, you can pass one or more VirtualNetworkID or VirtualNetworkTag values. :param virtualNetworkID: Network ID to filter the list for a single virtual network. :type virtualNetworkID: int

Parameters

- **virtualNetworkTag** (*int*) – Network tag to filter the list for a single virtual network.
- **virtualNetworkIDs** (*int*) – Network IDs to include in the list.
- **virtualNetworkTags** (*int*) – Network tag to include in the list.

list_virtual_volume_bindings(*virtual_volume_binding_ids=None*)
ListVirtualVolumeBindings returns a list of all virtual volumes in the cluster that are bound to protocol endpoints. :param virtualVolumeBindingIDs: A list of virtual volume binding IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume bindings. :type virtualVolumeBindingIDs: int

list_virtual_volume_hosts(*virtual_volume_host_ids=None*)
ListVirtualVolumeHosts returns a list of all virtual volume hosts known to the cluster. A virtual volume

host is a VMware ESX host that has initiated a session with the VASA API provider. :param virtualVolumeHostIDs: A list of virtual volume host IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume hosts. :type virtualVolumeHostIDs: UUID

list_virtual_volume_tasks (*virtual_volume_task_ids=None*)
ListVirtualVolumeTasks returns a list of virtual volume tasks in the system. :param virtualVolumeTaskIDs: A list of virtual volume task IDs for which to retrieve information. If you omit this parameter, the method returns information about all virtual volume tasks. :type virtualVolumeTaskIDs: UUID

list_virtual_volumes (*details=None*, *limit=None*, *recursive=None*, *start_virtual_volume_id=None*, *virtual_volume_ids=None*)
ListVirtualVolumes enables you to list the virtual volumes currently in the system. You can use this method to list all virtual volumes, or only list a subset. :param details: Specifies the level of detail about each virtual volume that is returned. Possible values are: true: Include more details about each virtual volume in the response. false: Include the standard level of detail about each virtual volume in the response. :type details: bool

Parameters

- **limit** (*int*) – The maximum number of virtual volumes to list.
- **recursive** (*bool*) – Specifies whether to include information about the children of each virtual volume in the response. Possible values are: true: Include information about the children of each virtual volume in the response. false: Do not include information about the children of each virtual volume in the response.
- **startVirtualVolumeID** (*UUID*) – The ID of the virtual volume at which to begin the list.
- **virtualVolumeIDs** (*UUID*) – A list of virtual volume IDs for which to retrieve information. If you specify this parameter, the method returns information about only these virtual volumes.

list_volume_access_groups (*start_volume_access_group_id=None*, *limit=None*, *volume_access_groups=None*)
ListVolumeAccessGroups enables you to return information about the volume access groups that are currently in the system. :param startVolumeAccessGroupID: The volume access group ID at which to begin the listing. If unspecified, there is no lower limit (implicitly 0). :type startVolumeAccessGroupID: int

Parameters

- **limit** (*int*) – The maximum number of results to return. This can be useful for paging.
- **volumeAccessGroups** (*int*) – The list of ids of the volume access groups you wish to list

list_volume_stats (*volume_ids=None*)
ListVolumeStats returns high-level activity measurements for a single volume, list of volumes, or all volumes (if you omit the volumeIDs parameter). Measurement values are cumulative from the creation of the volume. :param volumeIDs: A list of volume IDs of volumes from which to retrieve activity information. :type volumeIDs: int

list_volume_stats_by_account (*accounts=None*, *include_virtual_volumes=None*)
ListVolumeStatsByAccount returns high-level activity measurements for every account. Values are summed from all the volumes owned by the account. :param accounts: One or more account ids by which to filter the result. :type accounts: int

Parameters **includeVirtualVolumes** (*bool*) – Includes virtual volumes in the response by default. To exclude virtual volumes, set to false.

list_volume_stats_by_virtual_volume (*virtual_volume_ids=None*)
ListVolumeStatsByVirtualVolume enables you to list volume statistics for any volumes in the system that

are associated with virtual volumes. Statistics are cumulative from the creation of the volume. :param virtualVolumeIDs: A list of one or more virtual volume IDs for which to retrieve information. If you specify this parameter, the method returns information about only these virtual volumes. :type virtualVolumeIDs: UUID

list_volume_stats_by_volume(*include_virtual_volumes=None*)
ListVolumeStatsByVolume returns high-level activity measurements for every volume, by volume. Values are cumulative from the creation of the volume. :param includeVirtualVolumes: Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false. :type includeVirtualVolumes: bool

list_volume_stats_by_volume_access_group(*volume_access_groups=None*, *include_virtual_volumes=None*)
ListVolumeStatsByVolumeAccessGroup enables you to get total activity measurements for all of the volumes that are a member of the specified volume access group(s). :param volumeAccessGroups: An array of VolumeAccessGroupIDs for which volume activity is returned. If omitted, statistics for all volume access groups are returned. :type volumeAccessGroups: int

Parameters **includeVirtualVolumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

list_volumes(*start_volume_id=None*, *limit=None*, *volume_status=None*, *accounts=None*, *is_paired=None*, *volume_ids=None*, *volume_name=None*, *include_virtual_volumes=None*)
The ListVolumes method enables you to retrieve a list of volumes that are in a cluster. You can specify the volumes you want to return in the list by using the available parameters. :param startVolumeID: Only volumes with an ID greater than or equal to this value are returned. Mutually exclusive with the volumeIDs parameter. :type startVolumeID: int

Parameters

- **limit** (*int*) – Specifies the maximum number of volume results that are returned. Mutually exclusive with the volumeIDs parameter.
- **volumeStatus** (*str*) – Only volumes with a status equal to the status value are returned. Possible values are: creating snapshotting active deleted
- **accounts** (*int*) – Returns only the volumes owned by the accounts you specify here. Mutually exclusive with the volumeIDs parameter.
- **isPaired** (*bool*) – Returns volumes that are paired or not paired. Possible values are: true: Returns all paired volumes. false: Returns all volumes that are not paired.
- **volumeIDs** (*int*) – A list of volume IDs. If you supply this parameter, other parameters operate only on this set of volumes. Mutually exclusive with the accounts, startVolumeID, and limit parameters.
- **volumeName** (*str*) – Only volume object information matching the volume name is returned.
- **includeVirtualVolumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

list_volumes_for_account(*account_id*, *start_volume_id=None*, *limit=None*, *include_virtual_volumes=None*)
ListVolumesForAccount returns the list of active and (pending) deleted volumes for an account. :param accountID: [required] Returns all volumes owned by this AccountID. :type accountID: int

Parameters

- **startVolumeID** (*int*) – The ID of the first volume to list. This can be useful for paging results. By default, this starts at the lowest VolumeID.

- **limit** (*int*) – The maximum number of volumes to return from the API.
- **includeVirtualVolumes** (*bool*) – Specifies that virtual volumes are included in the response by default. To exclude virtual volumes, set to false.

modify_account(*account_id*, *username=None*, *status=None*, *initiator_secret=None*, *target_secret=None*, *attributes=None*)

ModifyAccount enables you to modify an existing account. When you lock an account, any existing connections from that account are immediately terminated. When you change an account's CHAP settings, any existing connections remain active, and the new CHAP settings are used on subsequent connections or reconnections. To clear an account's attributes, specify {} for the attributes parameter. :param accountID: [required] Specifies the AccountID for the account to be modified. :type accountID: int

Parameters

- **username** (*str*) – Specifies the username associated with the account. (Might be 1 to 64 characters in length).
- **status** (*str*) – Sets the status for the account. Possible values are: active: The account is active and connections are allowed. locked: The account is locked and connections are refused.
- **initiatorSecret** (CHAPSecret) – Specifies the CHAP secret to use for the initiator. This secret must be 12-16 characters in length and should be impenetrable. The initiator CHAP secret must be unique and cannot be the same as the target CHAP secret.
- **targetSecret** (CHAPSecret) – Specifies the CHAP secret to use for the target (mutual CHAP authentication). This secret must be 12-16 characters in length and should be impenetrable. The target CHAP secret must be unique and cannot be the same as the initiator CHAP secret.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

modify_backup_target(*backup_target_id*, *name=None*, *attributes=None*)

ModifyBackupTarget enables you to change attributes of a backup target. :param backupTargetID: [required] The unique target ID for the target to modify. :type backupTargetID: int

Parameters

- **name** (*str*) – The new name for the backup target.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

modify_cluster_admin(*cluster_admin_id*, *password=None*, *access=None*, *attributes=None*)

You can use ModifyClusterAdmin to change the settings for a cluster admin or LDAP cluster admin. You cannot change access for the administrator cluster admin account. :param clusterAdminID: [required] ClusterAdminID for the cluster admin or LDAP cluster admin to modify. :type clusterAdminID: int

Parameters

- **password** (*str*) – Password used to authenticate this cluster admin.
- **access** (*str*) – Controls which methods this cluster admin can use. For more details, see Access Control in the Element API Reference Guide.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

modify_cluster_full_threshold(*stage2_aware_threshold=None*, *stage3_block_threshold_percent=None*, *max_metadata_over_provision_factor=None*)

You can use ModifyClusterFullThreshold to change the level at which the system generates an event when the storage cluster approaches a certain capacity utilization. You can use the threshold setting to indicate the acceptable amount of utilized block storage before the system generates a warning. For example, if

you want to be alerted when the system reaches 3% below the "Error" level block storage utilization, enter a value of "3" for the stage3BlockThresholdPercent parameter. If this level is reached, the system sends an alert to the Event Log in the Cluster Management Console. :param stage2AwareThreshold: The number of nodes of capacity remaining in the cluster before the system triggers a capacity notification. :type stage2AwareThreshold: int

Parameters

- **stage3BlockThresholdPercent** (*int*) – The percentage of block storage utilization below the "Error" threshold that causes the system to trigger a cluster "Warning" alert.
- **maxMetadataOverProvisionFactor** (*int*) – A value representative of the number of times metadata space can be overprovisioned relative to the amount of space available. For example, if there was enough metadata space to store 100 TiB of volumes and this number was set to 5, then 500 TiB worth of volumes can be created.

modify_group_snapshot(*group_snapshot_id*, *expiration_time=None*, *enable_remote_replication=None*, *snap_mirror_label=None*)

ModifyGroupSnapshot enables you to change the attributes of a group of snapshots. You can also use this method to enable snapshots created on the Read/Write (source) volume to be remotely replicated to a target SolidFire storage system. :param groupSnapshotID: [required] Specifies the ID of the group of snapshots. :type groupSnapshotID: int

Parameters

- **expirationTime** (*str*) – Sets the time when the snapshot should be removed. If unspecified, the current time is used.
- **enableRemoteReplication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **snapMirrorLabel** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

modify_initiators(*initiators*)

ModifyInitiators enables you to change the attributes of one or more existing initiators. You cannot change the name of an existing initiator. If you need to change the name of an initiator, delete it first with DeleteInitiators and create a new one with CreateInitiators. If ModifyInitiators fails to change one of the initiators provided in the parameter, the method returns an error and does not modify any initiators (no partial completion is possible). :param initiators: [required] A list of objects containing characteristics of each initiator to modify. Values are: initiatorID: (Required) The ID of the initiator to modify. (Integer) alias: (Optional) A new friendly name to assign to the initiator. (String) attributes: (Optional) A new set of JSON attributes to assign to the initiator. (JSON Object) volumeAccessGroupID: (Optional) The ID of the volume access group into to which the initiator should be added. If the initiator was previously in a different volume access group, it is removed from the old volume access group. If this key is present but null, the initiator is removed from its current volume access group, but not placed in any new volume access group. (Integer) :type initiators: ModifyInitiator

modify_qos_policy(*qos_policy_id*, *name=None*, *qos=None*)

You can use the ModifyQoSPolicy method to modify an existing QoSPolicy on the system. :param qosPolicyID: [required] The ID of the policy to be modified. :type qosPolicyID: int

Parameters

- **name** (*str*) – If supplied, the name of the QoS Policy (e.g. gold, platinum, silver) is changed to this value.
- **qos** (QoS) – If supplied, the QoS settings for this policy are changed to these sttings. You can supply partial QoS values and only change some of the QoS settings.

modify_schedule(*schedule*)
ModifySchedule enables you to change the intervals at which a scheduled snapshot occurs. This allows for adjustment to the snapshot frequency and retention. :param schedule: [required] The "Schedule" object will be used to modify an existing schedule. The ScheduleID property is required. Frequency property must be of type that inherits from Frequency. Valid types are: DaysOfMonthFrequency DaysOrWeekFrequency TimeIntervalFrequency :type schedule: Schedule

modify_snapshot(*snapshot_id*, *expiration_time=None*, *enable_remote_replication=None*, *snap_mirror_label=None*)
ModifySnapshot enables you to change the attributes currently assigned to a snapshot. You can use this method to enable snapshots created on the Read/Write (source) volume to be remotely replicated to a target SolidFire storage system. :param snapshotID: [required] Specifies the ID of the snapshot. :type snapshotID: int

Parameters

- **expirationTime** (*str*) – Sets the time when the snapshot should be removed.
- **enableRemoteReplication** (*bool*) – Replicates the snapshot created to a remote cluster. Possible values are: true: The snapshot is replicated to remote storage. false: Default. The snapshot is not replicated.
- **snapMirrorLabel** (*str*) – Label used by SnapMirror software to specify snapshot retention policy on SnapMirror endpoint.

modify_storage_container(*storage_container_id*, *initiator_secret=None*, *target_secret=None*)
ModifyStorageContainer enables you to make changes to an existing virtual volume storage container. :param storageContainerID: [required] The unique ID of the virtual volume storage container to modify. :type storageContainerID: UUID

Parameters

- **initiatorSecret** (*str*) – The new secret for CHAP authentication for the initiator.
- **targetSecret** (*str*) – The new secret for CHAP authentication for the target.

modify_virtual_network(*virtual_network_id=None*, *virtual_network_tag=None*, *name=None*, *address_blocks=None*, *netmask=None*, *svip=None*, *gateway=None*, *namespace=None*, *attributes=None*)
You can use ModifyVirtualNetwork to change the attributes of an existing virtual network. This method enables you to add or remove address blocks, change the netmask, or modify the name or description of the virtual network. You can also use it to enable or disable namespaces, as well as add or remove a gateway if namespaces are enabled on the virtual network. Note: This method requires either the VirtualNetworkID or the VirtualNetworkTag as a parameter, but not both. Caution: Enabling or disabling the Routable Storage VLANs functionality for an existing virtual network by changing the "namespace" parameter disrupts any traffic handled by the virtual network. NetApp strongly recommends changing the "namespace" parameter only during a scheduled maintenance window. :param virtualNetworkID: The unique identifier of the virtual network to modify. This is the virtual network ID assigned by the cluster. Note: This parameter is optional but either virtualNetworkID or virtualNetworkTag must be specified with this API method. :type virtualNetworkID: int

Parameters

- **virtualNetworkTag** (*int*) – The network tag that identifies the virtual network to modify. Note: This parameter is optional but either virtualNetworkID or virtualNetworkTag must be specified with this API method.
- **name** (*str*) – The new name for the virtual network.
- **addressBlocks** (AddressBlockParams) – The new addressBlock to set for this virtual network. This might contain new address blocks to add to the existing object or

omit unused address blocks that need to be removed. Alternatively, you can extend or reduce the size of existing address blocks. You can only increase the size of the starting addressBlocks for a virtual network object; you can never decrease it. Attributes for this parameter are: start: The start of the IP address range. (String) size: The number of IP addresses to include in the block. (Integer)

- **netmask** (*str*) – New network mask for this virtual network.
- **svip** (*str*) – The storage virtual IP address for this virtual network. The svip for a virtual network cannot be changed. You must create a new virtual network to use a different svip address.
- **gateway** (*str*) – The IP address of a gateway of the virtual network. This parameter is only valid if the "namespace" parameter is set to true.
- **namespace** (*bool*) – When set to true, enables Routable Storage VLANs functionality by recreating the virtual network and configuring a namespace to contain it. When set to false, disables the VRF functionality for the virtual network. Changing this value disrupts traffic running through this virtual network.
- **attributes** (*dict*) – A new list of name-value pairs in JSON object format.

modify_volume(*volume_id*, *account_id=None*, *access=None*, *qos=None*, *total_size=None*, *attributes=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*)

ModifyVolume enables you to modify settings on an existing volume. You can make modifications to one volume at a time and changes take place immediately. If you do not specify QoS values when you modify a volume, they remain the same as before the modification. You can retrieve default QoS values for a newly created volume by running the GetDefaultQoS method. When you need to increase the size of a volume that is being replicated, do so in the following order to prevent replication errors: 1. Increase the size of the "Replication Target" volume. 2. Increase the size of the source or "Read / Write" volume. NetApp recommends that both the target and source volumes are the same size. Note: If you change the "access" status to locked or target, all existing iSCSI connections are terminated. :param volumeID: [required] VolumeID for the volume to be modified. :type volumeID: int

Parameters

- **accountID** (*int*) – AccountID to which the volume is reassigned. If unspecified, the previous account name is used.
- **access** (*str*) – Specifies the access allowed for the volume. Possible values are: readOnly: Only read operations are allowed. readWrite: Reads and writes are allowed. locked: No reads or writes are allowed. If not specified, the access value does not change. replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked. If a value is not specified, the access value does not change.
- **qos** (QoS) – New QoS settings for this volume. If not specified, the QoS settings are not changed.
- **totalSize** (*int*) – New size of the volume in bytes. 1000000000 is equal to 1GB. Size is rounded up to the nearest 1MB. This parameter can only be used to increase the size of a volume.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.
- **associateWithQoSPolicy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.

- **qosPolicyID** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. The volume will not maintain any association with the policy; this is an alternate way to apply QoS settings to the volume. This parameter and the qos parameter cannot be specified at the same time.

modify_volume_access_group(*volume_access_group_id*, *virtual_network_id=None*, *virtual_network_tags=None*, *name=None*, *initiators=None*, *volumes=None*, *delete_orphan_initiators=None*, *attributes=None*)

You can use ModifyVolumeAccessGroup to update initiators and add or remove volumes from a volume access group. If a specified initiator or volume is a duplicate of what currently exists, the volume access group is left as-is. If you do not specify a value for volumes or initiators, the current list of initiators and volumes is not changed. :param volumeAccessGroupID: [required] The ID of the volume access group to modify. :type volumeAccessGroupID: int

Parameters

- **virtualNetworkID** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **virtualNetworkTags** (*int*) – The ID of the SolidFire virtual network to associate the volume access group with.
- **name** (*str*) – The new name for this volume access group. Not required to be unique, but recommended.
- **initiators** (*str*) – List of initiators to include in the volume access group. If unspecified, the access group's configured initiators are not modified.
- **volumes** (*int*) – List of volumes to initially include in the volume access group. If unspecified, the access group's volumes are not modified.
- **deleteOrphanInitiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

modify_volume_access_group_lun_assignments(*volume_access_group_id*, *lun_assignments*)

The ModifyVolumeAccessGroupLunAssignments method enables you to define custom LUN assignments for specific volumes. This method changes only LUN values set on the lunAssignments parameter in the volume access group. All other LUN assignments remain unchanged. LUN assignment values must be unique for volumes in a volume access group. You cannot define duplicate LUN values within a volume access group. However, you can use the same LUN values again in different volume access groups. Note: Correct LUN values are 0 through 16383. The system generates an exception if you pass a LUN value outside of this range. None of the specified LUN assignments are modified if there is an exception. Caution: If you change a LUN assignment for a volume with active I/O, the I/O can be disrupted. You might need to change the server configuration before changing volume LUN assignments. :param volumeAccessGroupID: [required] The ID of the volume access group for which the LUN assignments will be modified. :type volumeAccessGroupID: int

Parameters **lunAssignments** (LunAssignment) – [required] The volume IDs with new assigned LUN values.

modify_volume_pair(*volume_id*, *paused_manual=None*, *mode=None*, *pause_limit=None*)

ModifyVolumePair enables you to pause or restart replication between a pair of volumes. :param volumeID: [required] The ID of the volume to be modified. :type volumeID: int

Parameters

- **pausedManual** (*bool*) – Specifies whether to pause or restart volume replication process. Valid values are: true: Pauses volume replication false: Restarts volume replication
- **mode** (*str*) – Specifies the volume replication mode. Possible values are: Async: Writes are acknowledged when they complete locally. The cluster does not wait for writes to be replicated to the target cluster. Sync: The source acknowledges the write when the data is stored locally and on the remote cluster. SnapshotsOnly: Only snapshots created on the source cluster are replicated. Active writes from the source volume are not replicated.
- **pauseLimit** (*int*) – Internal use only.

modify_volumes(*volume_ids*, *account_id=None*, *access=None*, *qos=None*, *total_size=None*, *associate_with_qos_policy=None*, *qos_policy_id=None*, *attributes=None*)

ModifyVolumes allows you to configure up to 500 existing volumes at one time. Changes take place immediately. If ModifyVolumes fails to modify any of the specified volumes, none of the specified volumes are changed. If you do not specify QoS values when you modify volumes, the QoS values for each volume remain unchanged. You can retrieve default QoS values for a newly created volume by running the GetDefaultQoS method. When you need to increase the size of volumes that are being replicated, do so in the following order to prevent replication errors:

> Increase the size of the "Replication Target" volume. Increase the size of the source or "Read / Write" volume.

Recommend that both the target and source volumes be the same size. NOTE: If you change access status to locked or replicationTarget all existing iSCSI connections are terminated. :param volumeIDs: [required] A list of volumeIDs for the volumes to be modified. :type volumeIDs: int

Parameters

- **accountID** (*int*) – AccountID to which the volume is reassigned. If none is specified, the previous account name is used.
- **access** (*str*) – Access allowed for the volume. Possible values:readOnly: Only read operations are allowed.readWrite: Reads and writes are allowed.locked: No reads or writes are allowed.If not specified, the access value does not change.replicationTarget: Identify a volume as the target volume for a paired set of volumes. If the volume is not paired, the access status is locked.If a value is not specified, the access value does not change.
- **qos** (QoS) – New quality of service settings for this volume.If not specified, the QoS settings are not changed.
- **totalSize** (*int*) – New size of the volume in bytes. 1000000000 is equal to 1GB. Size is rounded up to the nearest 1MB in size. This parameter can only be used to increase the size of a volume.
- **associateWithQoSPolicy** (*bool*) – Associate the volume with the specified QoS policy. Possible values: true: Associate the volume with the QoS policy specified in the QoSPolicyID parameter. false: Do not assosciate the volume with the QoS policy specified in the QoSPolicyID parameter. When false, any existing policy association is removed regardless of whether you specify a QoS policy in the QoSPolicyID parameter.
- **qosPolicyID** (*int*) – The ID for the policy whose QoS settings should be applied to the specified volumes. This parameter is mutually exclusive with the qos parameter.
- **attributes** (*dict*) – List of name/value pairs in JSON object format.

purge_deleted_volume(*volume_id*)

PurgeDeletedVolume immediately and permanently purges a volume that has been deleted. You must delete a volume using DeleteVolume before it can be purged. Volumes are purged automatically after a period of time, so usage of this method is not typically required. :param volumeID: [required] The ID of the volume to be purged. :type volumeID: int

purge_deleted_volumes(*volume_ids=None*, *account_ids=None*, *volume_access_group_ids=None*)

PurgeDeletedVolumes immediately and permanently purges volumes that have been deleted. You can use this method to purge up to 500 volumes at one time. You must delete volumes using DeleteVolumes before they can be purged. Volumes are purged by the system automatically after a period of time, so usage of this method is not typically required. :param volumeIDs: A list of volumeIDs of volumes to be purged from the system. :type volumeIDs: int

Parameters

- **accountIDs** (*int*) – A list of accountIDs. All of the volumes from all of the specified accounts are purged from the system.
- **volumeAccessGroupIDs** (*int*) – A list of volumeAccessGroupIDs. All of the volumes from all of the specified Volume Access Groups are purged from the system.

remove_account(*account_id*)

RemoveAccount enables you to remove an existing account. You must delete and purge all volumes associated with the account using DeleteVolume before you can remove the account. If volumes on the account are still pending deletion, you cannot use RemoveAccount to remove the account. :param accountID: [required] Specifies the AccountID for the account to be removed. :type accountID: int

remove_backup_target(*backup_target_id*)

RemoveBackupTarget allows you to delete backup targets. :param backupTargetID: [required] The unique target ID of the target to remove. :type backupTargetID: int

remove_cluster_admin(*cluster_admin_id*)

You can use RemoveClusterAdmin to remove a Cluster Admin. You cannot remove the administrator cluster admin account. :param clusterAdminID: [required] ClusterAdminID for the cluster admin to remove. :type clusterAdminID: int

remove_cluster_pair(*cluster_pair_id*)

You can use the RemoveClusterPair method to close the open connections between two paired clusters. Note: Before you remove a cluster pair, you must first remove all volume pairing to the clusters with the "RemoveVolumePair" API method. :param clusterPairID: [required] Unique identifier used to pair two clusters. :type clusterPairID: int

remove_drives(*drives*, *force_during_upgrade=None*)

You can use RemoveDrives to proactively remove drives that are part of the cluster. You might want to use this method when reducing cluster capacity or preparing to replace drives nearing the end of their service life. Any data on the drives is removed and migrated to other drives in the cluster before the drive is removed from the cluster. This is an asynchronous method. Depending on the total capacity of the drives being removed, it might take several minutes to migrate all of the data. Use the GetAsyncResult method to check the status of the remove operation. When removing multiple drives, use a single RemoveDrives method call rather than multiple individual methods with a single drive each. This reduces the amount of data balancing that must occur to even stabilize the storage load on the cluster. You can also remove drives with a "failed" status using RemoveDrives. When you remove a drive with a "failed" status it is not returned to an "available" or active status. The drive is unavailable for use in the cluster. Use the ListDrives method to obtain the driveIDs for the drives you want to remove. :param drives: [required] List of driveIDs to remove from the cluster. :type drives: int

Parameters **forceDuringUpgrade** (*bool*) – If you want to remove a drive during upgrade, this must be set to true.

remove_initiators_from_volume_access_group(*volume_access_group_id*, *initiators*, *delete_orphan_initiators=None*)

RemoveInitiatorsFromVolumeAccessGroup enables you to remove initiators from a specified volume access group. :param volumeAccessGroupID: [required] The ID of the volume access group from which the initiators are removed. :type volumeAccessGroupID: int

Parameters

- **initiators** (*str*) – [required] The list of initiators to remove from the volume access group.
- **deleteOrphanInitiators** (*bool*) – true: Delete initiator objects after they are removed from a volume access group. false: Do not delete initiator objects after they are removed from a volume access group.

remove_node_sslcertificate()
You can use the RemoveNodeSSLCertificate method to remove the user SSL certificate and private key for the management node. After the certificate and private key are removed, the management node is configured to use the default certificate and private key..

remove_nodes(*nodes*)
You can use RemoveNodes to remove one or more nodes that should no longer participate in the cluster. Before removing a node, you must remove all drives the node contains using the RemoveDrives method. You cannot remove a node until the RemoveDrives process has completed and all data has been migrated away from the node. After you remove a node, it registers itself as a pending node. You can add the node again or shut it down (shutting the node down removes it from the Pending Node list). :param nodes: [required] List of NodeIDs for the nodes to be removed. :type nodes: int

remove_sslcertificate()
You can use the RemoveSSLCertificate method to remove the user SSL certificate and private key for the cluster. After the certificate and private key are removed, the cluster is configured to use the default certificate and private key.

remove_virtual_network(*virtual_network_id=None*, *virtual_network_tag=None*)
RemoveVirtualNetwork enables you to remove a previously added virtual network. Note: This method requires either the virtualNetworkID or the virtualNetworkTag as a parameter, but not both. :param virtualNetworkID: Network ID that identifies the virtual network to remove. :type virtualNetworkID: int

Parameters **virtualNetworkTag** (*int*) – Network tag that identifies the virtual network to remove.

remove_volume_pair(*volume_id*)
RemoveVolumePair enables you to remove the remote pairing between two volumes. Use this method on both the source and target volumes that are paired together. When you remove the volume pairing information, data is no longer replicated to or from the volume. :param volumeID: [required] The ID of the volume on which to stop the replication process. :type volumeID: int

remove_volumes_from_volume_access_group(*volume_access_group_id*, *volumes*)
The RemoveVolumeFromVolumeAccessGroup method enables you to remove volumes from a volume access group. :param volumeAccessGroupID: [required] The ID of the volume access group to remove volumes from. :type volumeAccessGroupID: int

Parameters **volumes** (*int*) – [required] The ID of the volume access group to remove volumes from.

reset_drives(*drives*, *force*)
ResetDrives enables you to proactively initialize drives and remove all data currently residing on a drive. The drive can then be reused in an existing node or used in an upgraded node. This method requires the force parameter to be included in the method call. :param drives: [required] List of device names (not driveIDs) to reset. :type drives: str

Parameters **force** (*bool*) – [required] Required parameter to successfully reset a drive.

reset_node(*build*, *force*, *reboot=None*, *options=None*)
The ResetNode API method enables you to reset a node to the factory settings. All data, packages (software upgrades, and so on), configurations, and log files are deleted from the node when you call this method.

However, network settings for the node are preserved during this operation. Nodes that are participating in a cluster cannot be reset to the factory settings. The ResetNode API can only be used on nodes that are in an "Available" state. It cannot be used on nodes that are "Active" in a cluster, or in a "Pending" state. Caution: This method clears any data that is on the node. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later. :param build: [required] Specifies the URL to a remote Element software image to which the node will be reset. :type build: str

Parameters

- **force** (*bool*) – [required] Required parameter to successfully reset the node.
- **reboot** (*bool*) – Set to true if you want to reboot the node.
- **options** (*str*) – Used to enter specifications for running the reset operation. Available options: 'edebug': '', 'sf_auto': '0', 'sf_bond_mode': 'ActivePassive', 'sf_check_hardware': '0', 'sf_disable_otpw': '0', 'sf_fa_host': '', 'sf_hostname': 'SF-FA18', 'sf_inplace': '1', 'sf_inplace_die_action': 'kexec', 'sf_inplace_safe': '0', 'sf_keep_cluster_config': '0', 'sf_keep_data': '0', 'sf_keep_hostname': '0', 'sf_keep_network_config': '0', 'sf_keep_paths': '/var/log/hardware.xml 'sf_max_archives': '5', 'sf_nvram_size': '', 'sf_oldroot': '', 'sf_postinst_erase_root_drive': '0', 'sf_root_drive': '', 'sf_rtfi_cleanup_state': '', 'sf_secure_erase': '1', 'sf_secure_erase_retries': '5', 'sf_slice_size': '', 'sf_ssh_key': '1', 'sf_ssh_root': '1', 'sf_start_rtfi': '1', 'sf_status_httpserver': '1', 'sf_status_httpserver_stop_delay': '5m', 'sf_status_inject_failure': '', 'sf_status_json': '0', 'sf_support_host': 'sfsupport.solidfire.com', 'sf_test_hardware': '0', 'sf_upgrade': '0', 'sf_upgrade_firmware': '0', 'sf_upload_logs_url': ''

restart_networking (*force*)

The RestartNetworking API method enables you to restart the networking services on a node. Warning: This method restarts all networking services on a node, causing temporary loss of networking connectivity. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later. :param force: [required] Required parameter to successfully reset the node. :type force: bool

restart_services (*force*, *service=None*, *action=None*)

The RestartServices API method enables you to restart the services on a node. Caution: This method causes temporary node services interruption. Exercise caution when using this method. Note: This method is available only through the per-node API endpoint 5.0 or later. :param force: [required] Required parameter to successfully restart services on a node. :type force: bool

Parameters

- **service** (*str*) – Service name to be restarted.
- **action** (*str*) – Action to perform on the service (start, stop, restart).

restore_deleted_volume (*volume_id*)

RestoreDeletedVolume marks a deleted volume as active again. This action makes the volume immediately available for iSCSI connection. :param volumeID: [required] VolumeID of the deleted volume to be restored. :type volumeID: int

rollback_to_group_snapshot (*group_snapshot_id*, *save_current_state*, *name=None*, *attributes=None*)

RollbackToGroupSnapshot enables you to roll back all individual volumes in a snapshot group to each volume's individual snapshot. Note: Rolling back to a group snapshot creates a temporary snapshot of each volume within the group snapshot. Snapshots are allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5. :param groupSnapshotID: [required] Specifies the unique ID of the group snapshot. :type groupSnapshotID: int

Parameters

- **saveCurrentState** (*bool*) – [required] Specifies whether to save an active volume image or delete it. Values are: true: The previous active volume image is kept. false: (default) The previous active volume image is deleted.
- **name** (*str*) – Name for the group snapshot of the volume's current state that is created if "saveCurrentState" is set to true. If you do not give a name, the name of the snapshots (group and individual volume) are set to a timestamp of the time that the rollback occurred.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

rollback_to_snapshot(*volume_id*, *snapshot_id*, *save_current_state*, *name=None*, *attributes=None*)

RollbackToSnapshot enables you to make an existing snapshot of the "active" volume image. This method creates a new snapshot from an existing snapshot. The new snapshot becomes "active" and the existing snapshot is preserved until you delete it. The previously "active" snapshot is deleted unless you set the parameter saveCurrentState to true. Note: Creating a snapshot is allowed if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5. :param volumeID: [required] VolumeID for the volume. :type volumeID: int

Parameters

- **snapshotID** (*int*) – [required] The ID of a previously created snapshot on the given volume.
- **saveCurrentState** (*bool*) – [required] Specifies whether to save an active volume image or delete it. Values are: true: The previous active volume image is kept. false: (default) The previous active volume image is deleted.
- **name** (*str*) – Name for the snapshot. If unspecified, the name of the snapshot being rolled back to is used with "- copy" appended to the end of the name.
- **attributes** (*dict*) – List of name-value pairs in JSON object format.

secure_erase_drives(*drives*)

SecureEraseDrives enables you to remove any residual data from drives that have a status of "available." You might want to use this method when replacing a drive nearing the end of its service life that contained sensitive data. This method uses a Security Erase Unit command to write a predetermined pattern to the drive and resets the encryption key on the drive. This asynchronous method might take up to two minutes to complete. You can use GetAsyncResult to check on the status of the secure erase operation. You can use the ListDrives method to obtain the driveIDs for the drives you want to secure erase. :param drives: [required] List of driveIDs to be secure erased. :type drives: int

set_cluster_config(*cluster*)

The SetClusterConfig API method enables you to set the configuration this node uses to communicate with the cluster it is associated with. To see the states in which these objects can be modified, see Cluster Object Attributes. To display the current cluster interface settings for a node, run the GetClusterConfig API method. Note: This method is available only through the per-node API endpoint 5.0 or later. :param cluster: [required] Objects that are changed for the cluster interface settings. :type cluster: ClusterConfig

set_config(*config*)

The SetConfig API method enables you to set all the configuration information for the node. This includes the same information available via calls to SetClusterConfig and SetNetworkConfig in one API method. Note: This method is available only through the per-node API endpoint 5.0 or later. Caution: Changing the "bond-mode" on a node can cause a temporary loss of network connectivity. Exercise caution when using this method. :param config: [required] Objects that you want changed for the cluster interface settings. :type config: ConfigParams

set_default_qos(*min_iops=None*, *max_iops=None*, *burst_iops=None*)

SetDefaultQoS enables you to configure the default Quality of Service (QoS) values (measured in inputs and outputs per second, or IOPS) for a volume. For more information about QoS in a SolidFire cluster,

see the User Guide. :param minIOPS: The minimum number of sustained IOPS provided by the cluster to a volume. :type minIOPS: int

Parameters

- **maxIOPS** (*int*) – The maximum number of sustained IOPS provided by the cluster to a volume.
- **burstIOPS** (*int*) – The maximum number of IOPS allowed in a short burst scenario.

set_login_banner(*banner=None*, *enabled=None*)

You can use the SetLoginBanner method to set the active Terms of Use banner users see when they log on to the web interface. :param banner: The desired text of the Terms of Use banner. :type banner: str

Parameters **enabled** (*bool*) – The status of the Terms of Use banner. Possible values: true: The Terms of Use banner is displayed upon web interface login. false: The Terms of Use banner is not displayed upon web interface login.

set_login_session_info(*timeout*)

You can use SetLoginSessionInfo to set the period of time that a session's login authentication is valid. After the log in period elapses without activity on the system, the authentication expires. New login credentials are required for continued access to the cluster after the timeout period has elapsed. :param timeout: [required] Cluster authentication expiration period. Formatted in HH:mm:ss. For example, 01:30:00, 00:90:00, and 00:00:5400 can be used to equal a 90 minute timeout period. The default value is 30 minutes. The minimum value is 1 minute. :type timeout: str

set_network_config(*network*)

The SetNetworkConfig API method enables you to set the network configuration for a node. To display the current network settings for a node, run the GetNetworkConfig API method. Note: This method is available only through the per-node API endpoint 5.0 or later. Changing the "bond-mode" on a node can cause a temporary loss of network connectivity. Exercise caution when using this method. :param network: [required] An object containing node network settings to modify. :type network: NetworkParams

set_node_sslcertificate(*certificate*, *private_key*)

You can use the SetNodeSSLCertificate method to set a user SSL certificate and private key for the management node. :param certificate: [required] The PEM-encoded text version of the certificate. :type certificate: str

Parameters **privateKey** (*str*) – [required] The PEM-encoded text version of the private key.

set_ntp_info(*servers*, *broadcastclient=None*)

SetNtpInfo enables you to configure NTP on cluster nodes. The values you set with this interface apply to all nodes in the cluster. If an NTP broadcast server periodically broadcasts time information on your network, you can optionally configure nodes as broadcast clients. Note: NetApp recommends using NTP servers that are internal to your network, rather than the installation defaults. :param servers: [required] List of NTP servers to add to each nodes NTP configuration. :type servers: str

Parameters **broadcastclient** (*bool*) – Enables every node in the cluster as a broadcast client.

set_remote_logging_hosts(*remote_hosts*)

SetRemoteLoggingHosts enables you to configure remote logging from the nodes in the storage cluster to a centralized log server or servers. Remote logging is performed over TCP using the default port 514. This API does not add to the existing logging hosts. Rather, it replaces what currently exists with new values specified by this API method. You can use GetRemoteLoggingHosts to determine what the current logging hosts are, and then use SetRemoteLoggingHosts to set the desired list of current and new logging hosts. :param remoteHosts: [required] A list of hosts to send log messages to. :type remoteHosts: LoggingServer

set_snmp_acl (*networks*, *usm_users*)
SetSnmpACL enables you to configure SNMP access permissions on the cluster nodes. The values you set with this interface apply to all nodes in the cluster, and the values that are passed replace, in whole, all values set in any previous call to SetSnmpACL. Also note that the values set with this interface replace all network or usmUsers values set with the older SetSnmpInfo. :param networks: [required] List of networks and what type of access they have to the SNMP servers running on the cluster nodes. See SNMP Network Object for possible "networks" values. This parameter is required if SNMP v3 is disabled. :type networks: SnmpNetwork

Parameters **usmUsers** (SnmpV3UsmUser) – [required] List of users and the type of access they have to the SNMP servers running on the cluster nodes.

set_snmp_info (*networks=None*, *enabled=None*, *snmp_v3_enabled=None*, *usm_users=None*)
SetSnmpInfo enables you to configure SNMP version 2 and version 3 on cluster nodes. The values you set with this interface apply to all nodes in the cluster, and the values that are passed replace, in whole, all values set in any previous call to SetSnmpInfo. Note: SetSnmpInfo is deprecated. Use the EnableSnmp and SetSnmpACL methods instead. :param networks: List of networks and what type of access they have to the SNMP servers running on the cluster nodes. See the SNMP Network Object for possible "networks" values. This parameter is required only for SNMP v2. :type networks: SnmpNetwork

Parameters

- **enabled** (*bool*) – If set to true, SNMP is enabled on each node in the cluster.
- **snmpV3Enabled** (*bool*) – If set to true, SNMP v3 is enabled on each node in the cluster.
- **usmUsers** (SnmpV3UsmUser) – If SNMP v3 is enabled, this value must be passed in place of the networks parameter. This parameter is required only for SNMP v3.

set_snmp_trap_info (*trap_recipients*, *cluster_fault_traps_enabled*, *cluster_fault_resolved_traps_enabled*, *cluster_event_traps_enabled*)
You can use SetSnmpTrapInfo to enable and disable the generation of cluster SNMP notifications (traps) and to specify the set of network host computers that receive the notifications. The values you pass with each SetSnmpTrapInfo method call replace all values set in any previous call to SetSnmpTrapInfo. :param trapRecipients: [required] List of hosts that are to receive the traps generated by the Cluster Master. At least one object is required if any one of the trap types is enabled. :type trapRecipients: SnmpTrapRecipient

Parameters

- **clusterFaultTrapsEnabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterFaultNotification is sent to the configured list of trap recipients when a cluster fault is logged. The default value is false.
- **clusterFaultResolvedTrapsEnabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterFaultResolvedNotification is sent to the configured list of trap recipients when a cluster fault is resolved. The default value is false.
- **clusterEventTrapsEnabled** (*bool*) – [required] If the value is set to true, a corresponding solidFireClusterEventNotification is sent to the configured list of trap recipients when a cluster event is logged. The default value is false.

set_sslcertificate (*certificate*, *private_key*)
You can use the SetSSLCertificate method to set a user SSL certificate and a private key for the cluster. :param certificate: [required] The PEM-encoded text version of the certificate. :type certificate: str

Parameters **privateKey** (*str*) – [required] The PEM-encoded text version of the private key.

shutdown (*nodes*, *option=None*)
The Shutdown API method enables you to restart or shutdown a node that has not yet been added to a

cluster. To use this method, log in to the MIP for the pending node, and enter the "shutdown" method with either the "restart" or "halt" options. :param nodes: [required] List of NodeIDs for the nodes to be shutdown. :type nodes: int

Parameters **option** (*str*) – Specifies the action to take for the node shutdown. Possible values are: restart: Restarts the node. halt: Shuts down the node.

snmp_send_test_traps()

SnmpSendTestTraps enables you to test SNMP functionality for a cluster. This method instructs the cluster to send test SNMP traps to the currently configured SNMP manager.

start_bulk_volume_read(*volume_id*, *format*, *snapshot_id=None*, *script=None*, *script_parameters=None*, *attributes=None*)

StartBulkVolumeRead enables you to initialize a bulk volume read session on a specified volume. Only two bulk volume processes can run simultaneously on a volume. When you initialize the session, data is read from a SolidFire storage volume for the purposes of storing the data on an external backup source. The external data is accessed by a web server running on an SF-series node. Communications and server interaction information for external data access is passed by a script running on the storage system. At the start of a bulk volume read operation, a snapshot of the volume is made and the snapshot is deleted when the read is complete. You can also read a snapshot of the volume by entering the ID of the snapshot as a parameter. When you read a previous snapshot, the system does not create a new snapshot of the volume or delete the previous snapshot when the read completes. Note: This process creates a new snapshot if the ID of an existing snapshot is not provided. Snapshots can be created if cluster fullness is at stage 2 or 3. Snapshots are not created when cluster fullness is at stage 4 or 5. :param volumeID: [required] The ID of the volume to be read. :type volumeID: int

Parameters

- **format** (*str*) – [required] The format of the volume data. It can be either of the following formats: uncompressed: Every byte of the volume is returned without any compression. native: Opaque data is returned that is smaller and more efficiently stored and written on a subsequent bulk volume write.
- **snapshotID** (*int*) – The ID of a previously created snapshot used for bulk volume reads. If no ID is entered, a snapshot of the current active volume image is made.
- **script** (*str*) – The executable name of a script. If unspecified, the key and URL is necessary to access SF-series nodes. The script is run on the primary node and the key and URL is returned to the script so the local web server can be contacted.
- **scriptParameters** (*dict*) – JSON parameters to pass to the script.
- **attributes** (*dict*) – JSON attributes for the bulk volume job.

start_bulk_volume_write(*volume_id*, *format*, *script=None*, *script_parameters=None*, *attributes=None*)

StartBulkVolumeWrite enables you to initialize a bulk volume write session on a specified volume. Only two bulk volume processes can run simultaneously on a volume. When you initialize the write session, data is written to a SolidFire storage volume from an external backup source. The external data is accessed by a web server running on an SF-series node. Communications and server interaction information for external data access is passed by a script running on the storage system. :param volumeID: [required] The ID of the volume to be written to. :type volumeID: int

Parameters

- **format** (*str*) – [required] The format of the volume data. It can be either of the following formats: uncompressed: Every byte of the volume is returned without any compression. native: Opaque data is returned that is smaller and more efficiently stored and written on a subsequent bulk volume write.

- **script** (*str*) – The executable name of a script. If unspecified, the key and URL are necessary to access SF-series nodes. The script runs on the primary node and the key and URL is returned to the script, so the local web server can be contacted.
- **scriptParameters** (*dict*) – JSON parameters to pass to the script.
- **attributes** (*dict*) – JSON attributes for the bulk volume job.

start_cluster_pairing()
You can use the StartClusterPairing method to create an encoded key from a cluster that is used to pair with another cluster. The key created from this API method is used in the CompleteClusterPairing API method to establish a cluster pairing. You can pair a cluster with a maximum of four other clusters.

start_volume_pairing(*volume_id*, *mode=None*)
StartVolumePairing enables you to create an encoded key from a volume that is used to pair with another volume. The key that this method creates is used in the CompleteVolumePairing API method to establish a volume pairing. :param volumeID: [required] The ID of the volume on which to start the pairing process. :type volumeID: int

Parameters **mode** (*str*) – The mode of the volume on which to start the pairing process. The mode can only be set if the volume is the source volume. Possible values are: Async: (default if no mode parameter specified) Writes are acknowledged when they complete locally. The cluster does not wait for writes to be replicated to the target cluster. Sync: Source acknowledges write when the data is stored locally and on the remote cluster. SnapshotsOnly: Only snapshots created on the source cluster will be replicated. Active writes from the source volume are not replicated.

test_connect_ensemble(*ensemble=None*)
The TestConnectEnsemble API method enables you to verify connectivity with a specified database ensemble. By default, it uses the ensemble for the cluster that the node is associated with. Alternatively, you can provide a different ensemble to test connectivity with. Note: This method is available only through the per-node API endpoint 5.0 or later. :param ensemble: Uses a comma-separated list of ensemble node cluster IP addresses to test connectivity. This parameter is optional. :type ensemble: str

test_connect_mvip(*mvip=None*)
The TestConnectMvip API method enables you to test the management connection to the cluster. The test pings the MVIP and executes a simple API method to verify connectivity. Note: This method is available only through the per-node API endpoint 5.0 or later. :param mvip: If specified, tests the management connection of a different MVIP. You do not need to use this value when testing the connection to the target cluster. This parameter is optional. :type mvip: str

test_connect_svip(*svip=None*)
The TestConnectSvip API method enables you to test the storage connection to the cluster. The test pings the SVIP using ICMP packets, and when successful, connects as an iSCSI initiator. Note: This method is available only through the per-node API endpoint 5.0 or later. :param svip: If specified, tests the storage connection of a different SVIP. You do not need to use this value when testing the connection to the target cluster. This parameter is optional. :type svip: str

test_drives(*minutes=None*, *force=None*)
You can use the TestDrives API method to run a hardware validation on all drives on the node. This method detects hardware failures on the drives (if present) and reports them in the results of the validation tests. You can only use the TestDrives method on nodes that are not "active" in a cluster. Note: This test takes approximately 10 minutes. Note: This method is available only through the per-node API endpoint 5.0 or later. :param minutes: Specifies the number of minutes to run the test. :type minutes: int

Parameters **force** (*bool*) – Required parameter to successfully test the drives on the node.

test_ldap_authentication(*username*, *password*, *ldap_configuration=None*)
The TestLdapAuthentication method enables you to validate the currently enabled LDAP authentication

settings. If the configuration is correct, the API call returns the group membership of the tested user. :param username: [required] The username to be tested. :type username: str

Parameters

- **password** (*str*) – [required] The password for the username to be tested.
- **ldapConfiguration** (LdapConfiguration) – An ldapConfiguration object to be tested. If specified, the API call tests the provided configuration even if LDAP authentication is disabled.

test_ping(*attempts=None*, *hosts=None*, *total_timeout_sec=None*, *packet_size=None*, *ping_timeout_msec=None*, *prohibit_fragmentation=None*)

You can use the TestPing API method to validate the connection to all the nodes in a cluster on both 1G and 10G interfaces by using ICMP packets. The test uses the appropriate MTU sizes for each packet based on the MTU settings in the network configuration. Note: This method is available only through the per-node API endpoint 5.0 or later. :param attempts: Specifies the number of times the system should repeat the test ping. The default value is 5. :type attempts: int

Parameters

- **hosts** (*str*) – Specifies a comma-separated list of addresses or hostnames of devices to ping.
- **totalTimeoutSec** (*int*) – Specifies the length of time the ping should wait for a system response before issuing the next ping attempt or ending the process.
- **packetSize** (*int*) – Specifies the number of bytes to send in the ICMP packet that is sent to each IP. The number must be less than the maximum MTU specified in the network configuration.
- **pingTimeoutMsec** (*int*) – Specifies the number of milliseconds to wait for each individual ping response. The default value is 500 ms.
- **prohibitFragmentation** (*bool*) – Specifies that the Do not Fragment (DF) flag is enabled for the ICMP packets.

update_bulk_volume_status(*key*, *status*, *percent_complete=None*, *message=None*, *attributes=None*)

You can use UpdateBulkVolumeStatus in a script to update the status of a bulk volume job that you started with the StartBulkVolumeRead or StartBulkVolumeWrite methods. :param key: [required] The key assigned during initialization of a StartBulkVolumeRead or StartBulkVolumeWrite session. :type key: str

Parameters

- **status** (*str*) – [required] The status of the given bulk volume job. The system sets the status. Possible values are: running: Jobs that are still active. complete: Jobs that are done. failed: Jobs that failed.
- **percentComplete** (*str*) – The completed progress of the bulk volume job as a percentage value.
- **message** (*str*) – The message returned indicating the status of the bulk volume job after the job is complete.
- **attributes** (*dict*) – JSON attributes; updates what is on the bulk volume job.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

A

B

C

D

E

F

G

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z