Avere FXT 4.7 API Guide

2017-09-11

Table of Contents

About This Document

The Avere FXT 4.7 API Guide is written for system administrators who want to programmically administer an Avere cluster. It lists and describes the XML-RPC application programming interfaces (API) that are provided with the Avere product. It assumes that you are familiar with the Avere OS administrative .

XML-RPC is a set of implementations that allow software running on different operating systems and environments to make procedure calls over the Internet (using the HTTP protocol). It uses XML for the encoding.

This document assumes that you are familiar with programming an XML-RPC interface. Programming examples are provided in Python; however, you can use any programming language with an XML-RPC implementation. For information about XML-RPC, consult the following resources:

- The XML-RPC home page – www.xmlrpc.com/
- The XML-RPC specification – www.xmlrpc.com/spec
- Languages with XML-RPC implementations – en.wikipedia.org/wiki/XML-RPC
- XML-RPC tutorials – http://www.tutorialspoint.com/xml-rpc/index.htm

About the Avere OS XML-RPC API

The Avere OS XML-RPC API is provided for storage administrators who want an alternative to the Avere Control Panel for administrative tasks. The API does not cover every function available in the Avere Control Panel. If you cannot find an XML-RPC method that corresponds to the administrative function you need to perform, use the Avere Control Panel instead.

Caution: The API is subject to ongoing development and revision. Before using administrative routines intended for production environments, use the following standard XML-RPC methods to obtain current information about the API:

- system.listMethods
- system.listModules
- system.methodHelp

Accessing the Avere XML-RPC Server

To access the Avere XML-RPC server, direct your XML-RPC client application to `http://cluster_management_URL/python/rpc2.py` where `cluster_management_URL` is either the name or IP address of the Avere cluster's management interface.

Note: The examples in this document assume that you have a client connection called clientHandle. A sample client written in Python is provided at the end of this document.

Parameter Documentation

Each method is documented with the input and output parameters, with the type of each parameter in parentheses.

Element names within XML-RPC structures are fixed; if you are using a method with an input structure, you will need to use the exact name as documented. Other parameter names are simply for description, and not necessarily used by the API.

Most methods will return an activity ID (a UUID) that can be used as input to the `cluster.getActivity` method to return information about that activity. After the activity completes, any methods that have 'status' as their return parameter will generally return a string, which will either have a value of 'success' or a reason for failure. Otherwise, the returned values will simply depend upon the method.

NAME
admin.addSshKey

SYNOPSIS
admin.addSshKey(user,keyname,keyvalue) => status

DESCRIPTION
Adds or updates an ssh public key.

PARAMETERS
- user: (string) The administrative account for which the key is being added. Only 'admin' is currently supported.
- keyname: (string) A unique keyname
- keyvalue: (string) An SSH key, consisting of the key-type, the key, and an optional identifier, separated by a space

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.admin.addSshKey('admin','juser',
  'ssh-rsa AAA3NzaC1yc2AADAQABAABAQXVwu...bT4SgTcqoP juser@machine')
success
```

NAME
admin.addUser

SYNOPSIS
admin.addUser(user, permission, password) => status

DESCRIPTION
Adds an administrative user to a cluster.

PARAMETERS
- user: (string) The user name
- permission: (string) One of the following:
  - 'rw' for read-write administrative access, the default
  - 'ro' for read-only administrative access
- password: (string) The user's password

RETURNS
- status: (string) Either 'success' or a reason for failure. Note that this method also returns 'Failed' without a reason given if it is unable to find the user's password record.

EXAMPLE
```
print clientHandle.admin.addUser('juser','rw','gobbledygookpassword')
User juser already exists
print clientHandle.admin.addUser('newjuser','rw','gobbledygookpassword')
success
```

admin.changePermission

NAME
admin.changePermission

SYNOPSIS
admin.changePermission(user, newPermission) => status

DESCRIPTION
Changes the permission level of an administrative user.

PARAMETERS
- user: (string) The administrative user name

- newPermission: (string) One of the following:
  - 'rw' for read-write administrative access
  - 'ro' for read-only administrative access

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.admin.changePermission('juser','rw')
success

NAME
admin.changePwd

SYNOPSIS
admin.changePwd(user, newPassword) => status

DESCRIPTION
Changes the password of an administrative user.

PARAMETERS
- user: (string) The administrative user name
- newPassword: (string) The user's new password

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.admin.changePwd('juser','newSillyPassword')
success
```

admin.disableServiceAccessByNetworkRole

NAME

admin.disableServiceAccessByNetworkRole

SYNOPSIS

admin.disableServiceAccessByNetworkRole(service, networkRole) => status

DESCRIPTION

Disables access to an admin service for a given network type.

PARAMETERS

- service: (string) The name of the service to modify, either 'ssh' or 'http'
- networkRole: (string) The name of the network role to disable, either 'client' or 'management'

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

The following Python example disables both HTTP and SSH access on client-facing networks.

```
print clientHandle.admin.disableServiceAccessByNetworkRole('ssh','client')
success
print clientHandle.admin.disableServiceAccessByNetworkRole('http','client')
success
```

admin.disableServiceAuthentication

NAME

admin.disableServiceAuthentication

SYNOPSIS

admin.disableServiceAuthentication(service, auth_type) => status

DESCRIPTION

Disables password authentication type for a service.

PARAMETERS

- service: (string) The name of the service. Currently the only service that can be specified is 'ssh'.
- auth_type: (string) The authentication type to disable. Currently, the only authentication type that can be disabled is 'password', in which case, all SSH logins must use public key authentication.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

The following Python example disables password authentication for SSH access.

```
print clientHandle.admin.disableServiceAuthentication('ssh','password')
success
```

admin.enableServiceAccessByNetworkRole

NAME

admin.enableServiceAccessByNetworkRole

SYNOPSIS

admin.enableServiceAccessByNetworkRole(service, networkRole) => status

DESCRIPTION

Enables access to an admin service for a given network type.

PARAMETERS

- service: (string) The name of the service to modify, either 'ssh' or 'http'
- networkRole: (string) The name of the network role to enable, either 'client' or 'management'

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

The following Python example enables SSH access on client-facing networks.

```
print clientHandle.admin.enableServiceAccessByNetworkRole('ssh','client')
```

admin.enableServiceAuthentication

NAME

admin.enableServiceAuthentication

SYNOPSIS

admin.enableServiceAuthentication(service, auth_type) => status

DESCRIPTION

Enables the specified authentication type for a service. By default, authentication is by public key.

PARAMETERS

- service: (string) The name of the service. Currently the only service that can be specified is 'ssh'.
- auth_type: (string) The authentication type to enable, either 'password' or 'publickey'

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

The following Python example enables password authentication for SSH access.

```
print clientHandle.enableServiceAuthentication('ssh','password')
success
```

admin.getServiceAccessSettings

NAME
admin.getServiceAccessSettings

SYNOPSIS
admin.getServiceAccessSettings(service) => settingStruct

DESCRIPTION
Returns the current access settings for a service for each network role.

PARAMETERS
- service: (string) The name of the service, either 'ssh' or 'http'

RETURNS
- settingStruct: An XML-RPC struct containing the following name:value pairs:

  - networkRole: (string) The network's role, one of the following:
    - 'cluster', when the network is used for communication between the nodes
    - 'client', when the network is used for client-to-cluster communication
    - 'mgmt', when the network is used for administrative access (through the GUI or XML-RPC) to the cluster
  - enabled: (string) Either 'enabled' or 'disabled'

EXAMPLE
The following Python example displays SSH access settings for client-facing networks.

```
print clientHandle.admin.getServiceAccessSettings('ssh')
{'client': 'enabled', 'cluster': 'enabled', 'management': 'disabled'}
```

admin.getServiceAuthenticationSettings

NAME

admin.getServiceAuthenticationSettings

SYNOPSIS

admin.getServiceAuthenticationSettings(service) => settingStruct

DESCRIPTION

Returns the current service authentication settings.

PARAMETERS

- service: (string) The name of the service. Currently the only service that can be specified is 'ssh'.

RETURNS

- settingStruct: An XML-RPC struct containing the following name:value pairs:

  - auth_type: (string) The authentication type, either 'password' or 'publickey'
  - enabled: (string) Either 'enabled' or 'disabled'

EXAMPLE

The following Python example displays SSH authentication settings.

```
print clientHandle.getServiceAuthenticationSettings('ssh')
enabled
```

admin.listSshKeys

NAME

admin.listSshKeys

SYNOPSIS

admin.listSshKeys(user) => keylistStruct

DESCRIPTION

Returns any authorized SSH keys and their values for a specific user.

PARAMETERS

- user: (string) The administrative user name. Only 'admin' is supported currently.

RETURNS

- keylistStruct: An XML-RPC struct containing the following name:value pairs:

  - keyname: (string) A unique keyname
  - keyvalue: (string) An SSH key, consisting of the key-type, the key, and an optional identifier, separated by a space

EXAMPLE

The following Python example displays SSH key information.

```
print clientHandle.admin.listSshKeys('admin')
{'keyname': 'juser', 'keyvalue': 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABBAQ...
XVwubT4SgTcqoP juser@machine'}
```

NAME
admin.listUsers

SYNOPSIS
admin.listUsers() => array_of_structs

DESCRIPTION
Lists the administrative users in the cluster, with their permissions.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - name: (string) The administrative user name; 'admin' is the default
  - permission: (string) One of the following:
    - 'rw' for read-write administrative access
    - 'ro' for read-only administrative access

EXAMPLE
```
print clientHandle.admin.listUsers()
[{'name': 'admin', 'permission': 'rw'}, {'name': 'tony',
'permission': 'ro'}, {'name': 'me', 'permission': 'rw'},
{'name': 'dante', 'permission': 'rw'}]
```

NAME
admin.permission

SYNOPSIS
admin.permission(user) => permission

DESCRIPTION
Returns an administrative user's permission level.

PARAMETER
- user: (string) The administrative user name

RETURNS
- permission: (string) One of the following:
  - 'rw' for read-write administrative access
  - 'ro' for read-only administrative access

EXAMPLE
```
print clientHandle.admin.permission('juser')
ro
```

NAME
admin.removeSshKey

SYNOPSIS
admin.removeSshKey(user,keyname) => status

DESCRIPTION
Removes an existing public SSH key from the list of authorized keys.

PARAMETERS
- user: (string) The administrative account from which the key is being removed. Only 'admin' is currently supported.
- keyname: (string) The name of the key to be removed

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.admin.removeSshKey('admin','juser')
success
```

admin.removeUser

NAME
admin.removeUser

SYNOPSIS
admin.removeUser(user) => status

DESCRIPTION
Removes an administrative user.

PARAMETER
- user: (string) The administrative user name

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.admin.removeUser('dante')
success

NAME
alert.cancelOverride

SYNOPSIS
alert.cancelOverride(alertName, alertType) => status

DESCRIPTION
Cancels alert overrides, restoring original alert parameters.

PARAMETERS
- alertName: (string) One of the following:
  - The UUID of a single alert
  - The alert class, a comma-separated list of any for the following options:
    - clusterServices
    - NFSCIFS
    - hardwareFailure
    - directoryServices
    - network

- alertType: (string) The alert type, either 'alert' or 'class'

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.alert.cancelOverride('7f2c31e-8a85-11e2-8103-
  000c2a69ad8','alert')
success
```

NAME
alert.conditions

SYNOPSIS
alert.conditions([severity]) => array_of_structs

DESCRIPTION
Lists the currently active conditional alerts, optionally by severity.

PARAMETERS
- [severity]: (string) Optional. Minimum severity level of the alert ('red', 'yellow', or 'green'). When the severity level is specified, only conditions with that level or higher are returned. When no severity is specified, all current conditions are returned.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain some or all of the following name:value pairs:

  - activateTime: (string) When the alert was activated, in the format yyyy/mm/dd_hh:mm:ss
  - activateTimeRaw: (integer) When the alert was activated, given as one of the following:
    - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
    - For a dynamic graph, either zero (0) for 'now', or a negative number defining the number of seconds before 'now'.
  - active: (boolean) Whether the alert is still active (True) or not (False)
  - details: (string) Additional details on the alert, if any
  - dismissed: (boolean) Whether the alert has been dismissed (True) or not (False)
  - id: (deprecated) The UUID of the alert
  - [informational]: (boolean) Whether the alert is an event (True) or a condition (False). This field is not always included.
  - name: (string) The name of the alert
  - [alert_node]: (string) The node on which the alert was activated. This field is not always included.
  - [generating_process]: (string) The Avere OS process that generated the alert. This field is not always included.
  - rev: (deprecated) The revision number of the alert
  - severity: (string) One of 'red', 'yellow', or 'green'
  - xmldescription: (string) Description in XML

EXAMPLE
The following example contains the conditions for two alerts, each item in the array starting with "{'xmldescription'". The first alert is formatted so that the pairings can be easily noted; the second alert is in the format that will actually appear in a console.

You can use alert.get with the alert name or UUID to list a more readable form of this information.

```
print clientHandle.alert.conditions('yellow')
[{'xmldescription': '<Description>\n <message>\nThe machine account
  &lt;span param=&quot;cifsNetbios&quot;&gt;Dante-Golf-vs1-2-&lt;/span&gt;
    for vserver &lt;span param=&uot;vserver&quot;&gt;global&lt;/span&gt;
    is no longer joined to the CIFS Active Directory domain &lt;span
    param=&quot;ad_domain_name&quot;&gt;cifsqa.com&lt;/span&gt;.
```

Please verify whether the machine account has been removed or
disabled on the Active Directory Server.\n\n</message>\n <details>
Vserver: global Machine Account: Dante-Golf-vs1-2-
Domain: cifsqa.com</details>\n</Description>\n',
'activateTime': '2013/03/12_11:02:07',
'severity': 'yellow',
'activateTimeRaw': '1363100527',
'rev': 'c58c5e2b-8b25-11e2-af86-000c29153b30',
'dismissed': 'false',
'active': 'true',
'informational': 'false',
'id': 'c85cc1b4-24e4-11e2-9e17-000c29153b30',
'name': 'alert.cifsAdsPingFailedMachineAccountHasBeenRemoved_vsrv'},
{'xmldescription': '<Description>\n <message>\nThe cluster has more than
one node, but HA is not enabled. Click here to
configure HA.\n</message>\n</Description>\n', 'activateTime': '2012/11/12
_16:28:02', 'severity': 'green', 'activateTimeRaw': '1352755682', 'rev':'
d4c62ea9-2d0f-11e2-a02b-000c29153b30', 'dismissed': 'false', 'active': '
true', 'id': '783e6d70-28fa-11e2-a02b-000c29153b30', 'name': 'alert.haNo
tEnabledError'}]

NAME
alert.dismiss

SYNOPSIS
alert.dismiss(alert | alert_array) => status

DESCRIPTION
Dismisses one or more alerts.

PARAMETERS
One of the following:
- alert: (string) The alert name or UUID
- alert_array: (array) Alerts listed by name or UUID

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.alert.dismiss('alert.haNotEnabledError')
success
```

NAME
alert.events

SYNOPSIS
alert.events([severity]) => array_of_structs

DESCRIPTION
Lists the currently active event alerts, optionally by severity.

PARAMETER
- [severity]: (string) Optional. Minimum severity level of the alert ('red', 'yellow', or 'green'). When the severity level is specified, only conditions with that level or higher are returned. When no severity is specified, all current conditions are returned.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain some or all of the following name:value pairs:

  - xmldescription: (string) Description in XML
  - activateTime: (string) When the alert was activated, in the format yyyy/mm/dd_hh:mm:ss
  - severity: (string) One of 'red', 'yellow', or 'green'
  - activateTimeRaw: (integer) When the alert was activated, given as one of the following:
    - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
    - For a dynamic graph, either zero (0) for 'now', or a negative number defining the number of seconds before 'now'.
  - name: (string) The name of the alert
  - informational: (boolean) Whether the alert is an event (True) or a condition (False)
  - active: (boolean) Whether the alert is still active (True) or not (False)
  - dismissed: (boolean) Whether the alert has been dismissed (True) or not (False)
  - id: (deprecated) The UUID of the alert
  - rev: (deprecated) The revision number of the alert
  - [node]: (string) The node on which the alert was activated. This field is not always included.
  - [process]: (string) The Avere OS process that generated the alert This field is not always included.

EXAMPLE
```
print clientHandle.alert.events('yellow')
{'xmldescription': '<Description>\n  <message>\nCaching policy for &lt;span
param=&quot;corefiler&quot;&gt;grape&lt;/span&gt; has been modified.\n\n</message>\n  <deta
ils>New caching policy: {read-write, writeback time: 15, attribute checki
ng:never} (was {read-write, writeback time: 3600, attribute checking: never}
).</details>\n</Description>\n', 'activateTime': '2013/03/13_16:31:2
1','severity': 'yellow', 'activateTimeRaw': '1363206681', 'rev': 'f54272
4c-8c1c-11e2-af86-000c29153b30', 'dismissed': 'false', 'active': 'true',
'informational': 'true', 'id': 'f54271a3-8c1c-11e2-af86-000c29153b30', '
name': 'alert.f54271a3-8c1c-11e2-af86-000c29153b30'}
```

NAME
alert.get

SYNOPSIS
alert.get(name | alert_array) => alert_info_array

DESCRIPTION
Lists current information about the specified alerts.

PARAMETERS
One of the following:
- name: (string) The alert name or UUID
- alert_array: (array) Alerts listed by name or UUID.

RETURNS
- alert_info_array: An array of alert information that contains some or all of the following:

  - xmldescription: (string) Description in XML
  - activateTime: (string) When the alert was activated, in the format yyyy/mm/dd_hh:mm:ss
  - severity: (string) One of 'red', 'yellow', or 'green'
  - activateTimeRaw: (integer) When the alert was activated, given as one of the following:
    - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
    - For a dynamic graph, either zero (0) for 'now', or a negative number defining the number of seconds before 'now'.
  - rev: (deprecated) The revision number of the alert
  - dismissed: (boolean) Whether the alert has been dismissed (True) or not (False)
  - active: (boolean) Whether the alert is still active (True) or not (False)
  - informational: (boolean) Whether the alert is an event (True) or a condition (False)
  - id: (deprecated) The UUID of the alert
  - name: (string) The name of the alert
  - [node]: (string) The node on which the alert was activated. This field is not always included.
  - [process]: (string) The Avere OS process that generated the alert. This field is not always included.

EXAMPLE
```
print clientHandle.alert.get('alert.48cec5bd-10b7-11e3-8099-000c293a3789')
{'xmldescription': '<Description>\n  <message>\nAdvanced
networking is enabled.\n </message>\n</Description>\n',
'activateTime': '2013/08/29_10:28:36', 'severity': 'green', 'activateTimeRaw':
'1377786516', 'rev': '48cec627-10b7-11e3-8099-000c293a3789', 'dismissed':
'false', 'active': 'true', 'informational': 'true', 'id': '48cec5bd-10b7-11e3
-8099-000c293a3789', 'name': 'alert.48cec5bd-10b7-11e3-8099-000c293a3789'}
```

NAME
alert.history

SYNOPSIS
alert.history() => array_of_structs

DESCRIPTION
Lists information for all alerts that have been generated in the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain some or all of the following name:value pairs:

  - name: (string) The name of the alert
  - informational: (boolean) Whether the alert is an event (True) or a condition (False)
  - active: (boolean) Whether the alert is still active (True) or not (False)
  - dismissed: (boolean) Whether the alert has been dismissed (True) or not (False)
  - id: (deprecated) The UUID of the alert
  - rev: (deprecated) The revision number of the alert
  - severity: (string) One of 'red', 'yellow', or 'green'
  - activateTime: (string) When the alert was activated, in the format yyyy/mm/dd_hh:mm:ss
  - activateTimeRaw: (integer) When the alert was activated, given as one of the following:
    - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
    - For a dynamic graph, either zero (0) for 'now', or a negative number defining the number of seconds before 'now'.
  - xmldescription: (string) Description in XML
  - [node]: (string) The node on which the alert was activated. This field is not always included.
  - [process]: (string) The Avere OS process that generated the alert This field is not always included.

EXAMPLE
The following example is from a new cluster, which has only had one alert generated:

```
print clientHandle.alert.history()
{'xmldescription': '<Description>\n  <message>\nThe cluster has more
than one node, but HA is not enabled. Click &lt;a href=&quot;/fxt/ha.php&quot;&gt;he
re&lt;/a&gt; to configure HA.\n</message>\n</Description>\n', 'activateTime':
'2012/11/12_16:28:02', 'severity': 'green', 'activateTimeRaw': '1352755682', 'rev': 'd4c62ea9
-2d0f-11e2-a02b-000c29153b30', 'dismissed': 'false', 'active': 'true', 'id': '783e6d70-28fa-
11e2-a02b-000c29153b30', 'name': 'alert.haNotEnabledError'}
```

NAME
alert.listOverrides

SYNOPSIS
alert.listOverrides() => array_of_structs

DESCRIPTION
Lists any alert overrides active in the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain some or all of the following name:value pairs:

  - alarm: (string) The type of alarm setting for the override, one of the following:
    - 'email'
    - 'emailNow'
    - 'off'
  - type: (string) The alert type, either 'alert' or 'class'
  - name: (string) One of the following:
    - The UUID of a single alert
    - The alert class, a comma-separated list of any for the following options:
      - clusterServices
      - NFSCIFS
      - hardwareFailure
      - directoryServices
      - network

EXAMPLE
The following example is the returned array from a single override.

```
print clientHandle.alert.listOverrides()
{'alarm': 'email', 'type': 'alert', 'name': '7f2cd31e-8a85-11e2-8103-
000c29a69ad8'}
```

NAME
alert.override

SYNOPSIS
alert.override({overrideStruct}) => status

DESCRIPTION
Overrides alert parameters on a single alert or a set of alerts.

PARAMETERS
- overrideStruct: An XML-RPC struct containing the following name:value pairs:

  - name: (string) One of the following:
    - The UUID of a single alert
    - The alert class, a comma-separated list of any for the following options:
      - clusterServices
      - NFSCIFS
      - hardwareFailure
      - directoryServices
      - network

  - type: (string) The override type, either 'alert' or 'class'
  - alarm: (string) The type of alarm setting for the override, one of the following:
    - 'email'
    - 'emailNow'
    - 'off'

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.alert.override({'name':'7f2cd31e','type':'alert',
  'alarm': 'email'})
success
```

cachePolicy.create

NAME

cachePolicy.create

SYNOPSIS

cachePolicy.create(name, cacheMode, [writebackDelay], [checkAttributes], [localDirectories], [wtSchedule], [cacheQuota], [description]) => status

or

cachePolicy.create(attrs) => status

DESCRIPTION

Creates a cache policy with the given name and parameters. You may supply the parameters separately, or load them into a dict with the same name as above.

PARAMETERS

- name: (string) The human-readable name of the cache policy
- cacheMode: (string) 'read-write' or 'read'
- [writebackDelay]: (integer) Number of seconds that cluster should cache data before writing to filer
- [checkAttributes]: (dict) Dictionary for specifying the checkAttrPeriod and checkDirAttrPeriod
  - checkAttrPeriod: (integer) Enable attribute checking with this max interval in seconds
  - checkDirAttrPeriod:(integer) Enable directory attribute checking with this max interval in seconds
- [localDirectories]: (boolean) True for On, False for Off
- [wtSchedule]: (dict) Dictionary for specifying wtSchedName, wtPollWait, wtPollUrl
  - wtSchedName: (string) Name of write-through schedule to use
  - wtPollWait: (integer) Wait time in minutes once cluster has achieved write-through mode
  - wtPollUrl: (string) A URL which should be polled once write-through is achieved
- [cacheQuota]: (string) A cache quota policy for the core filer. Acceptable strings are:
  "" (for default "corefiler"), "uid", "export", "fsid", "qtree", "export,uid", "fsid,uid", "qtree,uid"
- [description]: (string) Free-formed description of the cache policy

OR

- attrs: (dict) A dict containing one or more of the above set of key/value pairs. 'name' and 'cacheMode' keys are required. Others are evaluated by the RPC appropriately to determine if they combine to make a valid cache policy.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

Using separate parameters:

```
print clientHandle.cachePolicy.create('My Cache Policy', 'read', 0, {'checkAttrPeriod':'30'}, True, {}, "", "Some description")
success
print clientHandle.cachePolicy.create('My Cache Policy', 'read-write', 30, {}, True, {'wtSchedName':'myschedule', 'wtPollWait':30}, "qtree", "Some description")
success
```

Using a dict:

```
print clientHandle.cachePolicy.create({'name':'Policy From Dict', 'cacheMode':'read'})
success
```

NAME
cachePolicy.delete

SYNOPSIS
cachePolicy.delete(name)

DESCRIPTION
Deletes a cachePolicy object.

PARAMETERS
- name (string) The name of the policy to delete

RETURNS
status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cachePolicy.delete("My Policy")

NAME
cachePolicy.get

SYNOPSIS
cachePolicy.get(name, [raw])

DESCRIPTION
Gets the cache policy with the given name.

PARAMETERS
name: (string) The name of the policy to get
[raw]: (bool) optional parameter, set to True if you want list to return the data in raw, system-level format. Default is False. Before calling modify(), you may first want to call get() with raw=False (default), change the desired attrs, and then complete the cycle with a call to modify().

RETURNS
status: (dict) Dictionary representing the policy

EXAMPLE
Default mode:

```
print clientHandle.cachePolicy.get("custom-cache-policy-1")
cacheMode        = 'read-write'
writebackDelay   = '43200'
name             = 'custom-cache-policy-1'
localDirectories = 'False'
description      = 'Custom cache policy created during cluster configuration on 07/14/15'
```

Raw mode:

```
print clientHandle.cachePolicy.get("Read Caching", True)
persistentDirectoriesOn  = '0'
writebackDelay           = '0'
description              = 'Use this cache policy when file read performance is the most critical resource of your workflow.  File and directory reads are cached by the cluster.  File and directory modifications pass directly to the core filer.  All clients must be directly mounting the Avere cluster when using the cache policy.'
writeThroughMode         = '1'
__name                   = 'cachePolicy.bc2f0fa9-8701-409b-bf1c-6a22824d57fe'
checkAttributesEnabled   = '0'
rwCheckAttributesAllowed = '0'
name                     = 'Read Caching'
```

cachePolicy.list

NAME
cachePolicy.list

SYNOPSIS
cachePolicy.list([raw]) => array_of_structs

DESCRIPTION
Returns an array of cachePolicy dictionary entries

PARAMETERS
[raw]: (bool) optional parameter, set to True if you want list to return the data in raw, system-level format. Default is raw=False.

RETURNS
- array_of_structs: An array of XML-RPC structs representing a cache policy object

EXAMPLE
Default output:
print clientHandle.cachePolicy.list()
{'cacheMode': 'read-write', 'writebackDelay': 43200, 'name': 'custom-cache-policy-1', 'localDirectories': False, 'description': 'A custom read-write cache policy'}

Raw output:
print clientHandle.cachePolicy.list(True)
{'persistentDirectoriesOn': '0', 'writebackDelay': '43200', '__owner': 'bdbab443-2a39-11e5-a671-000c29e17591', 'description': 'A custom read-write cache policy', 'system': '0', 'writeThroughMode': '0', '__name': 'cachePolicy.f2ef1c03-6b58-479a-85bd-421c8cf2493c', '__uuid': 'c4694510-ddd8-44c4-bbc1-f0d1713edb85', 'checkAttributesEnabled': '0', 'rwCheckAttributesAllowed': '0', '__rev': '079fa4eb-2bc6-11e5-b8b6-000c29e17591', 'name': 'custom-cache-policy-1'}

NAME
cachePolicy.listFilersUsing

SYNOPSIS
cachePolicy.listFilersUsing(name)

DESCRIPTION
Lists the core filers that are using the named cache policy.

PARAMETERS
- name (string) The name of the cache policy.

RETURNS
list: (list) The list of core filers using the policy (empty list for none).

EXAMPLE
print clientHandle.cachePolicy.list("Read Caching")
corefilerName

NAME

cachePolicy.modify

SYNOPSIS

cachePolicy.modify(name, writeMode, [writeBackDelay], [checkAttributes], [localDirectories], [wtSchedule], [cacheQuota], [description]) => success

DESCRIPTION

Modifies the cache policy with the given name, and saves the new parameters to it that are provided.

PARAMETERS

- name: (string) The human-readable name of the cache policy
- writeMode: (string) 'read-write' or 'read'
- [writeBackDelay]: (integer) Number of seconds that cluster should cache data before writing to filer
- [checkAttributes]: (dict) Dictionary for specifying the checkAttrPeriod and checkDirAttrPeriod
  - checkAttrPeriod: (integer) Enable attribute checking with this max interval in seconds
  - checkDirAttrPeriod:(integer) Enable directory attribute checking with this max interval in seconds
- [localDirectories]: (boolean) true for On, false for Off
- [wtSchedule]: (dict) Dictionary for specifying wtSchedName, wtPollWait, wtPollUrl
  - wtSchedName: (string) Name of write-through schedule to use
  - wtPollWait: (integer) Wait time in minutes once cluster has achieved write-through mode
  - wtPollUrl: (string) A URL which should be polled once write-through is achieved
- [cacheQuota]: (string) A cache quota policy for the core filer. Acceptable strings are: "" (for default "corefiler"), "uid", "export", "fsid", "qtree", "export,uid", "fsid,uid", "qtree,uid"
- [description]: (string) English description of the cache policy

RETURNS

- status: (string) Either 'success' or reason for failure

EXAMPLE

print clientHandle.cachePolicy.modify('My Cache Policy','read-write',30)

print clientHandle.cachePolicy.modify('My Cache Policy','read-write',30,{},True,{'wtSchedName':'my schedule','wtPollUrl':'http://someurl'})

success

cert.addCABundle

NAME
cert.addCABundle

SYNOPSIS
cert.addCABundle(args) => status

DESCRIPTION
Add a CA bundle to cluster

PARAMETERS
- args: An XML-RPC struct containing the following name:value pairs
  - name: (string) The name of the added CA bundle
  - pem: (string) The pem text of the added CA bundle
  - [note]: (string) Optional. Any text description added to the certificate

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cert.addCABundle({'name':'mybundle', 'pem':'-----BEGIN CER...-----'})
success

cert.addCRT

NAME
cert.addCRT

SYNOPSIS
cert.addCRT(crtText, [args]) => status

DESCRIPTION
Add CA-signed certificate to a pending client/server certificate or install a new CA certificate

PARAMETERS
- crtText: (string) The text string of the CA-signed certificate
- [args]: Optional. This is only needed if you want to provide extra arguments for adding a CA certificate. An XML-RPC struct containing the following name:value pairs
  - [note]: (string) Optional. Any text description added to the certificate

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cert.addCRT('------BEGIN CER.....------')
success
```

cert.delete

NAME
cert.delete

SYNOPSIS
cert.delete(certName, [issuer, serial, force]) => status

DESCRIPTION
Delete a certificate

PARAMETERS
- certName: (string) The name of a certificate. If it's a pending certificate, issuer and serial is not needed.
- [issuer]: (string) Optional. The issuer name of a certificate. Required if not deleting a pending certificate.
- [serial]: (string) Optional. The serial number of a certificate. Required if not deleting a pending certificate.
- [force]: (boolean)Optional. Whether warnings are displayed to delete in-use certificate

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cert.delete('certA', 'avere_CA', '1697')
successfully deleted certificate

NAME
cert.deleteCABundle

SYNOPSIS
cert.deleteCABundle(name) => status

DESCRIPTION
Delete a CA bundle from the cluster

PARAMETERS
- name: (string) The name of the deleting CA bundle

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cert.deleteCABundle('mybundle')
success

cert.generateCSR

NAME
cert.generateCSR

SYNOPSIS
cert.generateCSR(args, force) => CSR text

DESCRIPTION
Generate the detailed text of a certificate request.

PARAMETERS
- args: An XML-RPC struct containing the following name:value pairs

  - name: (string) The name of a certificate
  - country: (string) The country name in CSR subject
  - state: (string) The state name in CSR subject
  - location: (string) The location name in CSR subject
  - organization: (string) The organization name in CSR subject
  - unit: (string) The unit name in CSR subject
  - keySize: (string) The bits size used to generate the private key. Possible values are '2048' and '4096'. The default is '2048'.
  - type: (string) The type of the certificate. Possible values are 'CA', 'client', 'server', 'CABundle'.
  - [note]: (string) Optional. Any text description added to the certificate
- [force]: (boolean) Optional. Whether warnings are displayed during CSR generation to overwrite the existing pending certificate with same name

RETURNS
- certificate text: (string) The detailed text information of the specified certificate.

EXAMPLE
print clientHandle.cert.generateCSR('name':'certA', 'type':'client', 'country':'US', 'state':'PA', 'location':'Pitt', 'organization':'avere', 'unit':'test')

-----BEGIN CERTIFICATE REQUEST-----
MIICnTCCAYUCAQAwWDELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAlBBMQ0wCwYDVQQH
.....
.....
NvmKwzGN3XnEMYbKk3EpQO/lS6t8mzJLL+hy5Q5Mi1CY2UQa/vlDWq+wh6sC3hSO
YA==
-----END CERTIFICATE REQUEST-----

NAME

cert.get

SYNOPSIS

cert.get(certName, [options]) => array_of_struct

DESCRIPTION

Returns detailed information for the specified certificates.

PARAMETERS

- certName: (string) The name of a certificate

- options: An XML-RPC struct containing the following name:value pairs. It returns all certificates matching the name if you do not specify the following.
  - issuer: (string) The name of a certificate issuer
  - serial: (string) The serial number of a certificate from the issuer

RETURNS

- array_of_struct: (array) An array of valid certificate structs, including name, serial, issuer, type and expires.

EXAMPLE

```
print clientHandle.cert.get('certA')
[{'serial': '1B5C', 'expires': 'Nov  2 20:21:42 2024 GMT', 'type': 'client', 'name': 'certB', 'issuer': 'avere_CA'},
[{'serial': '1692', 'expires': 'Nov  2 20:21:47 2024 GMT', 'type': 'client', 'name': 'certB', 'issuer': 'avere_CA2'}]
print clientHandle.cert.get('certA', "{'issuer':'avere_CA'}")
[{'serial': '1B5C', 'expires': 'Nov  2 20:21:42 2024 GMT', 'type': 'client', 'name': 'certB', 'issuer': 'avere_CA'}]
```

cert.getCABundle

NAME
cert.getCABundle

SYNOPSIS
cert.getCABundle(name) => PEM text

DESCRIPTION
Get the PEM text of the CA bundle

PARAMETERS
- name: (string) The name of the CA bundle

RETURNS
- PEM text: (string) The complete PEM text of the CA bundle

EXAMPLE
```
print clientHandle.cert.getCABundle('mybundle')
success
```

NAME
cert.getCSR

SYNOPSIS
cert.getCSR(certName) => CSR text

DESCRIPTION
Returns detailed text of a previously generated certificate request.

PARAMETERS
- certName: (string) The name of a pending certificate

RETURNS
- certificate text: (string) The detailed text information of the specified certificate.

EXAMPLE
print clientHandle.cert.getCSR(certA)

```
-----BEGIN CERTIFICATE REQUEST-----
MIICnTCCAYUCAQAwWDELMAkGA1UEBhMCVVMxCzAJBgNVBAgMAlBBMQ0wCwYDVQQH
                .......
                .......
NvmKwzGN3XnEMYbKk3EpQO/lS6t8mzJLL+hy5Q5Mi1CY2UQa/vlDWq+wh6sC3hSO
YA==
-----END CERTIFICATE REQUEST-----
```

NAME
cert.getText

SYNOPSIS
cert.getText(certName, issuer, serial) => certificate text

DESCRIPTION
Returns detailed text information for the specified certificate.

PARAMETERS
- certName: (string) The name of a certificate
- issuer: (string) The name of a certificate issuer
- serial: (string) The serial number of a certificate

RETURNS
- certificate text: (string) The detailed text information of the specified certificate.

EXAMPLE
```
print clientHandle.cert.getText('certA', 'avere_CA', '1697')
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 5783 (0x1697)
   Signature Algorithm: sha256WithRSAEncryption
      Issuer: C=US, ST=PA, L=Pittsburgh, O=avere, OU=Lab,
CN=avere_CA/emailAddress=avere@averesystems.com
      Validity
         Not Before: Jan  5 22:14:17 2015 GMT
         Not After : Nov  2 22:14:17 2024 GMT
      Subject: C=US, ST=PA, L=Pitt, O=avere, OU=test, CN=certA
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
               00:b7:e0:13:01:18:2f:84:c5:30:bc:2e:e3:b0:11:
                  .......
                  .......
               28:ef
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         X509v3 Basic Constraints:
            CA:FALSE
         Netscape Cert Type:
            SSL Client
   Signature Algorithm: sha256WithRSAEncryption
       87:12:71:98:95:f7:d2:43:7c:ac:df:6e:08:3c:f4:1e:e3:d6:
                  .......
                  .......
       a0:6d:98:e0:df:02:c2:4f:56:93:21:1d:41:1e:15:3c:72:42:
       79:f3:77:ed
```

cert.importCert

NAME
cert.importCert

SYNOPSIS
cert.importCert(args) => status

DESCRIPTION
Imported a pair of signed certificate and its private key

PARAMETERS
- args: An XML-RPC struct containing the following name:value pairs
  - key: (string) The text string of the certificate private key
  - crt: (string) The text string of the CA-signed certificate
  - [note]: (string) Optional. Any text description added to the certificate

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cert.importCert({'key':'---BEGIN....', 'crt':'---BEGIN ...', 'note':'test')
success

cert.list

NAME
cert.list

SYNOPSIS
cert.list() => array_of_struct

DESCRIPTION
Lists certificates associated with the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_struct: (array) An array of valid certificate structs, including name, serial, issuer, type and expires.

EXAMPLE
print clientHandle.cert.list()
[{'serial': '1B5C', 'expires': 'Nov 2 20:21:42 2024 GMT', 'type': 'client', 'name': 'certB', 'issuer': 'avere_CA'} {'serial': '1697', 'expires': 'Nov 2 22:14:17 2024 GMT', 'type': 'client', 'name': 'certA', 'issuer': 'avere_CA'}]

NAME
cert.listAll

SYNOPSIS
cert.listAll() => array_of_struct

DESCRIPTION
Lists all certificates,including pending, associated with the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_struct: (array) An array of certificate structs, including name, serial, issuer, type and expires.

EXAMPLE
print clientHandle.cert.listAll()
[{'type': 'client', 'name': 'certA'},
{'serial': '1B5C', 'expires': 'Nov 2 20:21:42 2024 GMT', 'type': 'client', 'name': 'certB', 'issuer': 'avere_CA'}
{'serial': '1697', 'expires': 'Nov 2 22:14:17 2024 GMT', 'type': 'client', 'name': 'certA', 'issuer': 'avere_CA'}]

cert.setClusterSsl

NAME
cert.setClusterSsl

SYNOPSIS
cert.setClusterSsl(name, [issuer, serial]) => status

DESCRIPTION
Set the cluster wide SSL certificate

PARAMETERS
- name: (string) The name of the new cluster SSL certificate. Use 'default' if want to use the cluster self-signed certificate.
- [issuer]: (string) Optional. The issuer name of a certificate. Required if name is not 'default'.
- [serial]: (string) Optional. The serial number of a certificate. Required if name is not 'default'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cert.setClusterSsl('mysslcert', 'myca', '0FCE')
success
```

cert.setSystemCABundle

NAME
cert.setSystemCABundle

SYNOPSIS
cert.setSystemCABundle(name) => status

DESCRIPTION
Set the cluster wide system CA bundle

PARAMETERS
- name: (string) The name of the new CA bundle

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cert.setSystemCABundle('mybundle')
success
```

cifs.addLocalGroupMember

NAME

cifs.addLocalGroupMember

SYNOPSIS

cifs.addLocalGroupMember(vserverName, groupName, memberId)

DESCRIPTION

Add a user to a local group.

Supported groups:
* Administrators
* Run_As_Root

An attempt to add a member that is already in the local group will return 'success'.

PARAMETERS

- vserverName: (string) The name of the vserver.
- groupName: (string) The name of the local group.
- memberId: (string) The name or SID (security ID) of the user or group to add to the local group. SID elements must be represented as a decimal. Names must include the domain prefix (for example, "DOMAIN\User").

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.cifs.addLocalGroupMember('vserver1', 'Administrators', 'DOMAIN\juser')
success
print clientHandle.cifs.addLocalGroupMember('vserver1', 'Administrators', 'S-1-1-0')
success
```

cifs.addShare

NAME
cifs.addShare

SYNOPSIS
cifs.addShare(vserverName, shareName, nfsExport, [suffix],
accessControl, homeDir, [settings]) => status

DESCRIPTION
Creates a CIFS share on a vserver.

NOTE: This method replaces the deprecated cifs.newShare method.

PARAMETERS
- vserverName: (string) The name of the vserver on which the CIFS share is to be created
- shareName: (string) The name of the CIFS share that is to be created
- nfsExport: (string) The name of the NFS export that the CIFS share is to expose to CIFS clients
- [suffix]: (string) The optional path suffix relative to the NFS export path. For home shares, the suffix must contain one of the variable substitutions. The variable substitution %S is typically the best choice since it allows any user to access all other user home directories. The variable substitution %U allows a user to only access their own home directory. The variable substitution %u is used in cases where the CIFS username map feature maps the CIFS username to a different NFS username.
- accessControl: (string)
  - For a simple vserver, the access-control mechanism for the share, one of the following:
    - 'posix' for POSIX mode bits
    - 'nfsv4' for NFSv4 ACLs
    - 'cifs' for CIFS ACLs
  - For a GNS vserver, this field must be empty, because the access control is specified in junction XML-RPC calls.
- homeDir: (boolean) Whether the share should serve home directories (True) or not (False)

- [settings]: An optional XML-RPC struct containing the following name:value pairs, which are advanced settings for the share:

  - browseable: (string) Whether a Windows client can browse to the share, depending on the access permissions of the share, either 'yes' (the default) or 'no'
  - inherit permissions:
    (string) Whether new directories created under the share will inherit the permissions of their parent directory, 'no' (the default) or 'yes'

  - read only: (string) Specifies whether the share is read-only. The default is 'no'. Setting this option to 'yes' can lead to faster performance if the data will not need to be changed.

  - create mask: (octal) The mask for the UNIX permissions for a newly created file. The default is 0744.

  - security mask: (octal) The UNIX permissions that are set on a file whose permissions are changed by a Windows NT client from the native Windows NT security dialog box. The default is 0777.

  - directory mask: (octal) The UNIX permissions of a directory that is created with DOS permissions. The default is 0755.

- directory security mask:
(octal) The UNIX permissions that are set on
a directory whose permissions are changed by a Windows NT client
from the native Windows NT security dialog box. The default is 0777.
- force create mode:
(octal) The minimum set of UNIX permissions for
any file created by the Avere OS CIFS server. The default is 0000.
- force security mode:
(octal) The minimum set of UNIX permissions that
can be modified on a file whose permissions are changed by a Windows
NT client from the native Windows NT security dialog box.
The default is 07000.
- force directory mode:
(octal) The minimum set of UNIX permissions
for any directory created by the Avere OS CIFS server. The default is 0000.

- force directory security mode:
(octal) The minimum set of UNIX permissions that can be
modified on a directory whose permissions are changed by a Windows NT client
from the native Windows NT security dialog box. The default is 0000.

- force user: (string) The UNIX username that is assigned as the default
user for all users of the Avere OS CIFS server. There is no default value.

- force group: (string) The UNIX group name that is assigned as the default
group for all users of the Avere OS CIFS server. There is no default value.

- hide unreadable: (string) This setting controls CIFS access-based enumeration which hides files and
directories that a user can not access when listing directory contents. The
default is 'no' (disabled). When set to 'yes' (enabled) this setting can have a
performance impact due to access checks being made and this impact will be more
dramatic for directories that contain large numbers of files and/or directories.

- strict locking: (string) Enable a byte-range lock check prior to each read and write request.
The default is 'no' (disabled) which improves performance. While the SMB specification
requires the server-side lock check, in practice it is safe to disable this
because clients perform the lock check prior to sending read and write requests.

- oplocks: (string) Enables read/write oplocks. The default is 'yes' (enabled).

- level2 oplocks: (string) Enables read-only oplocks when oplocks are enabled. The default
is 'yes' (enabled) but this is only honored when the 'oplocks' parameter is
also 'yes' (enabled).

- read only optimized:
(string) A value of 'yes' enables performance related options that
are applicable to read-only shares. The default value is 'no'.

- guest ok: (string) A value of 'yes' allows users that are not present in the password
database to be mapped to the guest user. The default value is 'no'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cifs.addShare('vserver1', 'vserver1-cifs-share',

'default', '', 'posix', False)
success

cifs.addShareAce

NAME

cifs.addShareAce

SYNOPSIS

cifs.addShareAce(vserverName, shareName, shareAce)

DESCRIPTION

Add an ACE (access control entry) to the share level ACL (access control list) for a CIFS share.

If the share level ACL does not contain an ACE for the specified user or group, then an ACE for that user or group is added to the share level ACL using the specified type and permissions.

If the share level ACL already contains an ACE for the specified user or group, then an error is returned.

PARAMETERS

- vserverName: (string) The name of the vserver
- shareName: (string) The name of the CIFS share

- shareAce: An XML-RPC struct that contains the following name:value pairs:

  - id: (string) The name or SID (security ID) of a user or group, with SID elements represented as decimal. Names from a trusted domain must include the domain prefix (for example, "DOMAIN\UserOrGroup").

  - type: (string) The type of ACE (either 'ALLOW' or 'DENY')

  - perm: (string) The permissions to allow or deny, one of 'READ, 'CHANGE', or 'FULL'

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cifs.addShareAce('vserver1', 'example', {'id': 'juser',
  'type': 'DENY', 'perm': 'CHANGE'})
success
```

cifs.configure

NAME
cifs.configure

SYNOPSIS
cifs.configure(vserverName, cifsServerName, adminUserName, [adminPass], [organization]) => status

DESCRIPTION
Configures and enables CIFS service on a vserver.

PARAMETERS
- vserverName: (string) The name of the vserver on which CIFS is to be configured
- cifsServerName: (string) The name for the CIFS server (the NetBIOS name)
- adminUserName: (string) An administrative user with privileges to join the Active Directory domain
  usually in DOMAIN\user or user@FQDN format
- [adminPass]: (string) Optional. The administrative user's password
- [organization]: (string) Optional. The Organizational Unit (OU) of the CIFS configuration

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason
  for failure. If the activity is not complete, the activity UUID, which
  can be used as input for the cluster.getActivity and cluster.abortActivity
  methods.

EXAMPLE
```
print clientHandle.cifs.configure('vserver1', 'vserver1-cifs', 'DOMAIN\master-user',
  'supersecret', 'OUR_OU')
success
```

NAME
cifs.disable

SYNOPSIS
cifs.disable(vserverName) => status

DESCRIPTION
Disables CIFS service on a vserver.

PARAMETER
- vserverName: (string) The name of the vserver

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
print clientHandle.cifs.disable('vserver1-cifs')
success

cifs.enable

NAME
cifs.enable

SYNOPSIS
cifs.enable(vserverName) => status

DESCRIPTION
Enables CIFS service on a vserver.

PARAMETER
- vserverName: (string) The vserver on which the CIFS service is to be enabled

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cifs.enable('new-vserver')
success

NAME
cifs.getAdStatus

SYNOPSIS
cifs.getAdStatus(vserverName) => AD_status

DESCRIPTION
Returns information about the connection between a vserver and an Active Directory domain.

PARAMETER
- vserverName: The name of the vserver

RETURNS
- AD_status: An XML-RPC struct containing the following name:value pairs:

- distinguishedName:
(string) The distinguished name of the vserver on the Active Directory domain, given as a comma-separated list; for example:

CN=vserver1,CN=avereClus,DC=mySite,DC=example,DC=com

- userAccountControl:
(string) The Microsoft UAC (User Account Control) number flags for the user accounts in the Active Directory server

- dNSHostName: (string) The DNS hostname of the vserver

- msDS-AllowedToDelegateTo:
(array) The service principal names (SPNs) of services that can act on behalf of the vserver
- servicePrincipalName:
(array) The SPN of the vserver; for example:

CIFS/vserver1.mySite.example.com
CIFS/vserver1
HOST/vserver1.mySite.example.com
HOST/vserver1

EXAMPLE
print clientHandle.cifs.getAdStatus('gns')
{'distinguishedName': 'CN=8238dacc,CN=Computers,DC=dev,DC=cc,
DC=company,DC=com', 'name': '8238dacc', 'userAccountControl':
'16846848', 'dNSHostName': '8238dacc.dev.cc.company.com',
'msDS-AllowedToDelegateTo': ['cifs/grapnel'], 'servicePrincip
alName': ['HOST/8238dacc.dev.cc.company.com', 'HOST/8238DACC']}

cifs.getConfig

NAME
cifs.getConfig

SYNOPSIS
cifs.getConfig(vserverName) => cifsStruct

DESCRIPTION
Returns information about the CIFS configuration of a vserver.

PARAMETER
- vserverName: (string) The name of the vserver

RETURNS
- cifsStruct: An XML-RPC struct containing the following name:value pairs for the connection:

  - rev: (deprecated) The revision number of the configuration
  - enabled: (boolean) Whether CIFS service is enabled (True) or not (False)
  - id: (deprecated) The UUID of the configuration
  - CIFSServerName: (string) The name of the CIFS server (NetBIOS name)

EXAMPLE
```
print clientHandle.cifs.getConfig('vserver1')
{'rev': 'b875d01c-1028-11e3-8099-000c293a3789', 'enabled': False,
'id': 'b875cfd7-1028-11e3-8099-000c293a3789', 'CIFSServerName': '
ligo'}
```

cifs.getJoinStatus

NAME
cifs.getJoinStatus

SYNOPSIS
cifs.getJoinStatus(vserverName) => joinStatus

DESCRIPTION
Returns information about the connection between a vserver and an Active Directory domain.

PARAMETER
- vserverName: (string) The name of the vserver

RETURNS
- joinStatus: An XML-RPC struct containing the following name:value pairs for the connection:

  - joinResult: (string) The numeric value of the connection attempt, returned as a string. Zero ('0') indicates success; nonzero values indicate failure

  - joinStatus: (string) One of the following:
  - 'UNKNWON (internal error)': The method was unable to perform the query
  - 'UNKNOWN': The method was unable to contact the Active Directory domain's key distribution center, and therefore cannot determine the connection status. The method returns this status if CIFS is not yet configured for the vserver.
  - 'NOT JOINED': The vserver is not connected to the AD domain
  - 'JOINED': The vserver is connected to the AD domain

  - joinOutput: (string) Any messages emitted by the Active Directory domain during the query

EXAMPLE
print clientHandle.cifs.getJoinStatus('gns')
{'joinResult': '0', 'joinStatus': 'JOINED', 'joinOutput': 'Join is OK'}

cifs.getLocalGroupMembers

NAME

cifs.getLocalGroupMembers

SYNOPSIS

cifs.getLocalGroupMembers(vserverName, groupName)

DESCRIPTION

Returns the members of a local group.

Supported groups:
* Administrators
* Run_As_Root

PARAMETERS

- vserverName: (string) The name of the vserver
- groupName: (string) The name of the local group.

RETURNS

- members: An array of XML-RPC structs that that contains the name:value pairs:

  - name: (string) The name of a user or group, or an empty string if the name could not be determined.
  - sid: (string) The SID of a user or group, with SID elements represented as decimal.

EXAMPLE

```
print clientHandle.cifs.getLocalGroupMembers('vserver1', 'Administrators')
[{'name': 'DOMAIN\juser', 'sid': 'S-1-1-0'}, {'name': 'DOMAIN\kuser', 'sid': 'S-1-1-1'}]
```

cifs.getOptions

NAME
cifs.getOptions

SYNOPSIS
cifs.getOptions(vserverName) => cifs_options

DESCRIPTION
Returns the CIFS options set on a vserver.

PARAMETER
- vserverName: (string) The name of the vserver

RETURNS
- cifs_options: An XML-RPC struct containing the following name:value pair for the connection:

  - smb2: (string) Specifies whether the CIFS service on the vserver will allow clients to use the SMB2 protocol ('yes' or 'no').

  - guest_account: (string) CIFS shares with guest access enabled will map users that are not present in the password database to the user specified as the guest account. The specified account may be prefixed with the desired domain (e.g. DOMAIN\nobody). The default value is "nobody".

  - native_identity: (string) A value of 'yes' indicates that CIFS users which do not have a UID and primary GID value available from the Directory Services User Name download can access junctions with CIFS ACL access control using the native Active Directory SID identity. A value of 'no' indicates that all CIFS users are required to have UID and primary GID values assigned. The default value is 'yes' for newly created vservers.

  - read_only_optimized:
    (string) A value of 'yes' enables performance related options for the entire vserver that are applicable to read-only shares. Additional performance changes are enabled when this type of optimization is enabled at the vserver level rather than the individual share level. However, all shares for the vserver are read only. The CIFS service must be disabled and then enabled after changing this parameter. The default value is 'no'.

EXAMPLE
print clientHandle.cifs.getOptions('gns')
{'smb2': 'yes', 'guest_account': 'aUser', 'native_identity': 'yes', 'read_only_optimized': 'no'}

cifs.getShare

NAME
cifs.getShare

SYNOPSIS
cifs.getShare(vserverName, shareName) => cifsInfoStruct

DESCRIPTION
Returns information about a CIFS share.

PARAMETERS
- vserverName: (string) The name of the vserver
- shareName: (string) The name of the CIFS share

RETURNS
- cifsInfoStruct: An XML-RPC struct containing the following name:value pairs for the share:

  - accessControl: (string) The access-control mechanism for the share, one of the following:
    - 'posix' for POSIX mode bits
    - 'nfsv4' for NFSv4 ACLs
    - 'cifs' for CIFS ACLs
  - homeDir: (string) Specifies whether the share is the home share ('yes' or 'no')
  - shareName: (string) The name of the CIFS share
  - rev: (deprecated) The revision number of the CIFS share
  - export: (string) The NFS export that the CIFS share is making available to CIFS clients
  - suffix: (string) The suffix, if any, for the NFS export path
  - id: (deprecated) The UUID of the CIFS share

Advanced attributes, if present, may also be returned for the share. See cifs.modifyShare for a listing of advanced attributes.

EXAMPLE
```
print clientHandle.cifs.getShare('current-vserver', 'testshare')
{'accessControl': 'posix', 'homeDir': 'no', 'shareName': 'testshare',
'rev': '740fd08b-fb7c-11e2-a02b-000c299a83be', 'export': '/', 'id': '
710a58e0-7c35-4cda-9950-bf74d9dcc334'}
```

cifs.getShareAcl

NAME
cifs.getShareAcl

SYNOPSIS
cifs.getShareAcl(vserverName, shareName) => shareAcl

DESCRIPTION
Returns the share level ACL (access control list) for a CIFS share.

PARAMETERS
- vserverName: (string) The name of the vserver
- shareName: (string) The name of the CIFS share

RETURNS
- shareAcl: An array of XML-RPC structs that contain the following name:value pairs:

  - name: (string) The name of a user or group, or an empty string if the name could not be determined.
  - sid: (string) The SID of a user or group, with SID elements represented as decimal.
  - type: (string) The type of ACE (access control entry) (either "ALLOW" or "DENY").
  - perm: (string) The permissions to allow or deny (one of "READ", "CHANGE", or "FULL").

EXAMPLE
```
print clientHandle.cifs.getShareAcl('vserver1', 'example')
[{'name': 'Everyone', 'sid' : 'S-1-1-0', 'type': 'ALLOW', 'perm': 'FULL'}]
```

NAME
cifs.isEnabled

SYNOPSIS
cifs.isEnabled(vserverName) => isEnabled

DESCRIPTION
Determines whether or not CIFS is enabled on a vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- isEnabled: (boolean) Whether CIFS is enabled (True) or disabled (False) on the vserver

EXAMPLE
print clientHandle.cifs.isEnabled('virtual1')
False

NAME
cifs.listShares

SYNOPSIS
cifs.listShares(vserverName) => array_of_structs

DESCRIPTION
Returns information about all CIFS shares on a vserver.

PARAMETERS
vserverName: (string) The name of the vserver

RETURNS
- array_of_structs: An array of XML-RPC structs that contain
the following name:value pairs:

- accessControl: (string) The access-control mechanism for the share, one of the following:
- 'posix' for POSIX mode bits
- 'nfsv4' for NFSv4 ACLs
- 'cifs' for CIFS ACLs
- suffix: (string) The suffix, if any, for the NFS export path
- shareName: (string) The name of the CIFS share
- rev: (deprecated) The revision number of the CIFS share
- export: (string) The NFS export that the CIFS share is making available to CIFS clients
- id: (deprecated) The UUID of the CIFS share
- homeDir: (string) Specifies whether the share is the home share ('yes')
or not ('no')

Advanced attributes, if present, may also be returned for each share. See cifs.modifyShare for a listing of advanced attributes.

EXAMPLE
```
print clientHandle.cifs.listShares('vserver1')
[{'accessControl': 'posix', 'suffix': '', 'shareName': 'testshare',
'rev': '740fd08b-fb7c-11e2-a02b-000c299a83be', 'export': '/', 'id':
'710a58e0-7c35-4cda-9950-bf74d9dcc334', 'homeDir': 'no'}]
```

NAME
cifs.modifyShare

SYNOPSIS
cifs.modifyShare(vserverName, shareName, newSettings) => status

DESCRIPTION
Changes the settings of an existing CIFS share.

PARAMETERS
- vserverName: (string) The name of the vserver on which the CIFS share is located

- shareName: (string) The name of the CIFS share that is to be modified

- newSettings: An XML-RPC struct containing the following name:value pairs for the share, which are one or more of the following advanced settings:

  - accessControl: (string) The access-control mechanism for the share, one of the following:
    - 'posix' for POSIX mode bits (the default)
    - 'nfsv4' for NFSv4 ACLs
    - 'cifs' for CIFS ACLs

  - browseable: (string) Whether a Windows client can browse to the share, depending on the access permissions of the share, either 'yes' or 'no'

  - inherit permissions:
    (string) Whether new directories created under the share will inherit the permissions of their parent directory, either 'yes' or 'no'

  - read only: (string) Whether the share is read-only, either 'no' or 'yes'. Setting this option to 'yes' can lead to faster performance if the data will not need to be changed.

  - create mask: (octal) The mask for the UNIX permissions for a newly created file. The default is 0744.

  - security mask: (octal) The UNIX permissions that are set on a file whose permissions are changed by a Windows NT client from the native Windows NT security dialog box. The default is 0777.

  - directory mask: (octal) The UNIX permissions of a directory that is created with DOS permissions. The default is 0755.

  - directory security mask:
    (octal) The UNIX permissions that are set on a directory whose permissions are changed by a Windows NT client from the native Windows NT security dialog box. The default is 0777.

  - force create mode:
    (octal) The minimum set of UNIX permissions for any file created by the Avere OS CIFS server. The default is 0000.

  - force security mode:
    (octal) The minimum set of UNIX permissions that

can be modified on a file whose permissions are changed by a Windows NT client from the native Windows NT security dialog box. The default is 07000.

- force directory mode:
  (octal) The minimum set of UNIX permissions for any directory created by the Avere OS CIFS server. The default is 0000.

- force directory security mode:
  (octal) The minimum set of UNIX permissions that can be modified on a directory whose permissions are changed by a Windows NT client from the native Windows NT security dialog box. The default is 0000.

- force user: (string) The UNIX username that is assigned as the default user for all users of the Avere OS CIFS server. There is no default value.

- force group: (string) The UNIX group name that is assigned as the default group for all users of the Avere OS CIFS server. There is no default value.

- hide unreadable: (string) This setting controls CIFS access-based enumeration which hides files and directories that a user can not access when listing directory contents. The default is 'no' (disabled). When set to 'yes' (enabled) this setting can have a performance impact due to access checks being made and this impact will be more dramatic for directories that contain large numbers of files and/or directories.

- strict locking: (string) The default value of 'yes' means that each read and write request performs a byte range lock check prior to executing the operation. A value of 'no' disables the lock check in order to improve performance. While the SMB specifications require the server side lock check, in practice it is safe to disable this check since clients perform the lock check prior to sending the read or write request.

- oplocks: (string) The default value of 'no' disables all oplock support. A value of 'yes' enables read/write oplocks.

- level2 oplocks: (string) The default value of 'yes' enables read-only oplocks when 'oplocks' are enabled. A value of 'no' disables read-only oplocks.

- read only optimized:
  (string) A value of 'yes' enables performance related options that are applicable to read-only shares. The default value is 'no'.

- guest ok: (string) A value of 'yes' allows users that are not present in the password database to be mapped to the guest user. The default value is 'no'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cifs.modifyShare('vserver1', 'vserver1-cifs',
  {'browseable': True})
success
```

cifs.modifyShareAce

NAME

cifs.modifyShareAce

SYNOPSIS

cifs.modifyShareAce(vserverName, shareName, shareAce)

DESCRIPTION

Modify the type and/or permission level for an existing ACE (access control entry) that is present in the share level ACL (access control list) for a CIFS share. The provided id member of the shareAce parameter must refer to an existing ACE in the share level ACL.

If the share level ACL does not contain an ACE for the specified user or group, then an error is returned.

PARAMETERS

- vserverName: (string) The name of the vserver.
- shareName: (string) The name of the CIFS share.
- shareAce: An XML-RPC struct that contains the following name:value pairs:

  - id: (string) The name or SID of a user or group, with SID elements represented as decimal. Names from a trusted domain must include the domain prefix (ex, "DOMAIN\UserOrGroup").
  - type: (string) The type of ACE (either "ALLOW" or "DENY").
  - perm: (string) The permissions to allow or deny (one of "READ", "CHANGE", or "FULL").

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.cifs.modifyShareAce('vserver1', 'example', {'id': 'juser', 'type': 'DENY', 'perm': 'CHANGE'})
success
```

NAME
cifs.removeLocalGroupMember

SYNOPSIS
cifs.removeLocalGroupMember(vserverName, groupName, memberId)

DESCRIPTION
Remove a user from a local group.

Supported groups:
* Administrators
* Run_As_Root

An attempt to remove a member that is not in the local group will return 'success'.

PARAMETERS
- vserverName: (string) The name of the vserver.
- groupName: (string) The name of the local group.
- memberId: (string) The name or SID (security ID) of the user or group to remove from the local group. SID elements must be represented as a decimal. Names must include the domain prefix (for example, "DOMAIN\User").

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cifs.removeLocalGroupMember('vserver1', 'Administrators', 'DOMAIN\juser')
success
print clientHandle.cifs.removeLocalGroupMember('vserver1', 'Administrators', 'S-1-1-0')
success
```

cifs.removeShare

NAME
cifs.removeShare

SYNOPSIS
cifs.removeShare(vserverName, shareName) => status

DESCRIPTION
Removes a CIFS share from a specified vserver.

NOTE: This method replaces the deprecated cifs.deleteShare method.

PARAMETERS
- vserverName: (string) The name of the vserver
- shareName: (string)The name of the CIFS share

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.cifs.removeShare('vserver1', 'vserver1-cifs')
success
```

cifs.removeShareAce

NAME

cifs.removeShareAce

SYNOPSIS

cifs.removeShareAce(vserverName, shareName, shareAce)

DESCRIPTION

Remove an ACE (access control entry) from the share level ACL (access control list) for a CIFS share.

PARAMETERS

- vserverName: (string) The name of the vserver
- shareName: (string) The name of the CIFS share

- shareAce: An XML-RPC struct that contains the following name:value pairs:

  - id: (string) The name or SID (security ID) of a user or group, with SID elements represented as decimal. Names from a trusted domain must include the domain prefix (for example, "DOMAIN\UserOrGroup").

  - type: (string) The type of ACE (either 'ALLOW' or 'DENY')

  - perm: (string) The permissions to allow or deny, one of 'READ, 'CHANGE', or 'FULL'

RETURNS

- status: (string) Either 'success' or a reason for failure. An attempt to remove an ACE that does not exist in the share level ACL will return 'success'.

EXAMPLE

```
print clientHandle.cifs.removeShareAce('vserver1', 'example', {'id': 'juser', 'type': 'DENY', 'perm': 'CHANGE'})
success
```

cifs.setLocalGroupMembers

NAME
cifs.setLocalGroupMembers

SYNOPSIS
cifs.setLocalGroupMembers(vserverName, groupName, groupMembers)

DESCRIPTION
Set the members of a local group.

Supported groups:
* Administrators
* Run_As_Root

PARAMETERS
- vserverName: (string) The name of the vserver.
- groupName: (string) The name of the local group.
- groupMembers: (array) A list of strings, where each string is the name or SID (security ID) of a user or group.
SID elements must be represented as a decimal. Names must include the domain prefix (for example, "DOMAIN\User").

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cifs.setLocalGroupMembers('vserver1', 'Administrators', ['DOMAIN\juser', 'S-1-1-0-100'])
success

NAME
cifs.setOptions

SYNOPSIS
cifs.setOptions(vserverName, newOptions) => status

DESCRIPTION
Changes the CIFS options set on a vserver.

PARAMETERS
- vserverName: (string) The name of the vserver that is to be modified
- newOptions: An XML-RPC struct containing the following name:value pair for the share:

- smb2: Specifies whether the CIFS service on the vserver will allow clients to use the SMB2 protocol ('yes' or 'no').

- guest_account: (string) CIFS shares with guest access enabled will map users that are not present in the password database to the user specified as the guest account. The CIFS service must be disabled and then enabled after changing this parameter. The specified account may be prefixed with the desired domain (e.g. DOMAIN\nobody). The default value is "nobody".

- native_identity: (string) A value of 'yes' indicates that CIFS users which do not have a UID and primary GID value available from the Directory Services User Name download can access junctions with CIFS ACL access control using the native Active Directory SID identity. A value of 'no' indicates that all CIFS users are required to have UID and primary GID values assigned. The default value is 'yes' for newly created vservers.

- read_only_optimized:
(string) A value of 'yes' enables performance related options for the entire vserver that are applicable to read-only shares. Additional performance changes are enabled when this type of optimization is enabled at the vserver level rather than the individual share level. However, all shares for the vserver are read only. The CIFS service must be disabled and then enabled after changing this parameter. The default value is 'no'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cifs.setOptions('gns', {'smb2': 'no', 'guest_account': 'aUser', 'native_identity': 'no', 'read_only_optimized': 'no'})
success

NAME
cifs.setShareAcl

SYNOPSIS
cifs.setShareAcl(vserverName, shareName, shareAcl)

DESCRIPTION
Sets the share level ACL (access control list) for a CIFS share.

PARAMETERS
- vserverName: (string) The name of the vserver

- shareName: (string) The name of the CIFS share

- shareAcl: An array of XML-RPC structs that contain the following name:value pairs:

  - id: (string) The name or SID (security ID) of a user or group, with the SID elements represented as a decimal. Names from a trusted domain must include the domain prefix (for example, "DOMAIN\UserOrGroup").

  - type: (string) The type of ACE (access control entry) (either 'ALLOW' or 'DENY')

  - perm: (string) The permissions to allow or deny, one of 'READ, 'CHANGE', or 'FULL'

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cifs.setShareAcl('vserver1', 'example', [{'id': 'Everyone',
  'type': 'ALLOW', 'perm': 'FULL'}])
success
```

NAME
cluster.abortActivity

SYNOPSIS
cluster.abortActivity(activityId) => status

DESCRIPTION
Sends an abort signal to the specified activity. Depending on the activity's status, it might or might not accept the signal.

PARAMETERS
- activityID: (string) The UUID of the activity. You can obtain a list of activities and their IDs by using the cluster.listActivities method.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.abortActivity('72b74bbb-
  fac0-11e2-911c-001b21459eeb')
success
```

NAME
cluster.activateAltImage

SYNOPSIS
cluster.activateAltImage([ha]) => status

DESCRIPTION
Switches the system software image to the current alternate image.

PARAMETERS
- [ha]: (boolean) Optional. If True, the upgrade uses the high-availability service to ensure minimum client disruption. The default is False.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.activateAltImage()
success
```

cluster.activities

NAME
cluster.activities

SYNOPSIS
cluster.activities() => array_of_structs

DESCRIPTION
Returns current cluster-wide activities.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain
the following name:value pairs:
  - process: (string) A description of the activity
  - status: (string) Whether the job is currently running or not
  - state: (string) One of the following:
  - 'inprogress'
  - 'success'
  - 'failure'
  - rev: (string) The revision UUID of the cluster
  - id: (string) The UUID of the activity
  - percent: (string) This element is returned if the 'state' is 'inprogress'

EXAMPLE
```
print clientHandle.cluster.activities()
[{'process': 'Starting Data Management job 5427',
 'status': 'management job running.',
 'state': 'success',
 'rev': '8183c8bc-fac0-11e2-911c-001b21459eeb',
 'id': '72b74bbb-fac0-11e2-911c-001b21459eeb'}
{'process': 'Creating Data Management job', 'status': 'Data Manage
ment job 5427 created', 'state': 'success', 'rev': '68d5e5da-fac0-
11e2-911c-001b21459eeb', 'id': '68d5e580-fac0-11e2-911c-001b21459e
eb'}
{'process': 'Starting Data Management job 5427', 'status': 'manage
ment job running.', 'state': 'success', 'rev': '875b3f6f-fac0-11e2
-911c-001b21459eeb', 'id': '875b3dc3-fac0-11e2-911c-001b2159eeb'}]
```

cluster.addClusterIPs

NAME

cluster.addClusterIPs

SYNOPSIS

cluster.addClusterIPs(ipRange) => status

DESCRIPTION

Adds IP addresses to a range of cluster IP addresses in an advanced-networking VLAN configuration.

- You can obtain information about existing cluster IP addresses from the 'name' parameter of the 'clusterIPs' struct returned by the cluster.get method.
- Advanced networking must be enabled, which can be done with the cluster.enableAdvancedNetworking method.

The new range's name is set by the system.

PARAMETERS

- ipRange: An array of XML-RPC structs that contain the following name:value pairs:

  - firstIP: (string) The first address in the range of cluster IP addresses
  - netmask: (string) The netmask for the VLAN
  - vlan: (string) The name of the VLAN
  - lastIP: (string) The last address in the range of cluster IP addresses

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.enableAdvancedNetworking()
success
print clientHandle.cluster.addClusterIPs([
 {'firstIP': '10.1.22.208', 'netmask': '255.255.224.0', 'lastIP':
 '10.1.22.215'}
 {'firstIP': '10.1.22.230', 'netmask': '255.255.224.0', 'lastIP':
 '10.1.22.235'}])
 6aad832e-fbae-11e2-9a30-000c299a83be
```

NAME
cluster.addLicense

SYNOPSIS
cluster.addLicense(licenseKey) => status

DESCRIPTION
Adds a license for additional features (such as FlashMove or FlashMirror) to the cluster.

PARAMETERS
- licenseKey: (string) The license number sent to you by Avere.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.cluster.addLicense('sIOaKQw00CDXxWip9wRYQ')
success

cluster.addNetwork

NAME

cluster.addNetwork

SYNOPSIS

cluster.addNetwork(networkName,[attrStruct]) => status

DESCRIPTION

Adds a network to the cluster.

NOTE: Before this method can return successfully, advanced networking must be enabled using the cluster.enableAdvancedNetworking method.

PARAMETERS

- networkName: (string) The name of the network. This should be between 1 and 64 characters; alphanumeric, "_", or "-"

- [attrStruct]: An optional XML-RPC struct that must include one or more of the following name:value pairs. Only the 'addressesPerNode' parameter is currently available:

  - addressesPerNode: (integer) The number of addresses per node to configure for this network. The default is '1'.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.enableAdvancedNetworking()
success
print clientHandle.cluster.addNetwork('newNetwork',
  {'addressesPerNode': '5'} )
success
```

cluster.addNetworkAddressRange

NAME

cluster.addNetworkAddressRange

SYNOPSIS

```
cluster.addNetworkAddressRange(networkName, range) => status
```

DESCRIPTION

Adds a new address range to a network.

PARAMETERS

- networkName: (string) The name of the network
- range: An XML-RPC struct that must include one or more of the following name:value pairs:
  - index: (string) The address range index to modify
  - firstIP: (string) The first IP address for the range
  - lastIP: (string) The last IP address for the range
  - netmask: (string) The netmask for the range
  - [vlan]: (string) Optional. The name of the VLAN for the range

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.addNetworkAddressRange('testnetwork', {'index':'0',
  'firstIP':'192.168.1.100','lastIP':'192.168.1.100','netmask':'255.255.248.0',
  'vlan':'vlan32'})
success
```

cluster.addNodeMgmtIPs

NAME

cluster.addNodeMgmtIPs

SYNOPSIS

cluster.addNodeMgmtIPs(ipRange) => status

DESCRIPTION

Adds IP addresses to a range of node-management IP addresses in an advanced-networking VLAN configuration.

- You can obtain information about existing node management IP addresses from the 'name' parameter of the 'nodeMgmtIPs' struct returned by the cluster.get method.
- Advanced networking must be enabled, which can be done with the cluster.enableAdvancedNetworking method.

The new range's name is set by the system.

PARAMETERS

- ipRange: An XML-RPC struct that must include one or more of the following name:value pairs:
  - firstIP: (string) The first address in the range of node-management IP addresses
  - netmask: (string) The netmask for the VLAN
  - vlan: (string) The name of the VLAN
  - lastIP: (string) The last address in the range of node-management IP addresses

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.enableAdvancedNetworking()
success
print clientHandle.cluster.addNodeMgmtIPs([
  {'firstIP': '10.1.22.208', 'netmask': '255.255.224.0', 'lastIP':
  '10.1.22.215'}
 {'firstIP': '10.1.22.230', 'netmask': '255.255.224.0', 'lastIP':
  '10.1.22.235'}])
6aad832e-fbae-11e2-9a30-000c299a83be
```

cluster.addSchedule

NAME
cluster.addSchedule

SYNOPSIS
cluster.addSchedule(schedName, clauseArray) => status

DESCRIPTION
Creates a schedule.

NOTE: This method replaces the deprecated cluster.createSchedule method.

PARAMETERS
- schedName: (string) The name of the schedule.

- clauseArray: (array) An array of XML-RPC structs containing the following name:value pairs for each schedule:

  - hours: (string) One of the following:
  - A comma-separated list of numbers from 0 (midnight) to 23 (11:00 p.m.)
  - * for all hours
  - minutes: (string) One of the following:
  - A comma-separated list of numbers that are multiples of 5 between 5 and 60
  - * for all five-minute intervals
  - days: (string) One of the following:
  - A comma-separated list of numbers from 0 (Sunday) to 6 (Saturday)
  - A comma-separated list of case-insensitive day names (Monday,Wednesday,Friday)
  - * for all days

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.cluster.addSchedule('xmlSched1', "[
  {'hours': '3','minutes': '10','days': '4'}, {'hours': '5',
  'minutes': '15','days': 'Thursday' }]"
success
```

cluster.addVLAN

NAME
cluster.addVLAN

SYNOPSIS
cluster.addVLAN(vlanName, tag, routerName, roles, [mtu]) => status

DESCRIPTION
Creates a VLAN with the specified parameters.

PARAMETERS
- vlanName: (string) The name of the VLAN
- tag: (integer) The VLAN's tag number, between 1 and 4094, inclusive
- routerName: (string) The IPv4 address of the VLAN's router

- roles: (string) The VLAN's role or roles, one of the following:
  - 'cluster' to make networking available to all parts of the cluster
  - 'client' to make networking available between clients
  - 'core_access' to make networking available to core filers
  - 'mgmt' to make networking available between management IPs

  Multiple values can be specified as a comma-separated list.

- [mtu]: (integer) Optional. The VLAN's maximum transmission unit (MTU) setting. The value of the MTU can range from 512 to 9000.

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.cluster.addVLAN('myTestLAN', 22, '10.1.0.76',
  'client,cluster')
1bc68229-fbad-11e2-9a30-000c299a83be
```

cluster.cancelUpgrade

NAME

cluster.cancelUpgrade

SYNOPSIS

cluster.cancelUpgrade() => status

DESCRIPTION

Cancels a software upgrade. If there is no upgrade in progress, the method has no effect.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.cancelUpgrade()
success
```

NAME

cluster.createProxyConfig

SYNOPSIS

cluster.createProxyConfig(name, options) => status

DESCRIPTION

Creates a proxy configuration that can be set on the cluster or a cloud core filer.

PARAMETERS

- name: (string) The administrative name for the proxy configuration.
- options: An XML-RPC struct that includes the following name:value pairs.
  - url: (string) The URL for the proxy server.
  - [user]: (string) The username needed to connect to the proxy server.
  - [password]: (string) The password needed to connect to the proxy server.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.createProxyConfig()
success
```

NAME
cluster.deleteProxyConfig

SYNOPSIS
cluster.deleteProxyConfig(name) => status

DESCRIPTION
Deletes a proxy configuration.

PARAMETERS
- name: (string) The name of the proxy configuration to delete.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.deleteProxyConfig()
success
```

NAME
cluster.disableHA

SYNOPSIS
cluster.disableHA() => status

DESCRIPTION
Disables high availability for the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.disableHA()
success
```

cluster.disableVMwareOptimization

NAME

cluster.disableVMwareOptimization

SYNOPSIS

cluster.disableVMwareOptimization() => status

DESCRIPTION

Disables VMware optimization for all vservers in the cluster.

WARNING: Turning off VMware optimization mode will cause all services in the cluster to be stopped. At this point the cluster will be reconfigured for general-purpose workloads and then the cluster services will be restarted.

The length of this cluster outage will be approximately 5 minutes.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.disableVMWareOptimization()
success
```

cluster.enableAdvancedNetworking

NAME

cluster.enableAdvancedNetworking

SYNOPSIS

cluster.enableAdvancedNetworking() => status

DESCRIPTION

Enables advanced networking on the cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.enableAdvancedNetworking()
67ff9e5a-fbab-11e2-9a30-000c299a83be
print clientHandle.cluster.enableAdvancedNetworking()
success
```

NAME
cluster.enableHA

SYNOPSIS
cluster.enableHA() => status

DESCRIPTION
Enables high availability for the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.cluster.enableHA()
success

cluster.enableVMwareOptimization

NAME

cluster.enableVMwareOptimization

SYNOPSIS

cluster.enableVMwareOptimization() => status

DESCRIPTION

Enable VMware optimization for all vservers in the cluster.

WARNING: Optimizing a cluster for VMware limits the cluster's use to VMware-only workloads. Do not enable VMware optimization and then use the cluster for more general-purpose workloads.

Turning on VMware optimization mode will cause all services in the cluster to be stopped. At this point the cluster will be reconfigured for VMware workloads and the cluster services will be restarted.

The length of this cluster outage will be approximately 5 minutes.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.enableVMWareOptimization()
success
```

cluster.get

NAME
cluster.get

SYNOPSIS
cluster.get() => clusterInfo

DESCRIPTION
Lists cluster attributes.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- clusterInfo: An XML-RPC struct containing the following name:value pairs. For some of the elements, if the parameter is not set, it will not be included in the struct.

  - [avereDataParameters]: When HA is enabled in the cluster, an XML-RPC struct that identifies the directory where the cluster stores its voting data. It includes the following name:value pairs:
    - dataExport: (string) The path to the export used by the cluster for HA
    - dataDir: (string) The name of the directory where the cluster writes the HA data; the default is '.avere'
    - corefiler | mass: (string) The name of the core filer containing the voting directory. Note that parameters containing "mass" are deprecated, and only present for backward compatibility. "Corefiler" should be used for all new applications.

  - ntpMulticast: (string) Determines whether NTP multicast is enabled for the cluster ('yes') or not ('no')

  - mgmtIP: An XML-RPC struct that must include one or more of the following name:value pairs:
    - IP: (string) The primary management IP address
    - netmask: (string) The netmask for the management network, if any
    - vlan: (string) The name of the VLAN, if any, containing the management IP
    - group: (string) The name of the port group, if any, to which management IP is assigned

  - alternateImage: (string) The version number of the alternate software image

  - advancedNetworking: (string) Whether advanced networking is 'disabled' or 'enabled'

  - VMwareOptimizationEnabled: (string) 'yes' if the cluster's use is limited to VMware-only workloads

  - proxy: (string) The name of the proxy configuration used by the cluster.
  - default_mtu: (integer) The cluster's default maximum transmission unit (MTU) setting
  - timezone: (string) The timezone set for the cluster
  - id: (string) The cluster's UUID
  - DNSserver: (string) The cluster's DNS server IP address
  - DNSdomain: (string) The cluster's fully qualified DNS domain name

- clusterIPs: An array of structs, each struct containing information about
a range of cluster IP addresses. If advanced networking is not enabled,
the 'netmask' and 'name' fields are not returned.
  - firstIP: (string) The first address in the range of management IP addresses
  - netmask: (string) The netmask for the management VLAN, if any
  - name: (string) The name of the IP address range for a VLAN
  - lastIP: (string) The last address in the range of management IP addresses
  - vlan: (string) The name of the VLAN, if any, to which the range is assigned
  - group: (string) The name of the port group, if any, to which the range is assigned

- separate_management_network:
(string) Whether the cluster uses a separate management network ('yes' or 'no')
to ensure that the e0a and e0b ports on the node are always used for management

- rev: (deprecated) The cluster's revision number
- default_router: (string) The IP address of the cluster's default router
- allowAllNodesToJoin:
(string) Whether unconfigured nodes on the network are permitted
to join the cluster automatically ('yes' or 'no'); the default is 'no'

- allowClusterDataManagerToUseCoreFiler:
(string) Whether the cluster data manager is permitted to use
the associated core filer for high availability ('yes' or 'no');
the default is 'no'

- DNSsearch: (string) The cluster's DNS search path
- activeImage: (string) The active software image version
- proxy: (string) The name of the proxy configuration used by the cluster.
- proxyuser: (deprecated) Userid to pass to the proxy server, if required. This value is now
determined
by the proxy configuration listed under the proxy attribute.
- proxyurl: (deprecated) The URL of the proxy server, if required. This value is now determined
by the proxy configuration listed under the proxy attribute.
- NTPservers: (string) A space-separated list of any NTP servers used by the cluster
- netmask: (string) The cluster's netmask
- ha: (string) Whether high availability is 'disabled' or 'enabled'

- [dirmgrDistributionState]:
(string) If local directories are enabled, this parameter indicates
the state of the cluster's local directory manager distribution,
one of the following:
- 'ok': the local directory managers do not need to be rebalanced
- 'rebalanceNeeded': the local directory managers need to be rebalanced
- 'rebalancing': the local directory managers are being
redistributed around the cluster

- useLinkAggregation:
(string) Whether the cluster uses link aggregation ('yes' or 'no')
- useLACP: (string) Whether the cluster uses dynamic LACP ('yes' or 'no'). You can
enable LACP only if link aggregation is also enabled.

- [static_routes]: (array) The static routes configured for the cluster, if any

- [ipmi]: If the IPMI cards on the cluster's nodes are configured on
a cluster-wide (as opposed to per-node) basis, an XML-RPC struct

containing the following name:value pairs for the cluster's IPMI settings.

Use the node.get method to obtain additional IPMI configuration values.

- mode: (string) Configuration mode for each IPMI card, one of the following:
- 'none': No configuration. This value clears the configuration on a cluster whose IPMI cards have previously been configured.
- 'static': A static IP address. NOTE: The remaining elements will only returned if the 'mode' parameter is set to 'static'.
- 'dhcp': A DHCP-assigned IP address
- firstIP: (string) The first IP address in the range of addresses allocated for the cluster's IPMI card. ('static' mode only)
- lastIP: (string) Last IP address in the range of addresses allocated for the cluster's IPMI card. ('static' mode only)
- netmask: (string) Netmask for the IPMI card. ('static' mode only)
- router: (string) Default router for the IPMI card. ('static' mode only)

- name: (string) The name of the cluster

- clusterIPNumPerNode:
(string) The number of IP addresses assigned to each node in the cluster

- nodeMgmtIPs: (struct | array_of_structs)
- If advanced networking is not enabled, an XML-RPC struct containing the following name:value pairs:
  - firstIP: (string) The first address in the range of node-management IP addresses
  - lastIP: (string) The last address in the range of node-management IP addresses

- If advanced networking is enabled, an array of structs, each of which contains the following name:value pairs:
  - firstIP: (string) The first address in the range of node-management IP addresses
  - netmask: (string) The netmask for the node-management VLAN
  - vlan: (string) The name of the node-management VLAN
  - lastIP: (string) The last address in the range of node-management IP addresses
- cloudAdminCredential:
(string) The cloud credential used to authenticate the cluster to the provider's (Azure, EC2, GCE, Swift) cloud APIs.
- nonMgmtNetmask: (string) The netmask for the non-management cluster network, in dotted notation. It is used when 'separate_management_network' is 'yes'. If unspecified, the default netmask is used.
- nonMgmtMtu: (integer) MTU for the non-management cluster network. If unspecified, the default MTU is used. In order to utilize a different MTU for the management and non-management networks, 'separate_management_network' must be set to 'yes'.
- fipsMode: (string) Displays whether FIPS is enabled or disabled on the cluster.
- internetVlan: The name of a VLAN that has external internet connectivity.
- ec2AdminCredential: (deprecated) This parameter is deprecated. Use cloudAdminCredential instead.

EXAMPLE

```
print clientHandle.cluster.get()
{'avereDataParameters': {'dataExport':'/vol/ha_voters','dataDir':'.avere
','mass':'grape'},'ntpMulticast':'no','mgmtIP':{'IP':'10.1.16.123','netm
ask':'255.255.224.0'},'alternateImage':'AvereOS_V3.1-1d25','advancedNetw
orking':'enabled','default_mtu':1000,'timezone':'America/New_York',
'id':'0acbba-971-112-83ae-009a83e','DNSserver':'10.0.12.40','DNSdomain':'company.com','clusterIPs':[{'
firstIP':'10.1.22.208','netmask':'255.255.224.0','name':'clusterIP1','la
stIP':'10.1.22.215'},{'firstIP':'10.1.16.124','netmask':'255.255.224.0',
```

'name':'clusterIP0','lastIP':'10.1.16.127'}],'separate_management_networ
k':'no','rev':'6b0c0-30b-1e3-8a16-000c99b','default_router':'10.1.0.76',
'allowAllNodesToJoin':'no','allowClusterDataManagerToUseCoreFiler':'yes'
,'DNSsearch':'company.com','activeImage':'AvereOS_V3.1.5-fe60c06',
'NTPservers':'ntp.company.com','netmask':'255.255.224.0'
,'ha':'enabled','useLinkAggregtion':'no','useLACP':'no','name':'LiGo_Clu
ster','clusterIPNumPerNode':2,'nodeMgmtIPs':[{'firstIP':'10.1.125.141','
netmask':'255.255.224.0','name':'nodeMgmtIP0','lastIP':'10.1.125.142','proxy':''}]}

NAME
cluster.getActivity

SYNOPSIS
cluster.getActivity(activityID) => activityStruct

DESCRIPTION
Returns information about the specified activity.

PARAMETERS
- activityID: (string) The UUID of the activity. You can obtain a list of activities and their IDs by using the cluster.listActivities method.

RETURNS
- activityStruct: An XML-RPC struct containing the following name:value pairs about the activity:

  - status: (string) What the activity is currently doing, or whether it is completed
  - lastUpdateTime: (string) If available, when the activity was last updated, in the format 'DDD mmm dd hh:mm:ss yyyy'
  - process: (string) Overall description of the activity
  - state: (string) One of the following:
    - 'inprogress'
    - 'success'
    - 'failure'
  - creationTime: (string) When the activity was created, in the format 'DDD mmm dd hh:mm:ss yyyy'
  - percent: (string) Percentage complete of the activity; returned only if 'state' is 'inprogress'
  - id: (string) The UUID of the activity

EXAMPLE
```
print clientHandle.cluster.getActivity('1f67af2f-090e-11e3-9979-002590208a54')
{'status': 'completed', 'lastUpdateTime': 'Mon Aug 19 16:30:18 2013', 'proce
ss': 'Login health check', 'creationTime': 'Mon Aug 19 16:30:17 2013', 'perc
ent': '100', 'state': 'success', 'id': '1f67af2f-090e-11e3-9979-002590208a54'}
```

cluster.getDataParameters

NAME

cluster.getDataParameters

SYNOPSIS

cluster.getDataParameters() => dataParamStruct

DESCRIPTION

DEPRECATED; use the cluster.getHADataParameters method instead.

cluster.getHADataParameters

NAME

cluster.getHADataParameters

SYNOPSIS

cluster.getHADataParameters() => dataParamStruct

DESCRIPTION

Returns information about the cluster's high-availability data repository.

NOTE: This method replaces the deprecated cluster.getDataParameters method.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- dataParamStruct: An XML-RPC struct that includes the following name:value pairs for the HA data repository:

  - dataExport: (string) The NFS export on which the data repository is located
  - dataDir: (string) The directory in which the data repository is located
  - corefiler: (string) If the data repository is located on a GNS-enabled vserver, the name of the core filer on which the data repository's vserver is located

EXAMPLE

```
print clientHandle.cluster.getHADataParameters()
{'dataExport': '/vol/ha_voters', 'dataDir': '.avere', 'corefiler': 'largeFiler'}
```

NAME

cluster.getNetwork

SYNOPSIS

cluster.getNetwork(networkName) => networkInfoStruct

DESCRIPTION

Returns a network configuration.

PARAMETERS

- networkName: (string) The name of the network

RETURNS

- networkInfoStruct:
  An XML-RPC struct containing the following name:value pairs
  about the network:

- addressesPerNode: (integer) The number of addresses per node to configure for this network

- [addressRangeN]:
  If applicable, an XML-RPC struct that contains the following
  name:value pairs describing the address range for the
  cluster network:

  - index: (string) The index number N of the range
  - firstIP: (string) The first IP address in the range
  - netmask: (string) The netmask for the range
  - lastIP: (string) The last IP address in the range
  - vlan: (string) If applicable, the VLAN for this range

- reserved: (boolean) If True, the network is reserved and cannot be modified
- <networkName>:
  (string) The name of the network

EXAMPLE

```
print clientHandle.cluster.getNetwork('myNetwork')
{'addressRange0': {'index': '0', 'firstIP': '10.1.16.124', 'netmask':
'255.255.224.0', 'lastIP': '10.1.16.127'}, 'addressRange1': {'index':
'1', 'firstIP':  '10.1.22.208', 'netmask': '255.255.224.0', 'lastIP':
'10.1.22.215'}, 'reserved': True, 'myNetwork': 'myNetwork'}
```

cluster.getStaticRoutes

NAME
cluster.getStaticRoutes

SYNOPSIS
cluster.getStaticRoutes(routerName) => array_of_structs

DESCRIPTION
Returns information about the static routes associated with the specified router.

PARAMETERS
- routerName: (string) The name or IP address of the router for which you want information

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:
  - destIP: (string) Destination IP of the static route
  - netmask: (string) Netmask of the static route
  - gateway: (string) Gateway of the static route

The method returns an error if advanced networking is not enabled on the cluster.

EXAMPLE
```
print clientHandle.cluster.getStaticRoutes('10.1.0.1')
[{'destIP': '10.0.8.0', 'netmask': '255.255.248.0', 'gateway': '10.1.0.76'}]
```

cluster.getVLAN

NAME
cluster.getVLAN

SYNOPSIS
cluster.getVLAN(name) => vlanStruct

DESCRIPTION
Returns information about the specified VLAN.

PARAMETERS
- name: (string) The name of the VLAN

RETURNS
- vlanStruct: An XML-RPC struct that contains the following name:value pairs about the VLAN:

  - name: (string) The name of the VLAN
  - roles: (string) The VLAN's role or roles, one of the following:
    - 'cluster' to make networking available to all parts of the cluster
    - 'client' to make networking available between clients
    - 'core_access' to make networking available to core filers
    - 'mgmt' to make networking available between management IPs
  - rev: (deprecated) The VLAN's revision number
  - [mtu]: (integer) Optional. The VLAN's maximum transmission unit (MTU) setting
  - tag: (integer) The VLAN's tag number, between 1 and 4094, inclusive.
  - router: (string) The IP address of the VLAN's router
  - id: (deprecated) The VLAN's ID

EXAMPLE
```
print clientHandle.luster.getVLAN('myTestLAN')
{'name': 'myTestLAN', 'roles': 'cluster',
'rev': '1bc68296-fbad-11e2-9a30-000c299a83be',
'tag': '22', 'router': '10.1.0.76',
'id': '1bc6824b-fbad-11e2-9a30-000c299a83be'}
```

cluster.isLicensed

NAME

cluster.isLicensed

SYNOPSIS

cluster.isLicensed(feature) => allowedFeature

DESCRIPTION

Tells whether a specified feature is licensed or not.

PARAMETERS

- feature: (string) The name of the feature, one of the following:
  - 'FlashMirror' for the mirroring feature, which implies 'FlashMove' and 'LocalDirectories'
  - 'FlashMove' for the moving feature, which implies 'LocalDirectories'
  - 'LocalDirectories'

RETURNS

- allowedFeature: (boolean) Whether the specified feature is licensed (True) or not (False)

EXAMPLE

```
print clientHandle.cluster.isLicensed('FlashMove')
True
```

cluster.listActivities

NAME
cluster.listActivities

SYNOPSIS
cluster.listActivities() => array_of_structs

DESCRIPTION
Returns information about cluster-wide activities.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - status: (string) What the activity is currently doing, or whether it is completed
  - lastUpdateTime: (string) If available, when the activity was last updated, in the format 'DDD mmm dd hh:mm:ss yyyy'
  - process: (string) Overall description of the activity
  - creationTime: (string) When the activity was created, in the format 'DDD mmm dd hh:mm:ss yyyy'
  - percent: (string) Percentage complete of the activity; returned only if 'state' is 'inprogress'
  - state: (string) One of the following:
    - 'inprogress'
    - 'success'
    - 'failure'
  - id: (string) The UUID of the activity

EXAMPLE
```
print clientHandle.cluster.listActivities()
[{'status': 'completed',
 'lastUpdateTime': 'Tue Aug  6 10:27:33 2013',
 'process': 'Login health check',
 'creationTime': 'Tue Aug  6 10:27:32 2013',
 'percent': '100',
 'state': 'success',
 'id': '52d84eb7-fea4-11e2-9eb8-001b21459eeb'},
{'status': 'writing out user data', 'lastUpdateTime': 'Tue Aug  6 10:36
:59 2013', 'process': 'Enabling local directory manager for MyFiler', '
creationTime': 'Tue Aug  6 00:02:08 2013', 'percent': '1', 'state': 'in
progress', 'id': 'f685510e-fe4c-11e2-9eb6-001b21459eeb'}, {'status': 'w
riting out user data', 'lastUpdateTime': 'Tue Aug  6 10:36:20 2013', 'p
rocess': 'Enabling local directory manager for MigrationMatrix_A', 'cre
ationTime': 'Mon Aug  5 19:40:29 2013', 'percent': '1', 'state': 'inpro
gress', 'id': '68ac7ab8-fe28-11e2-9eb6-001b21459eeb'}]
```

cluster.listCloudEndpoints

NAME
cluster.listCloudEndpoints

SYNOPSIS
cluster.listCloudEndpoints() => array_of_structs

DESCRIPTION
Returns storage cloud endpoints that are used by a cloud core filer
to connect to a cloud such as Amazon's s3 storage service.
The selected endpoint is added to the core filer during the creation
of the core filer.

Storage clouds that support multiple regions will return a list of
regions each of which will contain a list of endpoints.

Clouds that do not support regions will return a list of endpoints.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain
the following name:value pairs:

- name: (string) The cloud name
- title: (string) The cloud title
- version: (integer) The version number of this cloud storage definition
- addressStyle (string) Either 'host' for S3 virtual host style or 'path' for S3 path style
- type (string) Storage cloud type. Current value is S3.
- regions (array) A list of regions
- region (struct) An instance of a region
  - name: (string) Name of the region
  - title: (string) The title of the region
  - endpoints (string) The network address of the endpoint.
  - endpoint (string) A network address that connects to specified storage cloud.

EXAMPLE
```
print clientHandle.cluster.listCloudEndpoints
[{'name': 's3Amazon', 'title': 'Amazon Simple Storage Service (S3)',
'regions': [{'endpoints': ['s3-us-west-2.amazonaws.com'],
'name': 'us-west-2', 'title': 'US West (Oregon) Region'},
          {'endpoints': ['s3-us-west-1.amazonaws.com'],
'name': 'us-west-1', 'title': 'US West (Northern California) Region'},
          {'endpoints': ['s3-eu-west-1.amazonaws.com'],
'name': 'eu-west-1', 'title': 'EU (Ireland) Region'},
          {'endpoints': ['s3-ap-southeast-1.amazonaws.com'],
'name': 'ap-southeast-1', 'title': 'Asia Pacific (Singapore) Region'},
          {'endpoints': ['s3-ap-southeast-2.amazonaws.com'],
'name': 'ap-southeast-2', 'title': 'Asia Pacific (Sydney) Region'},
         {'endpoints': ['s3-ap-northeast-1.amazonaws.com'],
'name': 'ap-northeast-1', 'title': 'Asia Pacific (Tokyo) Region'},
         {'endpoints': [{'s3-sa-east-1.amazonaws.com'],
'name': 'sa-east-1', 'title': 'SouthAmerica (Sao Paulo) Region'}],
'version': '1', 'addressStyle': 'host', 'type': 'S3'}]
```

cluster.listLicenses

NAME

cluster.listLicenses

SYNOPSIS

cluster.listLicenses() => licenseStruct

DESCRIPTION

Lists licenses available to the current cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- licenseStruct: An XML-RPC struct that contains the following name:value pairs:

  - status: (string) One of the following:
    - 'no licensed features'
    - 'OK'
    - a reason why the cluster is not in compliance

  - cluserUUID: (string) The UUID of the cluster

  - features: (string) The available features purchased for the cluster, one of the following:
    - 'FlashMove'
    - 'FlashMirror'
    - 'LocalDirectories' (included with FlashMove and FlashMirror)

  - <idNumber>: (string) The license code available to the cluster
  - maxNodes: (string) Number of nodes for which the cluster's license is configured
  - desc_<idNumber>:
    (string) A summary of the features and maximum nodes the license covers
  - expire_<idNumber>:
    (integer) For temporary licenses, the expiration date of the license, given in epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970

EXAMPLE

```
print clientHandle.cluster.listLicenses()
{'status': 'OK', 'clusterUUID': '0acb6aba-9711-11e2-83ae-000c299a83be',
'features': 'FlashMove FlashMirror LocalDirectories', '1': 'sIOaKQwAgCD
XxWip9owRYQ==', 'maxNodes': '8', 'desc_1': 'FlashMove FlashMirror Local
Directories, 8 nodes'}
```

NAME

cluster.listNetworks

SYNOPSIS

cluster.listNetworks() => networkInfoStruct

DESCRIPTION

Lists the currently configured networks in the cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- networkInfoStruct:
  An XML-RPC struct containing the following name:value pairs about the network:

  - <networkName>:
    (string) The name of the network

  - [rangeStruct]: If applicable, an XML-RPC struct that contains the following name:value pairs describing the address range for each cluster network:
    - addressRange<N>:
      An XML-RPC struct that contains the following name:value pairs:

      - index: (string) The index of the range (<N>)
      - firstIP: (string) The first IP address in the range
      - netmask: (string) The netmask for the range
      - lastIP: (string) The last IP address in the range

    - addressesPerNode:
      (integer) The number of addresses per node for this network
    - reserved: (boolean) If True, the network is reserved and cannot be modified
    - name: (string) The name of the network

EXAMPLE

```
print clientHandle.cluster.listNetworks()
{'myNetwork':{'addressRange0':{'index':'0','firstIP':'10.1.20.131'
,'netmask': '255.255.224.0', 'lastIP': '10.1.20.134'}, 'addressesP
erNode': '2', 'reserved': True, 'name': 'myNetwork'}, 'newNetwork':
{'addressesPerNode': '5', 'reserved': False, 'name': 'newNetwork'}}
```

NAME

cluster.listProxyConfigs

SYNOPSIS

cluster.listProxyConfigs() => list

DESCRIPTION

Lists all the available proxy configurations in the cluster.

PARAMETERS

RETURNS

- list: An XML-RPC list containing individual XML-RPC structs for each proxy configuration with the following name:value pairs
  - name: The name of the proxy configuration.
  - url: The URL used to connect to the proxy server.
  - user: The username used to connect to the proxy server.

EXAMPLE

```
print clientHandle.cluster.listProxyConfigs()
[{'url': 'example.com', 'name': 'proxy2', 'user': 'ThisIsMyName'}, {'url': 'example.com',
'name': 'proxy5', 'user': ''}, {'url': 'example.com', 'name': 'proxy3', 'user': 'AlsoAName'}]
```

cluster.listSchedules

NAME
cluster.listSchedules

SYNOPSIS
cluster.listSchedules() => array_of_structs

DESCRIPTION
Returns an array of currently configured schedules.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

- clauseStruct: An XML-RPC struct of clauses that contains the following name:value pairs for each schedule:

  - clauseNum: (string) The name of the clause, given by the system, of the form 'clause#'

    - timeStruct: An XML-RPC struct with name:value pairs that contain the following schedule information:

      - hours: (string) One of the following:
      - A comma-separated list of numbers from 0 (midnight) to 23 (11:00 p.m.)
      - * for all hours
      - minutes: (string) One of the following:
      - A comma-separated list of numbers that are multiples of 5 between 5 and 60
      - * for all five-minute intervals
      - days: (string) One of the following:
      - A comma-separated list of numbers from 0 (Sunday) to 6 (Saturday)
      - A comma-separated list of case-insensitive day names (Monday,Wednesday,Friday)
      - * for all days

  - schedName: (string) The name of the schedule.

EXAMPLE
```
print clientHandle.cluster.listSchedules()
[{'clause2': {'hours':'5', 'minutes':'15', 'days':'4'},
 'clause1': {'hours':'3', 'minutes':'10', 'days':'4'}, 'name':'xmlSched1,'}
{'clause1': {'hours':'3', 'minutes':'45', 'days':'0'}, 'name':'hrllo'}
{'clause1': {'hours':'6', 'minutes':'20', 'days':'*'}, 'name':'second_ppl'}]
```

cluster.listVLANs

NAME
cluster.listVLANs

SYNOPSIS
cluster.listVLANs() => array_of_structs

DESCRIPTION
Lists currently configured VLANS and their attributes.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - name: (string) The name of the VLAN
  - roles: (string) The VLAN's role or roles, one of the following:
    - 'cluster' to make networking available to all parts of the cluster
    - 'client' to make networking available between clients
    - 'core_access' to make networking available to core filers
    - 'mgmt' to make networking available between management IPs
  - rev: (deprecated) The VLAN's revision number
  - [mtu]: (integer) Optional. The VLAN's maximum transmission unit (MTU) setting
  - tag: (integer) The VLAN's tag number, between 1 and 4094, inclusive.
  - router: (string) The IP address of the VLAN's router
  - id: (deprecated) The VLAN's ID

EXAMPLE
```
print clientHandle.cluster.listVLANs()
[{'name': 'myTestLAN', 'roles': 'cluster',
 'rev': '1bc68296-fbad-11e2-9a30-000c299a83be',
 'tag': '22', 'router': '10.1.0.76',
 'id': '1bc6824b-fbad-11e2-9a30-000c299a83be'}]
```

cluster.maxActiveAlertSeverity

NAME

cluster.maxActiveAlertSeverity

SYNOPSIS

cluster.maxActiveAlertSeverity() => struct

DESCRIPTION

Return the current maximum event and condition alert severity values and the cluster health.
These values are used to control the cluster health status displayed on the dashboard in the GUI.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- struct: A XML-RPC struct containing the following name:value pairs:

  - maxCondition: (string) The maximum active condition severity. The value can be one of: 'red', 'yellow', or 'green'.
  - maxEvent: (string) The maximum active event severity. The value can be one of: 'red', 'yellow', or 'green'.
  - clusterHealth: (string) Possible values are: 'OK', 'Warning', and 'Error'.

EXAMPLE

```
print clientHandle.cluster.maxActiveAlertSeverity()
    {'maxCondition': 'yellow', 'maxEvent': 'yellow', 'clusterHealth': 'Warning'}
```

NAME

cluster.modify

SYNOPSIS

cluster.modify(clusterAttributes) => status

DESCRIPTION

Sets one or more cluster attributes. The current values of these attributes can be obtained by using the cluster.get method.

- Use the cluster.modifyClusterIPs method to set the cluster's IP addresses.
- Use the cluster.setHADataParameters method to set the cluster's Avere data parameters for HA.

PARAMETERS

- clusterAttributes: An XML-RPC struct that must include one or more of the following name:value pairs:

  - mgmtIP: An XML-RPC struct that must include one or more of the following name:value pairs:

    - IP: (string) The primary management IP address
    - netmask: (string) (required for advanced networking) The netmask for the management network
    - vlan: (string) (required for advanced networking) The name of the VLAN, if any, containing the management IP
    - [group]: (string) Optional. The name of the port group for this range.

  - timezone: (string) The timezone set for the cluster
  - DNSserver: (string) The cluster's DNS server, specified as either a fully qualified domain name or an IP address
  - DNSdomain: (string) The cluster's fully qualified DNS domain name
  - DNSsearch: (string) The cluster's DNS search path
  - ntpMulticast: (string) Whether NTP multicast is enabled ('yes') or disabled ('no') on the cluster
  - NTPservers: (string) A space-separated list of NTP servers to be used by the cluster
  - netmask: (string) The cluster's netmask
  - default_router: (string) The cluster's default router
    NOTE: Use cluster.modifyVLAN() when Advanced Networking is enabled.
  - static_routes: (string) A list of static routes to be used by the cluster
  - default_mtu: (integer) The cluster's default maximum transmission unit (MTU) setting
    The value of the MTU can range from 512 to 9000.
    NOTE: Use cluster.modifyVLAN() when Advanced Networking is enabled.
  - separate_management_network:
    (string) Whether the cluster uses a separate management network ('yes' or 'no'). This is only available when none of the nodes in the cluster are virtual nodes.
  - useLinkAggregation:
    (string) Whether the cluster uses link aggregation ('yes' or 'no')
  - useLACP: (string) Whether the cluster uses dynamic LACP ('yes' or 'no'). You can enable LACP only if link aggregation is also enabled.
  - allowAllNodesToJoin:
    (string) Whether unconfigured nodes on the network are permitted to join the cluster automatically ('yes' or 'no'). The default is 'no'.
  - allowClusterDataManagerToUseCoreFiler:
    (string) Whether the cluster data manager is permitted to use the associated core filer for high availability ('yes' or 'no');

the default is 'no'.
- proxy: (string) The name of the proxy configuration that is used by the cluster. Proxy configs may be created using the cluster.createProxyConfig method.
- cloudAdminCredential: (string) The name of a cloud credential used to authenticate the cluster to the provider's (Azure, EC2, GCE, Swift) cloud APIs.
- nonMgmtNetmask: (string) The netmask for the non-management cluster network, in dotted notation. It is used when 'separate_management_network' is 'yes'. If unspecified, the default netmask is used.
- nonMgmtMtu: (integer) MTU for the non-management cluster network. If unspecified, the default MTU is used. In order to utilize a different MTU for the management and non-management networks, 'separate_management_network' must be set to 'yes'. The value of the MTU can range from 512 to 9000.
- fipsMode: (string) Disables or enables FIPS mode for the cluster. Accepted values are 'enable' and 'disable'. A cluster restart is required to apply this setting. Please run cluster.restartService to restart your cluster. When the cluster has FIPS 140-2 mode enabled, it can only use a limited number of secure cryptographic algorithms. If you have cloud core filers configured with HTTPS, make sure that it supports secure TLS ciphers. If you have cloud core filers using Amazon S3, the MD5 checksums to ensure data integrity of PUT requests cannot be used.
- internetVlan: The name of a VLAN that has external internet connectivity.
- ec2AdminCredential: (deprecated) Deprecated. Use the cloudAdminCredential instead.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.modify(
 {'allowClusterDataManagerToUseCoreFiler': 'yes'})
success
```

cluster.modifyCloudRegion

NAME
cluster.modifyCloudRegion

SYNOPSIS
cluster.modifyCloudRegion(regionname, options) => status

DESCRIPTION
Modifies or creates a cloud region's configuration.

PARAMETERS
- regionname: (string) The name of the region to add or modify
- options: An XML-RPC struct that includes one or more of the following name:value pairs.
  - [type]: (string) The vendor type of this region: only 'AWS' is supported. (required)
  - [s3Endpoint]: (string) The FQDN of the S3 endpoint in this region. (required for AWS regions)
  - [ec2Endpoint]: (string) The FQDN of the EC2 endpoint in this region. (required for AWS regions)
  - [force]: (string) Set to 'yes' if this is defining an 'isolated' region. Only set this when recommended by Avere Support.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.cluster.modifyCloudRegion('us-other-1', { 'type': 'AWS', 's3Endpoint': 's3.us-other-1.amazonaws.com', 'ec2Endpoint': 'ec2.us-other-1.amazonaws.com'})
success

cluster.modifyClusterIPNumPerNode

NAME

cluster.modifyClusterIPNumPerNode

SYNOPSIS

cluster.modifyClusterIPNumPerNode(number) => status

DESCRIPTION

Modifies the number of cluster IP addresses allocated to each node in the cluster.

PARAMETERS

- number: (integer) The number of cluster IP addresses per node

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.modifyClusterIPNumPerNode(2)
success
```

cluster.modifyClusterIPs

NAME
cluster.modifyClusterIPs

SYNOPSIS
cluster.modifyClusterIPs(rangeName) => status

DESCRIPTION
Modifies cluster IP addresses as specified by the 'rangeName' parameter.
You can obtain information about existing cluster IP addresses from
the 'name' parameter of the 'clusterIPs' struct returned by the cluster.get method.

PARAMETERS
- rangeName: An XML-RPC struct whose contents depend on whether advanced networking is enabled on the cluster.

  - If advanced networking is not enabled, the struct must contain the following name:value pairs:
    - firstIP: (string) The first address in the range of cluster IP addresses
    - lastIP: (string) The last address in the range of cluster IP addresses

  - If advanced networking is enabled, 'range' must include an array of one or more structs, each of which contains the following name:value pairs:

    - name: (string) The name of the range of cluster IP addresses. Use the cluster.get method to obtain this attribute.
    - firstIP: (string) The first address in the range of node-management IP addresses
    - netmask: (string) The netmask for the node-management VLAN
    - vlan: (string) The name of the node-management VLAN
    - lastIP: (string) The last address in the range of node-management IP addresses
    - [group]: (string) The port group name that the range is bound to

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.cluster.modifyClusterIPs([
 {'firstIP': '10.1.22.208', 'netmask': '255.255.224.0', 'lastIP':
 '10.1.22.215'}
6aad832e-fbae-11e2-9a30-000c299a83be
```

cluster.modifyDNS

NAME

cluster.modifyDNS

SYNOPSIS

cluster.modifyDNS(server1, server2, server3, domain, search) => status

DESCRIPTION

Modifies cluster DNS settings

PARAMETERS

- server1: (string) The IP address of the first DNS server
- server2: (string) The IP address of the second DNS server, or an empty string
- server3: (string) The IP address of the third DNS server, or an empty string
- domain: (string) The DNS domain name
- search: (string) The DNS search paths, a space-separated list of up to six domains to search, or an empty string

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.cluster.modifyDNS('192.168.100.1','','','example.com','')
success
```

cluster.modifyIPMI

NAME
cluster.modifyIPMI

SYNOPSIS
cluster.modifyIPMI(mode,[modeOptionalArguments]) => status

DESCRIPTION
Configures a cluster's IPMI cards.

PARAMETERS
- mode: (string) Configuration mode for the IPMI cards, one of the following:
  - 'none': No configuration. This value clears the configuration on a cluster whose IPMI cards have previously been configured.
  - 'static': A range of static IP addresses is used to assign IP addresses to the cluster's IPMI cards. The firstIP, lastIP, netmask, and router options must be set for this mode.
  - 'dhcp': DHCP-assigned IP addresses are assigned to the cluster's IPMI cards.

- [modeOptionalArguments]:
  An optional XML-RPC struct that contains the following name:value pairs:
  - firstIP: (string) The first address in the range of a cluster's IPMI IP addresses
  - lastIP: (string) The last address in the range of a cluster's IPMI IP addresses
  - netmask: (string) Netmask for the IPMI card
  - router: (string) The default router address used by the IPMI card

  NOTE: This parameter is applied only if the 'mode' parameter is set to 'static'. It has no effect if 'mode' is specified as 'none' or 'dhcp'.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.modifyIPMI('static', {'address': '10.12.0.124',
  'netmask': '255.255.244.0', 'router': '10.1.0.77'})
success
```

cluster.modifyNTP

NAME
cluster.modifyNTP

SYNOPSIS
cluster.modifyNTP(server1, server2, server3) => status

DESCRIPTION
Modifies cluster Network Time Protocol (NTP) settings

PARAMETERS
- server1: (string) The IP address or hostname of the first NTP server
- server2: (string) The IP address of hostname of the second NTP server, or an empty string
- server3: (string) The IP address of hostname of the third NTP server, or an empty string

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.cluster.modifyNTP('ntp.example.com')
'success'

NAME

cluster.modifyNetwork

SYNOPSIS

cluster.modifyNetwork(networkName, networkAttributes) => status

DESCRIPTION

Modifies the name or number of addresses per node on an existing network in the cluster.

- networkName: (string) The current name of the network

- networkAttributes: An XML-RPC struct that must include one or more of the following name:value pairs:

  - [addressesPerNode]: (string) Optional. The number of addresses per node for this network. This parameter must be included if 'name' is not included.
  - [name]: (string) Optional. The name of the network. This parameter must be included if 'addressesPerNode' is not included.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.modifyNetwork('testnetwork',
  {'addressesPerNode':'2', 'name':'newNetworkName'})
success
```

cluster.modifyNetworkAddressRange

NAME

cluster.modifyNetworkAddressRange

SYNOPSIS

cluster.modifyNetworkAddressRange(networkName, newRange) => status

DESCRIPTION

Modifies an existing address range. If the modification would cause the cluster to have fewer addresses in this network than nodes, the method will return an error.

PARAMETERS

- networkName: (string) The name of the network

- newRange: An XML-RPC struct that must include one or more of the following name:value pairs:
  - index: (string) The address range index to modify
  - firstIP: (string) The first IP address for the range
  - lastIP: (string) The last IP address for the range
  - netmask: (string) The netmask for the range
  - [vlan]: (string) Optional. The updated vlan name for the range
  - [group]: (string) The updated port group name for the range

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.modifyNetworkAddressRange('testnetwork',
  {'index':'0','firstIP':'192.168.1.100','lastIP':'192.168.1.100',
  'netmask':'255.255.248.0','vlan':'vlan32'})
success
```

cluster.modifyNodeMgmtIPs

NAME

cluster.modifyNodeMgmtIPs

SYNOPSIS

cluster.modifyNodeMgmtIPs(rangeStruct) => status

DESCRIPTION

Modifies a range of node-management IP addresses in an advanced-networking configuration. You can obtain information about existing node management IP addresses from the 'name' parameter of the 'nodeMgmtIPs' struct returned by the cluster.get method.

PARAMETERS

- rangeStruct: An XML-RPC struct that must include the 'name' parameter, and one or more of the remaining name:value pairs:

  - name: (string) The name of the range of node-management IP addresses
  - firstIP: (string) The first address in the range of node-management IP addresses
  - netmask: (string) The netmask for the VLAN
  - [vlan]: (string) Optional. The name of the VLAN
  - lastIP: (string) The last address in the range of node-management IP addresses
  - [group]: (string) Optional. The name of the port group for this range.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.modifyNodeMgmtIPs({'name': 'nodeMgmtIP0',
  'lastIP': '10.1.125.142', 'group': 'my_very_own_group'})
success
```

cluster.modifyProxyConfig

NAME

cluster.modifyProxyConfig

SYNOPSIS

cluster.modifyProxyConfig(name, options) => status

DESCRIPTION

Modifies an existing proxy configuration.

PARAMETERS

- name: (string) The administrative name for the proxy configuration.
- options: An XML-RPC struct that includes one or more of the following optional name:value pairs.
  - [name]: (string) The new name of the proxy configuration.
  - [url]: (string) The URL for the proxy server.
  - [user]: (string) The username needed to connect to the proxy server.
  - [password]: (string) The password needed to connect to the proxy server.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.modifyProxyConfig()
success
```

cluster.modifySchedule

NAME
cluster.modifySchedule

SYNOPSIS
cluster.modifySchedule(scheduleName, clauseStruct) => status

DESCRIPTION
Modifies the specified schedule.

PARAMETERS
- scheduleName: (string) The name of the schedule

- clauseStruct: An XML-RPC struct of clauses that contains the following name:value pairs for each schedule:

  - clauseNum: (string) The name of the clause, given by the system, of the form 'clause#'

  - timeStruct: An XML-RPC struct with name:value pairs that contain the following schedule information:

    - hours: (string) One of the following:
    - A comma-separated list of numbers from 0 (midnight) to 23 (11:00 p.m.)
    - * for all hours
    - minutes: (string) One of the following:
    - A comma-separated list of numbers that are multiples of 5 between 5 and 60
    - * for all five-minute intervals
    - days: (string) One of the following:
    - A comma-separated list of numbers from 0 (Sunday) to 6 (Saturday)
    - A comma-separated list of case-insensitive day names (Monday,Wednesday,Friday)
    - * for all days

To add a new clause to an existing schedule, include it in the 'clauses' parameter of this method.

To delete a clause from an existing schedule, specify {'clause#':'' } in the clauses parameter of this method, where # is the number of the clause to be deleted. Clause numbers can be obtained by using the cluster.listSchedules method.

Schedules can not be renamed.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cluster.modifySchedule("weekday_evenings",
  {'clause1': {'hours': '5', 'minutes': '10', 'days': '*'}, 'clause2':''})
success
```

cluster.modifyStaticRoutes

NAME

cluster.modifyStaticRoutes

SYNOPSIS

cluster.modifyStaticRoutes(routerName, array_of_structs) => status

DESCRIPTION

Modifies static routes for the given router.

PARAMETERS

- routerName: (string) The name or IP address of the router

- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - destIP: (string) Destination IP of the static route
  - netmask: (string) Netmask of the static route
  - gateway: (string) Gateway of the static route

  You can clear a router's static routes by specifying an empty struct as the value of the 'routes_struct' parameter.

  The method returns an error if advanced networking is not enabled on the cluster.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.modifyStaticRoutes("10.1.0.1", "[{'destIP':
  '192.168.1.2','netmask':'255.255.255.255', 'gateway':'192.168.1.1'}]"
success
```

cluster.modifyVLAN

NAME
cluster.modifyVLAN

SYNOPSIS
cluster.modifyVLAN(vlanStruct) => status

DESCRIPTION
Changes the attributes for a specified VLAN.

NOTE: Attributes that are not listed by the cluster.getVLAN method cannot be changed by using the XML-RPC API.

PARAMETERS
- vlanStruct: An XML-RPC struct that contains the following name:value pairs for the VLAN:

  - name: (string) The name of the VLAN
  - router: (string) The IPv4 address of the VLAN's router
  - roles: The VLAN's role or roles, one of the following:
    - 'cluster', when the network is used for node-to-node communication
    - 'client', when the network is used for client-to-cluster communication
    - 'core_access' to make networking available to core filers
    - 'mgmt', when the network is used for administrative access (through the GUI or XML-RPC) to the cluster

    Multiple values can be specified as a comma-separated list.

  - [mtu]: (integer) Optional. The VLAN's maximum transmission unit (MTU) setting
    The value of the MTU can range from 512 to 9000.

To modify the cluster default VLAN, you must specify:
- id: (integer) Must be zero (0).
- name: (string) 'default'
- router: (string) As above
- [mtu]: (integer) Optional, as above.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.modifyVLAN({'name': 'myTestLAN',
  'router': '10.1.0.76', 'roles':'client,cluster'})
success
```

cluster.moveHADataParameters

NAME

cluster.moveHADataParameters

SYNOPSIS

cluster.moveHADataParameters(newFilerName, dataExport, dataDir) => status

DESCRIPTION

Moves the HA data repository to a different core filer.

NOTE: If the current parameters are not set, this will set the parameters in the same way as the cluster.setHADataParameters method.

PARAMETERS

- newFilerName: (string) The name of the core filer to which the HA data repository is to be moved. This parameter must be specified for a GNS-enabled vserver, but is optional for a simple-namespace vserver.
- dataExport: (string) The name of the NFS export to which the HA data repository is to be moved
- dataDir: (string) The directory on the NFS export to which the HA data repository is to be moved

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.moveHADataParameters('upgradeFiler',
  '/vol/myhome', '.avere')
success
```

cluster.powerdown

NAME

cluster.powerdown

SYNOPSIS

cluster.powerdown() => status

DESCRIPTION

Powers down every node in the cluster. Client access is terminated until power is restored. No committed data is lost.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.powerdown()
success
```

NAME
cluster.reboot

SYNOPSIS
cluster.reboot([ha]) => status

DESCRIPTION
Reboots every node in the cluster. No committed data is lost.

PARAMETERS
- [ha]: (boolean) Optional. Whether the operation should the high-availability service to ensure minimum client disruption (True) or not (False)

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.reboot(True)
success
```

NAME
cluster.removeClusterIPs

SYNOPSIS
cluster.removeClusterIPs(rangeName) => status

DESCRIPTION
Removes a range of cluster IP addresses from an advanced-networking configuration.
You can obtain information about existing cluster IP addresses from
the 'name' parameter of the 'clusterIPs' struct returned by the cluster.get method.

PARAMETERS
- rangeName: (string) The name of the range of node-management IP addresses

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason
for failure. If the activity is not complete, the activity UUID, which
can be used as input for the cluster.getActivity and cluster.abortActivity
methods.

EXAMPLE
print clientHandle.cluster.removeClusterIPs('cluster_range')
success

cluster.removeLicense

NAME
cluster.removeLicense

SYNOPSIS
cluster.removeLicense(license) => status

DESCRIPTION
Removes a license for a feature, such as FlashMove or FlashMirror, from the cluster.

PARAMETERS
- <idNumber>: (string) The license code currently used by the cluster. You can obtain this using the cluster.listLicences method, and looking at the 'id_number' parameter that is returned, just before the description field.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.cluster.removeLicense('sIOaKQwAKKKxWip9owRYQ=')
success
```

NAME

cluster.removeNetwork

SYNOPSIS

cluster.removeNetwork(networkName) => status

DESCRIPTION

Removes a network, if the network is no longer in use.

If an internal object, such as a core filer, is using the network, use the corefiler.unsetNetwork or corefiler.setNetwork methods to configure the core filer to use a different network.

PARAMETERS

- networkName: (string) The name of the network to remove

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.cluster.removeNetwork('testnetwork')
success
```

cluster.removeNetworkAddressRange

NAME

cluster.removeNetworkAddressRange

SYNOPSIS

cluster.removeNetworkAddressRange(networkName, index) => status

DESCRIPTION

Removes an existing network address range from a network. If the removal would cause the cluster to have fewer addresses in this network than nodes, the method will return an error.

Use the cluster.listNetworks method to find the address range index.

PARAMETERS

- networkName: (string) The name of the network from which to remove the address range
- index: (string) The index of the address range to remove

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.cluster.removeNetworkAddressRange('newNetwork', '0')
success
```

cluster.removeNodeMgmtIPs

NAME

cluster.removeNodeMgmtIPs

SYNOPSIS

cluster.removeNodeMgmtIPs(rangeName) => status

DESCRIPTION

Removes a range of node management IP addresses from an advanced-networking configuration. You can obtain information about existing node management IP addresses from the 'name' parameter of the 'nodeMgmtIPs' struct returned by the cluster.get method.

PARAMETERS

- rangeName: (string) The name of the range of node-management IP addresses

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.cluster.removeNodeMgmtIPs('management_range')
success
```

cluster.removeSchedule

NAME
cluster.removeSchedule

SYNOPSIS
cluster.removeSchedule(schedName) => status

DESCRIPTION
Removes the specified schedule. Use the cluster.listSchedules method to find the value of the 'name' parameter.

NOTE: This method replaces the deprecated cluster.deleteSchedule method.

PARAMETERS
- schedName: (string) The name of the schedule

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.removeSchedule('xmlSched1')
success
```

NAME
cluster.removeVLAN

SYNOPSIS
cluster.removeVLAN(vlanName) => status

DESCRIPTION
Removes the specified VLAN.

PARAMETERS
- vlanName: (string) The name of the VLAN

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.cluster.removeVLAN('virtual_0')
success
```

NAME
cluster.rename

SYNOPSIS
cluster.rename(newname) => status

DESCRIPTION
Renames the cluster to the specified new name.

PARAMETERS
- newname: (string) The name with which to replace the current cluster's name

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.cluster.rename('newest')
success

cluster.restartService

NAME
cluster.restartService

SYNOPSIS
cluster.restartService([ha]) => status

DESCRIPTION
Restarts all services on every node in the cluster. No committed data is lost.

PARAMETERS
- [ha]: (boolean) Optional. If True, the operation uses the high-availability service to ensure minimum client disruption.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.cluster.restartService(True)
success
```

NAME

cluster.setHADataParameters

SYNOPSIS

cluster.setHADataParameters(filerName, dataExport, dataDir) => status

DESCRIPTION

Sets the Avere data parameters for use by the HA service. The action fails if these parameters are already set. Use the cluster.getHADataParameters method to determine whether they are already set.

PARAMETERS

- filerName: (string) The name of the core filer associated with the vserver. This parameter is optional if GNS is disabled for the vserver, and is required if GNS is enabled.
- dataExport: (string) The export where the cluster will store internal information about HA states
- dataDir: (string) The subdirectory on the export where the internal information will be stored

RETURNS

- status: (string) Either 'success' or a reason for failure

NOTE: This method returns a failure if these parameters are already set. (Use the cluster.getHADataParameters method to determine whether they are already set.)

EXAMPLE

```
print clientHandle.cluster.setHADataParameters('/vol/juser', '.avere', 'grape')
success
```

cluster.upgrade

NAME

cluster.upgrade

SYNOPSIS

cluster.upgrade(url, [options]) => status

DESCRIPTION

Downloads new system software from the specified url. The new software image is listed as the alternate image and is not activated automatically. You can activate the alternate image using the cluster.activateAltImage method, or the user interface.

PARAMETERS

- url: (string) The location of the new software image
- [options]: An optional XML-RPC struct that can contain any of the following name:value pairs to be changed:

  - username: (string) Username for the download server, if required
  - password: (string) Password for the download server, if required
  - proxyurl: (string) The URL of a proxy server to use for software downloads and support uploads, if direct connectivity from the cluster to the primary server is not available
  - proxyuser: (string) Userid to pass to the proxy server, if required
  - proxy_pass: (string) The password associated with proxyuser, if required

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.cluster.upgrade('http://web/download/3.1.0/avere.pkg')
success
```

NAME

cluster.upgradeSetGUIDefaults

SYNOPSIS

cluster.upgradeSetGUIDefaults([args]) => defaults

DESCRIPTION

Sets/retrieves default values for the GUI software upgrade function. The return and (optional) argument dictionaries may
contain the 'url' key whose value is the default download URL as a string.

PARAMETERS

A optional dictionary that may contain the 'url' key whose value will replace the current default download URL.

RETURNS

A dictionary that may contain the 'url' key whose value is the current default download URL displayed by the GUI.

EXAMPLE

cluster.upgradeStatus

NAME
cluster.upgradeStatus

SYNOPSIS
cluster.upgradeStatus() => upgradeStatus

DESCRIPTION
Returns the status of a system software upgrade.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- upgradeStatus: An XML-RPC struct that contains the following name:value pairs:
  - status: (string) Overall upgrade status for the cluster, including whether it has started, what actions it is performing, and whether the upgrade is complete.
  - allowActivate: (boolean) Whether the cluster is allowed to perform alternate image activation ('True' or 'False')
  - allowDownload: (boolean) Whether the cluster is allowed to perform upgrades ('True' or 'False')
  - nodeStatus: An XML-RPC struct with name:value pairs that contain each node's current upgrade status:
    - <nodeName>: (string) Name of the node
    - <status>: (string) The status of the node's upgrade, whether it has started, is in progress, or has finished.

EXAMPLE
```
print clientHandle.cluster.upgradeStatus()
{'status': 'FAILED: Cannot proceed with restart, the following nodes
are not responding and may be down:  LiGo (node1f21)', 'allowActivat
e': True, 'allowDownload': True, 'nodeStatus': {'LiGo1': 'Download%2
0AvereOS_V3.0.0.2-c3a431e%20complete', 'LiGo': 'Download%20AvereOS_V
3.0.0.2-c3a431e%20complete'}}
```

NAME
cluster.upgradeVersionUnjoined

SYNOPSIS
cluster.upgradeVersionUnjoined(nodeName) => status

DESCRIPTION
Upgrades the Avere OS on the specified node without joining the cluster.

PARAMETERS
- nodeName: (string) The name of the node to upgrade

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.cluster.upgradeVersionUnjoined('newestNode')
success

corefiler.activateMasterKey

NAME

corefiler.activateMasterKey

SYNOPSIS

corefiler.activateMasterKey(filerName, keyID, [recoveryFile]) => status

DESCRIPTION

Activates the most recently generated master key for a cloud core filer, using the parameters generated by the corefiler.generateMasterKey method.

NOTE: If you attempt to validate any master key besides the most recent one, a failure error will be returned.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer
- keyID: (string) The ID of the key to activate
- [recoveryFile]: (string) Optional. Required if keyMgmtType is simpleKey, the master key, represented as a base64-encoded string. Disallowed if keyMgmtType is kmip.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

The first of the following Python examples provides the keyID and recoveryFile directly. The second example assumes that 'result' was generated using the corefiler.generateMasterKey method:

```
print clientHandle.corefiler.activateMasterKey('cloudFiler1',
  '10000033000000002', 'U2FsdGVkX1...EA\n')
success

print clientHandle.corefiler.activateMasterKey('cloudFiler1',
  result['keyId'], result['recoveryFile'])
success
```

corefiler.create

NAME

corefiler.create

SYNOPSIS

corefiler.create(filerName, networkName, [ignoreWarning], [settings]) => status

DESCRIPTION

Creates a core filer configuration. The configuration is created with a read-only caching policy and core filer verification set to 30 seconds by default. Alternatively, you can supply a cache policy name using the cachePolicy option, or use the corefiler.setCachePolicy method change the cache policy at a later time.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- networkName: (string) The primary IP address or fully qualified domain name of the core filer. This may also be a space-separated list of IP addresses or domain names, where subsequent network names are used in advanced networking configurations.
- [ignoreWarning]: (boolean) Optional. Whether warnings are displayed during core filer creation (True) or not (False - the default)

- [settings]: If this value is set, the ignoreWarning parameter must also be supplied. An optional XML-RPC struct that contains the following name:value pairs:

  - extractFsidFromFilehandle: (string) A value of "yes" enables extracting the fsid from the filehandle for this core filer. The default is "no". Only change this option from the default if the core filer does not return persistent FSID values in NFS attributes and the core filer does not provide a mechanism to assign persistent FSIDs.
  - localDirectories: (deprecated) Do not supply this parameter. You may enable or disable this feature through the cache policy assigned to the filer instead.

  - cachePolicy: (string) The name of the cache policy to use, or "default". The cache policy is set to "Clients Bypassing the Cluster" if no cache policy is provided.

  - filerClass: (string) One of the following predefined values that describes this core filer. If not set explicitly here, will default to "Other".
    Acceptable NAS values:
    NetappNonClustered
    NetappClustered
    HitachiHNAS
    EmcIsilon
    Other

  - [filerNetwork]: (string) Optional. The name of the network to use for communication. The default network is the cluster network.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.corefiler.create('gentoo','10.1.16.222',False,{'cachePolicy':'Read Caching'})
success
```

NAME
corefiler.createCloudFiler

SYNOPSIS
corefiler.createCloudFiler(cloudfilerName, filerOptions) => status

DESCRIPTION
Creates a cloud filer configuration, with a read-write caching policy.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- cloudfilerName: (string) The name for the new cloud filer.

- filerOptions: An XML-RPC struct that contains the following name:value pairs:

  - cachePolicyName (string) The name of the cache policy to use. If no name is supplied, then Full Caching is used by default. The cache policy provided is validated to make sure it is cloud-compatible.

  - cloudType: (string) The type of the cloud filer.
  - [s3Type]: (string) Optional. The type of S3 service provider, only necessary if the cloudType is 's3'. Valid values are 'Amazon' (the default), 'Google', 'Amplidata', 'Cleversafe', 'SwiftStack', and 'ECS'.
  - [force]: (boolean) Optional. Whether warnings are displayed during core filer creation (False - the default) or not (True)
  - bucket: (string) The name of the bucket. This parameter is only required if the cloud type is 's3'. For non-commercial Amazon regions a format of 'bucket':'region' may need to be used.
  - serverName: (string) The IP address or fully-qualified domain name of the cloud service provider. This is optional for Amazon S3 cloud filers, as it is automatically determined by querying the bucket's location. If a serverName is specified, it is used only to send the location query. For Cleversafe, Amplidata, and other cloud filers, the serverName parameter is mandatory.
  - cloudCredential: (string) The cloud credential name. This parameter is only required if the cloud type is 's3'.
  - [port]: (string) Optional. The port number(s) used by the cloud filer. For Amazon S3 cloud filers, it can be one of the following:
    - '443' for https (the default)
    - '80' for http
    For some cloud filers, other port numbers or a space-separated list of ports can be specified. A range of ports (maximum of 8 ports) can also be specified using a hyphen ('-') symbol. Some examples are:
    - '7070 7071 7072 7073'
    - '7070-7073'
    This parameter is only required if the cloud type is 's3'.
  - [https]: (string) Optional. Determines whether the cloud filer uses https for security, either 'yes' (the default) or 'no'.
  - [pathStyle]: (string) Optional. Set to 'yes' if the cloud filer uses path-style requests. The default is 'no'.
  - [cryptoMode]: (string) Optional. The encryption mode, one of the following: This parameter is only required if the cloud type is 's3'.
    - 'DISABLED' (the default)
    - 'CBC-AES-256-HMAC-SHA-512'
  - [keyMgmtType]: (string) Optional. If the encryption mode is on, one of the following:

- 'simpleKey' (the default): store all keks on local disk
- 'kmip': store all keks on a specified kmip server

- [kmipServer]: (string) Optional. If the encryption mode is on and keyMgmtType is kmip, this is the name of the kmip server.
- [compressMode]: (string) Optional. The compression mode, one of the following:
This parameter is only required if the cloud type is 's3'.
- 'DISABLED'
- 'LZ4' (the default)
- 'LZ4HC (deprecated)'
- [sslVerifyMode]: (string) Optional. The certification verification used by the cluster to verify the X.509 certificate
returned by the cloud filer for identification. The options are:

- 'OCSP_CRL' use OCSP to verify the certificate. If the connection to the
OCSP responder fails fall back to CRL.
- 'OCSP' use OCSP to verify that the certificate is valid.
- 'CRL' use CRL to inspect and verify that the certificate is valid.
- 'DISABLED' (DANGEROUS) do not inspect or validate the certificate, and trust it blindly.

The default value for Amazon S3 cloud filers is 'OCSP_CRL'. The default value for all other cloud filer types will be 'DISABLED'.
- [proxy]: (string) Optional. The name of a proxy configuration to use when verifying the cloud filer certificate and accessing the cloud filer storage provider bucket.
- [nearline]: (string) Optional. Set to 'yes' if this cloud filer will be connected to a Google Cloud Storage Nearline bucket.
WARNING: connecting a cloud filer to nearline storage can result in excessive charges due to filesystem
directory metadata updates. To reduce this risk choose a cache policy for this cloud filer with a writeback
time of two hours or more.
- [bucketContents]: (string) Optional. This variable indicates whether the bucket assigned to this cloud filer contains
preexisting Avere Systems cloud filer data. The possible values are: 'empty', or 'used'.
The default value is 'empty'.
- [filerNetwork]: (string) Optional. The name of the network to use for communication. The default network
is the cluster network.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason
for failure. If the activity is not complete, the activity UUID, which
can be used as input for the cluster.getActivity and cluster.abortActivity
methods.

EXAMPLE

```
print clientHandle.corefiler.createCloudFiler('blueAngel', {'cloudType':'s3', 'force':True,
 'bucket':'consumerItems', 'cloudCredential':'cloudCred', 'cryptoMode':'CBC-AES-256-HMAC-SHA-512',
 'compressMode':'LZ4', 'cachePolicyName':'Full Caching'})
48a29694-311c-11e3-b9cb-000c293a3789
```

corefiler.createCredential

NAME
corefiler.createCredential

SYNOPSIS
corefiler.createCredential(credentialName, credentialType, attrsStruct) => status

DESCRIPTION
Creates a new cloud credential.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- credentialName: (string) The name for the new credential
- credentialType: (string) The type of cloud filer for which the credential is intended.
  The 's3-sts' type is available for situations where Amazon's STS service is being used to obtain security tokens. In this case an external process must periodically update the credential's security token.

- attrsStruct: An XML-RPC struct containing the following name:value pairs (see below for additional type-specific values):
  - type: (string) One of the values 's3', 's3-sts', 'azure', or 'swift'
  - [note]: (string) Optional. Any text description added to the credential
  - when "type" is "s3" or "s3-sts":
    - accessKey: (string) The public key associated with this credential. This parameter is only required for 's3' credentials.
    - privateKey: (string) The private key associated with this credential. This parameter is only required for 's3' credentials.
  - when "type" is "azure":
    - tenant: (string) AD tenant to associate with this credential.
    - subscription: (string) Azure subscription to associate with this credential.
    - clientId: (string) AD clientId to associate with this credential.
    - clientSecret: (string) AD client secret to associate with this credential
  - when "type" is "swift"
    - tenant: (string) Swift tenant to associate with this credential.
    - user: (string) Swift user to associate with this credential.
    - auth_url (string) URL of the swift authentication service.
    - password: (string) Password corresponding to the tenant/user pair.

RETURNS
status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.corefiler.createCredential('testCred', 's3',
 {'note': 'Credential for help', 'accessKey': 'AKIAI6...FC73RE
 I7Q', 'privateKey': 'z/TaNqLco0rn...4yI8nJOI'})
success
```

corefiler.downloadKeyRecoveryFile

NAME

corefiler.downloadKeyRecoveryFile

SYNOPSIS

corefiler.downloadKeyRecoveryFile(cloudFilerName) => resultStruct

DESCRIPTION

Downloads the key recovery file (encoded string) for a cloud core filer, generated by the most recent corefiler.generateMasterKey operation.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- cloudFilerName: (string) The name of the core filer

RETURNS

- resultStruct: An XML-RPC struct containing the following name:value pairs:

  - keyId: (string) The ID of the key, in hex format
  - signature: (string) The SHA-1 signature of the recovery file
  - recoveryFile: (string) The master key, represented as a base64-encoded string

EXAMPLE

```
print clientHandle.corefiler.downloadKeyRecoveryFile('blueAngel')
{'keyID': '10003330000000002', 'signature': '980ddb19cf4bd5abb0f24d
1120bcd29c83bb36f3','recoveryFile': 'U2FsdGVkX18fePMC0ZLCaGYKUTvgHo
7JTD+m4RlhI0t6GedakgkExhqvQqHERojErrM4DYjD/QSN\nZtblnTSnwhZL74fWBuj
f1dOs3xK6fysBYMPhS8C+iVZP1BRTH4HHI1ys+8gjLT0IsjVjh2sGfHTH\nMN9I2rMO
QkdiMwGF4lJv8OXr7BUr4VDZYXByIdo7niV7y6DX1jzbGwBUrdbQLWIR/likL8HPy3r
Z\nbw2nL2ykeLF1urHU2OC6d/59qrMrBgAAAAEA\n'}
```

NAME

corefiler.enableLocalDirectories

SYNOPSIS

corefiler.enableLocalDirectories(filerName, enableState) => deprecated

DESCRIPTION

This call is deprecated. It is replaced by cachePolicy.modify where you can modify local directories settings on a single cache policy that applies to N core filers. You can also use cachePolicy.create to create a new cache policy with specific settings and then use corefiler.setCachePolicy to set that cache policy on a core filer.

corefiler.generateMasterKey

NAME

corefiler.generateMasterKey

SYNOPSIS

corefiler.generateMasterKey(cloudFilerName, [passphrase], [userKey]) => resultStruct

DESCRIPTION

Creates a new master encryption key for use when storing an object to a cloud core filer, which will be activated using the corefiler.activateMasterKey method.

NOTE: The cloud filer must have encryption enabled when it is created.

WARNING: It is critically important to save a copy of the most recently generated recovery file (key string). If the FXT cluster suffers a failure, this file will be needed to recover data from the cloud. It is also a required argument for the activateMasterKey method.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- cloudFilerName: (string) The name of a cloud core filer with encryption enabled
- [passphrase]: (string) Passphrase used to encrypt the returned key recovery file, required for simpleKey and disallowed for kmip.
- [userKey]: (string) Optional. A user-provided key, in hex format.

RETURNS

- resultStruct: An XML-RPC struct containing the following name:value pairs:

  - keyId: (string) The ID of the newly generated key, in hex format
  - signature: (string) Optional. The SHA-1 signature of the recovery file if cloud filer keyMgmtType is simpleKey.
  - recoveryFile: (string) Optional. The new master key, represented as a base64-encoded string if cloud filer keyMgmtType is simpleKey.

EXAMPLE

```
print clientHandle.corefiler.generateMasterKey('cloudFiler1',
  'supersecret','737570657272736563726574')
{'keyId': '10000000000000002', 'signature': '980ddb19cf4bd5abb0f24d112
0bcd29c83bb36f3', 'recoveryFile': 'U2FsdGVkX18fePMC0ZLCaGYKUTvgHo7JTD+
m4RlhI0t6GedakgkExhqvQqHERojErrM4DYjD/QSN\nZtblnTSnwhZL74fWBujf1dOs3xK
6fysBYMPhS8C+iVZP1BRTH4HHI1ys+8gjLT0IsjVjh2sGfHTH\nMN9I2rMOQkdiMwGF4lJ
v8OXr7BUr4VDZYXByIdo7niV7y6DX1jzbGwBUrdbQLWIR/likL8HPy3rZ\nbw2nL2ykeLF
F1urHU2OC6d/59qrMrBgAAAAEA\n'}
print clientHandle.corefiler.generateMasterKey('cloudFiler2')
{'keyId': '10000000000000002'}
```

corefiler.get

NAME
corefiler.get

SYNOPSIS
corefiler.get({filerName | filerNameArray}) => filer_info_struct

DESCRIPTION
Returns detailed information for the specified core filer or filers.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
One of the following:
- filerName: (string) The name of a core filer
- filerNameArray: (array) An array of core filer names. The names are strings.

RETURNS
- filer_info_struct: An XML-RPC struct that specifies one or more of the following name:value pairs for each core filer:

  - type: (string) The type of core filer, either 'NFS' or 'cloud'

  - writeMode: (string) Write mode for the core filer, any of the following:
    - 'Write Through' if all external client updates are performed through the cluster. The client can still transfer information directly TO the core filer. This mode is equivalent to a 'Read' caching policy.
    - 'Write Around' if the client can transfer information directly to and from the core filer without using data cached in the cluster. If 'checkAttributesPeriod' is greater than or equal to 0, then attribute checks are performed on each file before using cached data. This mode is equivalent to a 'Read' caching policy.
    - 'Write Back' if information transferred between the client and the core filer is all passed through the cluster. This mode is equivalent to a 'Read/Write' caching policy.
  - maxWritebackDelay:
    (integer) Maximum writeback delay, in seconds, for the core filer

  - name: (string) Core filer name

  - writeThroughSchedule:
    An XML-RPC struct that specifies one or more of the following name:value pairs for each core filer:
    - schedule: (string) Name of the schedule
    - One of the following, depending on the schedule's polling method:
      - pollWait: (integer) The poll-waiting period, in minutes
      - pollUrl: (string) A URL to poll for a value

  - snapshotName: (string) Name of the core filer's snapshot directory, '.snapshot' is the default

  - rev: (string) The core filer's revision number

  - checkAttributesPeriod:
    (integer) The interval in seconds between file attribute checks when 'writeMode' is "Write Around'.

- If this value is greater than or equal to 0, then attribute checks
  are performed on each file before using cached data.
- If this value is blank or less than 0, then attribute checks are
  never performed before using cached data.

- adminState: (string) The core filer's administrative state, one of the following:
  - 'online' if the core filer is available to the cluster
  - 'offline' if the core filer is not available to the cluster, but
    still recognized by the cluster
  - 'removing' if the core filer is in the process of being removed
  - 'removed' if the core filer was previously recognized by the
    cluster, but has been removed from the cluster configuration
  - 'flushing' if the cluster is transferring information from the cache to the core filer

- vservers: (array) The names of vservers associated with the core filer. The names are strings.

- dataMgmtFeatures:
  (string) The data management features supported by, although not necessarily licensed
  to, this core filer; either 'move' or 'move,mirror'

- networkName: (string) The primary IP address or fully qualified domain name of the core filer.
  - For NFS core filers, this may also be a space-separated list of IP addresses or
  domain names, where subsequent network names are used in advanced
  networking configurations.
  - For cloud core filers, this is the address of the cloud
  service provider. For AmazonS3, this is the endpoint of the S3 region being used.

- network: (string) The name of the network the cluster will use to communicate with this core filer.

- id: (string) The core filer's UUID

- localDirectories:
  (boolean) Whether local directories are enabled (True) or not (False)

- corefilerIPs: (array) The core filer's IP address or addresses. The IP addresses are strings.

- cloudType: (string) Cloud core filers only. The type of the cloud filer.

- s3Type: (string) Cloud core filers only. The type of S3 service provider.
  Valid values are 'Amazon', 'Amplidata', and 'Cleversafe'.

- bucket: (string) Cloud core filers only. The name of the bucket.

- region: (string) The bucket's location constraint for Amazon S3 cloud filers.

- cloudCredential: (string) Cloud core filers only. The cloud credential name.

- port: (string) Cloud core filers. The port number used by the cloud core filer.
  Some common values are:
  - '443' for https (the default)
  - '80' for http

  Other port numbers may also be used for some cloud core filers. A space-separated list of
port
  numbers or a range of port numbers separated by a hyphen ('-') symbol may also be used.
  The total number of ports cannot exceed 8.

- https: (string) For cloud core filers, determines whether HTTPS is used for security, either 'yes' or 'no'.

- pathStyle: (string) For cloud core filers, whether to use path-style request or not. Values are 'yes', or 'no'.

- cryptoMode: (string) The encryption mode used by cloud core filers. It can be one of the following:
  - 'DISABLED', or
  - 'CBC-AES-256-HMAC-SHA-512'

- compressMode: (string) The compression mode used by cloud core filers. It can be one of the following:
  - 'DISABLED'
  - 'LZ4'
  - 'LZ4HC (deprecated)'

- internalName: (string) The internal name of the corefiler. The internal name starts with the prefix 'mass'
  followed by an unique integer identifier. The use of the corefiler internal name is found in custom settings and in the names of some statistics counters.

- clientSuspendStatus:
  (string) The client suspend status. See the vserver.clientSuspendStatus method.

- sslVerifyMode: (string) Optional. The certification verification used by the cluster to verify the X.509 certificate
  returned by the cloud filer for identification. The options are:

  - 'OCSP_CRL' use OCSP to verify the certificate. If the connection to the
    OCSP responder fails fall back to CRL.
  - 'OCSP' use OCSP to verify that the certificate is valid.
  - 'CRL' use CRL to inspect and verify that the certificate is valid.
  - 'DISABLED' (DANGEROUS) do not inspect or validate the certificate, and trust it blindly.

- [proxy]: (string) Optional. The name of a proxy configuration to use when verifying the cloud filer certificate and accessing the cloud filer storage provider bucket.

- nearline: (string) Optional. Set to 'yes' if this cloud filer will be connected to a Google Cloud Storage Nearline bucket.
  WARNING: connecting a cloud filer to nearline storage can result in excessive charges due to filesystem
  directory metadata updates. To reduce this risk choose a cache policy for this cloud filer with a writeback
  time of two hours or more.

- autoWanOptimize: (string) Optional for NFS core filers. If enabled WAN optimization is in effect for this core filer.
  WAN optimization is enabled on cloud filers on creation and it cannot be disabled. Note that this setting
  is a custom setting and it can also be displayed and modified using the support custom option commands.

- filerClass: (string) A predefined value that describes this core filer.
  Acceptable NAS values:
  NetappNonClustered

NetappClustered
HitachiHNAS
EmcIsilon
Other
Acceptable Cloud value:
AvereCloud

EXAMPLE

```
print clientHandle.corefiler.get('thor')
thor = '{'writeMode': 'Write Through', 'maxWritebackDelay': 43200,
'name': 'thor', 'writeThroughSchedule': {'schedule': 'checkOps',
'pollWait': 15}, 'snapshotName': '.snapshot', 'rev': '362c2432-c31d
-11e2-b233-000c299a83be', 'checkAttributesPeriod': 30, 'adminState':
'online', 'vservers': ['vserver1','vserver2','vserver3'],
'dataMgmtFeatures':'move,mirror', 'networkName': 'thor.cc.company.com',
'id': '1a9efa46-c31d-11e2-b233-000c299a83be', 'localDirectories':
False, 'corefilerIPs': [{'IP': '10.1.11.50'}], 'internalName' : 'mass15'  }'
```

corefiler.getBandwidthThrottle

NAME
corefiler.getBandwidthThrottle

SYNOPSIS
corefiler.getBandwidthThrottle(filerName) => throttleStruct

DESCRIPTION
Returns the bandwidth-throttling settings for the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer

RETURNS
- throttleStruct: An XML-RPC struct that contains the following name:value pairs:

  - maxBandwith: (integer) The core filer's maximum bandwidth in MB per second, between 1 and 800. This value will be returned if 'enable' is True.
  - enable: (boolean) Whether bandwidth throttling is enabled for the core filer (True), or not (False). If this value is True, the 'maxBandwidth' value will also be specified.
  - name: (string) The name of the core filer
  - earlyWriteback: (boolean) Whether the cluster commits written data to the core filer before the maxWritebackDelay period (True) or not (False)

EXAMPLE
```
print clientHandle.corefiler.getBandwidthThrottle('thor')
{'maxBandwidth': '60', 'enable': True, 'name': 'thor', 'earlyWriteback': False}
```

NAME

corefiler.getCacheQuota

SYNOPSIS

corefiler.getCacheQuota(filerName)=> cacheQuota_struct

DESCRIPTION

Returns the cache utilization control settings for the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer

RETURNS

- cacheQuota_struct: An XML-RPC struct that contains the following name:value pairs:

  - policy: (string) The cache utilization control setting, one of the following:
    - 'uid' applies controls on a per-user basis
    - 'export' applies controls on a per-export basis
    - 'fsid' applies controls on a per-FSID basis
    - 'qtree' applies controls to the core filer
    - 'export,uid' applies controls on a per-export and a per-user basis
    - 'fsid,uid' applies controls on a per-FSID and a per-user basis
    - 'qtree,uid' applies controls to the core filer, and on a per-user basis
  - enabled: (boolean) Whether cache utilization control is enabled for this core filer (True) or disabled (False)
  - name: (string) The name of the core filer

EXAMPLE

```
print clientHandle.corefiler.getCacheQuota('thor')
{'policy': '', 'enabled': False, 'name': 'thor'}
```

corefiler.getCredential

NAME
corefiler.getCredential

SYNOPSIS
corefiler.getCredential(credentialName) => credentialStruct

DESCRIPTION
Returns information about a cloud credential, which depends on the credential type.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- credentialName: (string) The name of the credential

RETURNS
- credentialStruct: An XML-RPC struct containing the following name:value pairs:

  - coreFilers: (array) The names of the core filers using this credential
  - internalName: (string) The name provided by the system for the credential
  - name: (string) The name given to the credential using the corefiler.createCredential method
  - accessKey: (string) The public key associated with this credential. This parameter is only returned for 's3' and 'swift' credentials.
  - tenant: (string) The cloud 'tenant' associated with this credential. This parameter is only returned for 'azure' and 'swift' credentials.
  - user: (string) The cloud 'user' associated with this credential. This parameter is only returned for 'swift' credentials (accessKey is a synonym for 'user' in the swift case).
  - auth_url (string) The URL of the authentication service for this credential. This parameter is only returned for 'swift' credentials.
  - [note]: (string) Optional. Any text description added to the credential
  - type: (string) The type of cloud filer for which the credential is intended.

EXAMPLE
```
print clientHandle.corefiler.getCredential('updatedCred')
{'coreFilers': [ ], 'internalName': 'cloudCredential1', 'name': 'updatedCred',
'accessKey': 'AKIAI...FC73REI7Q', 'note': 'Updated information', 'type': 's3'}
```

corefiler.getNfs

NAME

corefiler.getNfs

SYNOPSIS

corefiler.getNfs(filerName) => settingsStruct

DESCRIPTION

Returns the NFS attributes of the core filer; that is, whether Kerberos is used or not.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer

RETURNS

- settingsStruct: An XML-RPC struct that contains the following name:value pair:
  - kerberos: (string) Whether the NFS system can work with Kerberos ('yes') or not ('no')

EXAMPLE

```
print clientHandle.corefiler.getNfs('grape')
{'kerberos': 'no'}
```

NAME

corefiler.list

SYNOPSIS

corefiler.list() => filerNameArray

DESCRIPTION

Lists core filer names associated with the cluster.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- No input parameters are required for this method.

RETURNS

filerNameArray: (array) An array of core filer names. The names are strings.

EXAMPLE

```
print clientHandle.corefiler.list()
['thor', 'gentoo', 'grape']
```

corefiler.listCredentials

NAME
corefiler.listCredentials

SYNOPSIS
corefiler.listCredentials() => array_of_structs

DESCRIPTION
Returns an array of cloud credentials known to the cluster.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - coreFilers: (array) The names of the core filers using this credential
  - internalName: (string) The name provided by the system for the credential
  - name: (string) The name given to the credential using the corefiler.createCredential method
  - [accessKey]: (string) The public key associated with this credential. This parameter is only returned for 's3' and 'swift' credentials.
  - [tenant]: (string) The cloud tenant associated with this credential. This parameter is only returned for 'azure' and 'swift' credentials;
  - [user]: (string) The user name associated with this credential (an alias for 'accessKey'). This parameter is only returned for 'swift' credentials.
  - [auth_url]: (string) The authentication service URL associated with this credential. This parameter is only returned for 'swift' credentials.
  - [note]: (string) The text description added to the credential, if any
  - type: (string) The type of cloud filer for which the credential is intended.

EXAMPLE
```
print clientHandle.corefiler.listCredentials()
[{'coreFilers': ['testFiler'], 'internalName': 'cloudCredential1',
'name': 'testCred', 'accessKey': 'AKIAI6...FC73REI7Q', 'note': 'Cr
edential for help', 'type': 's3'}]
```

NAME

corefiler.listExports

SYNOPSIS

corefiler.listExports({filerName | filerNameArray}) => filerInfoStruct

DESCRIPTION

Returns the export list for the specified core filer or list of core filers.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

NOTE: The corefiler.listExports method returns the export list from the core filer, whereas the nfs.listExports method returns the export list, and might also contain vserver-specific objects.

PARAMETERS

One of the following:

- filerName: (string) The name of a core filer
- filerNameArray: (array) An array of core filer names. The names are strings.

RETURNS

- If advanced networking is enabled, an array of structs, each of which contains the following name:value pairs:

- filerInfoStruct: An XML-RPC struct that contains the following name:value pairs:
  - <filerName>: (string) The name of the core filer

  - exportArray: An array of structs, each of which contains the following element (name:value pair) for each export:
  - path: (string) The path to any exports on the core filer

EXAMPLE

```
print clientHandle.corefiler.listExports('thor')
{'thor': '[{'path': '/vol0/user-a'}, {'path': '/vol0/user-b'}]}
```

NAME

corefiler.masterKeyStatus

SYNOPSIS

corefiler.masterKeyStatus(cloudFilerName) => statusStruct

DESCRIPTION

Returns information about the master key currently in use by a given cloud core filer, and any generated key waiting to be activated.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- cloudFilerName: (string) The name of the cloud filer

RETURNS

- statusStruct: An XML-RPC struct that contains the following name:value pairs:

  - activeMasterKeyId:
    (string) The ID of the master key currently being used by the cloud filer, in hex format

  - [masterKeyIdReadyToActivate]:
    (string) The latest key that has been generated, in hex format. If this parameter is returned, the 'recoveryFileSignature' is also returned.

  - keyActivationTime: (string) Either 'NO ACTIVATION TIME' or the time in Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970

  - nodesWithoutLatestMasterKey:
    (string) A space-separated list of nodes that don't have the new master key waiting to be activated. This is usually a temporary condition, caused by nodes being down or by networking problems.

  - keyActivationTime:
    (string) Either 'NO ACTIVATION TIME' or the time in Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970

  - recoveryFileSignature:
    (string) The SHA-1 signature of the recovery file corresponding to the master key waiting to be activated. This parameter will be returned if 'masterKeyIdReadyToActivate' is returned.

- activeMasterKeyId:
  (string) Either 'NO ACTIVE MASTER KEY' or the master key currently being used

EXAMPLE

The following python script shows the status of a cloud core filer that does not have an activated key, and then with the key activated.

```
print clientHandle.corefiler.masterKeyStatus('blueAngel')
{'masterKeyIdReadyToActivate': '10000000330000002', 'nodesWithoutLatestMasterKey': '',
'KEKActivationTime': 'NO ACTIVATION TIME', 'recoveryFileSignature': '980ddb19cf4bd5abb
0f24d1120bcd29c83bb36f3', 'activeMasterKeyId': 'NO ACTIVE MASTER KEY'}
```

```
print clientHandle.corefiler.activateMasterKey('blueAngel', '10000000330000002', 'U2Fs
d...GV9c83bbkX')
success

print clientHandle.corefiler.masterKeyStatus('blueAngel')
{'masterKeyIdReadyToActivate': '10000000330000002', 'nodesWithoutLatest
MasterKey': '', 'keyActivationTime': '1381769226', 'recoveryFileSignatu
re': '980ddb19cf4bd5abb0f24d1120bcd29c83bb36f3', 'activeMasterKeyId': '
10000000330000002'}
```

corefiler.modifiedFilesInfo

NAME
corefiler.modifiedFilesInfo

SYNOPSIS
corefiler.modifiedFilesInfo(filerName) => fileInfoStruct

DESCRIPTION
Indicates whether files have been changed on the Avere cluster
but have not yet been written back to the core filer ("dirty files").

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer

RETURNS
- fileInfoStruct: An XML-RPC struct that contains the following name:value pairs:

  - oldestAge: (integer) The number of seconds since the 'oldest' (last written-back) file in the set of modified files was written back to the core filer
  - numFiles: (signed int8) The number of modified files on the core filer

EXAMPLE
```
print clientHandle.corefiler.modifiedFilesInfo('thor')
{'oldestAge': 85, 'numFiles': 8.0}
```

NAME
corefiler.modify

SYNOPSIS
corefiler.modify(filerName, struct) => status
corefiler.modify(filerName, newNetworkName, [ignoreWarning]) => status (deprecated)

DESCRIPTION
Modifies the core filer network name.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

NOTE: corefiler.modify(filerName, newNetworkName, [ignoreWarning]) is a deprecated API that cannot be used to modify cloud core filers.

PARAMETERS
- filerName: (string) The name of the core filer

- newNetworkName: (deprecated) The new primary IP address or fully qualified domain name of the core filer. This may also be a space-separated list of IP addresses or domain names, where subsequent network names are used in advanced networking configurations.

- [ignoreWarning]: (deprecated) Optional. Whether warnings will be displayed during core filer modification (False - the default) or not (True).

  The new network name must point to the same core filer unless this parameter is set to True.

- settingStruct: An XML-RPC struct containing the following name:value pairs.

  - networkName: (string) The new primary IP address or fully qualified domain name of the core filer.
    - For NFS core filers, this may also be a space-separated list of IP addresses or domain names, where subsequent network names are used in advanced networking configurations.
    - For cloud core filers, this is the address of the cloud service provider.
    - For AmazonS3 cloud filers, it can only take endpoint addresses that belong to the same region as the bucket's location constraint.

  - [ignoreWarning]: (boolean) Optional. Whether warnings will be displayed during core filer modification (False - the default) or not (True).

    The new network name must point to the same core filer unless this parameter is set to True.

  - https: (string) Optional. Cloud core filer only. Set to 'yes' if https encryption is used for communication between the cluster and the core cloud filer. Set to 'no' for clear text communication.

  - port: (string) Optional. Cloud core filer only. Set the port value for http(s) communication between the cluster and the cloud filer. The default value is 80 for http and 443 for https. Set this value only if your site

uses non-standard http(s) ports. If setting the port value the
https value must be set as well. A list of port numbers may
also be used separated by space. A range of ports can also
be specified using the hyphen ('-') symbol. A maximum of 8 ports can be used.

- [pathStyle]: (string) Optional, cloud core filers only. Set to 'yes' if the cloud
filer uses path-style requests.

- [compressMode]: (string) Optional, cloud core filers only. The choices are 'DISABLED' or 'LZ4'
compression.

- filerMgmtUrl: (string) The path to the core filer web interface.

- filerDescription: (string) Text description of the core filer.

- accessMode: (string) Set to 'yes' to force access cache enabled for the core filer. Set
to 'no' to have the access cache determined by CIFS related configuration.

- extractFsidFromFilehandle:
(string) Optional. A value of "yes" enables extracting the fsid from the filehandle for this core filer.
The default is 'no'. Only change this option from the default if the core filer does not
return persistent FSID values in NFS attributes and the core filer does not provide a
mechanism to assign persistent FSIDs. For NFS filers only.

- sslVerifyMode: (string) Optional. The certification verification used by the cluster to verify the X.509 certificate
returned by the cloud filer for identification. The options are:

- 'OCSP_CRL' use OCSP to verify the certificate. If the connection to the
OCSP responder fails fall back to CRL.
- 'OCSP' use OCSP to verify that the certificate is valid.
- 'CRL' use CRL to inspect and verify that the certificate is valid.
- 'DISABLED' (DANGEROUS) do not inspect or validate the certificate, and trust it blindly.

- [proxy]: (string) Optional. The name of a proxy configuration to use when verifying the cloud filer
certificate and accessing the cloud filer storage provider bucket.

- clientSuspend: (string) Optional. Set to 'yes' to suspend client access to all junctions on all vservers
associated with this core filer. See the vserver.clientSuspendStatus method
for client suspend response options.

- autoWanOptimize: (string) Optional for NFS core filers. Set to 'enable' to enable WAN optimization for NFS core filers.
Set to 'disable' to disable WAN optimization. WAN optimization enabled on cloud filers on creation
and it is not modifiable. Note that this setting is a custom setting and will it is also displayed
and managed from the settings support page in the GUI.

- filerClass: (string) One of the following predefined values that describes this NAS core filer.
Modifying filerClass is not permitted for cloud core filers.
Acceptable NAS values:
NetappNonClustered
NetappClustered
HitachiHNAS
EmcIsilon

Other

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

The following example returns an activityID for the method.

```
print clientHandle.corefiler.modify('thor','10.2.22.195',True)
af7516af-f49c-11e2-8d3e-000c299a83be

print clientHandle.corefiler.modify('Amazon1',{'networkName':'s3-us-west-1.amazonaws.com',
  'ignoreWarning' : True, 'https' : 'yes', 'filerDescription' : 'cloud
  filer example' })
af7516af-f49c-11e2-8d3e-000c299a83be
```

corefiler.modifyAutoExcludeList

NAME

corefiler.modifyAutoExcludeList

SYNOPSIS

corefiler.modifyAutoExcludeList(filerName, autoExcludeList) => status

DESCRIPTION

Sets the list of file or directory names that the cluster will exclude from its cache. These files are typically distinct to the file system running on the core filer, and commonly include files such as ".etc" and ".zfs". Do not specify the snapshot file name using this RPC.

PARAMETERS

- filerName: (string) The name of the core filer
- autoExcludeList: (string) A comma-delimited list of names (spaces are ignored) that the cluster will not keep in its cache or treat as read-only because they are distinct to the core filer. You may not specify full paths (no '/' characters) or wild card characters in this list.

RETURNS

- status: (string) success or a reason for failure

EXAMPLE

```
print clientHandle.corefiler.modifyAutoExcludeList('myfiler', '.etc, .zfs')
success
```

corefiler.modifyBandwidthThrottle

NAME

corefiler.modifyBandwidthThrottle

SYNOPSIS

corefiler.modifyBandwidthThrottle(filerName, isEnabled, [maxBandwidth], [earlyWriteback]) => status

DESCRIPTION

Changes the bandwidth-throttling settings for the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer
- isEnabled: (boolean) Whether bandwidth throttling is enabled for the core filer (True), or not (False). If this value is True, the 'maxBandwidth' value must also be specified.

- [maxBandwith]: (integer) Optional. The core filer's maximum bandwidth in MB per second, between 1 and 800. Specify this value if 'isEnabled' is set to True.

- [earlyWriteback]: (boolean) Optional. Whether the cluster commits written data to the core filer before the maxWritebackDelay period (True) or not (False). This parameter can be specified only if 'isEnabled' is set to True AND a value is specified for the 'maxBandwidth' parameter.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.corefiler.modifyBandwidthThrottle('thor', True, 300, False)
success
```

NAME

corefiler.modifyCachePolicy

SYNOPSIS

corefiler.modifyCachePolicy(filerName, policy, [writebackDelay],
[checkIntervals], [writethroughScheduleSettings]) => deprecated

DESCRIPTION

This RPC is deprecated. It is replaced by cachePolicy.modify, where you can modify the cache policy settings for a single cache policy that applies to N core filers. You can also use cachePolicy.create to create a new cache policy with specific cache policy settings and then use that cache policy on a core filer using corefiler.setCachePolicy.

NAME

corefiler.modifyCacheQuota

SYNOPSIS

corefiler.modifyCacheQuota(filerName, enabled, [policy]) => deprecated

DESCRIPTION

This RPC is deprecated. It is replaced by cachePolicy.modify, where you can modify the cache quota setting for a single cache policy that applies to N core filers.
You can also use cachePolicy.create to create a new cache policy with specific cache quota settings and then use that cache policy on a core filer using corefiler.setCachePolicy.

corefiler.modifyCredential

NAME
corefiler.modifyCredential

SYNOPSIS
corefiler.modifyCredential(credentialName, attrsStruct) => status

DESCRIPTION
Modifies an existing cloud credential.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- credentialName: (string) The name of the credential

- attrsStruct: An XML-RPC struct containing the following name:value pairs (see below for additional type-specific values):

  - [name]: (string) A new name for the credential
  - [note]: (string) Optional. Any text description added to (or changed in) the credential
  - [type]: (string) Optional. Must be one of the values 's3', 's3-sts', 'azure', or 'swift', and must match the type of the existing cloud credential.
  - when "type" is "s3" or "s3-sts":
    - [accessKey]: (string) Optional, unless the 'privateKey' parameter is specified. A new public key to associate with this credential.
    - [privateKey]: (string) Optional, unless the 'accessKey' parameter is specified. A new private key to associate with this credential.
    - [token]: (string) Required when updating the privateKey of an 's3-sts' credential. Only valid for 's3-sts' credentials.
    - [expiration]: (string) Optional, but highly recommended, when updating the privateKey of an 's3-sts' credential. Format is either an ISO date string in the GMT timezone ("2015-03-14T09:26:53Z") or a decimal unix timestamp ("1426339613"). Only valid for 's3-sts' credentials.
  - when "type" is "azure":
    - [tenant]: (string) Optional. A new Azure tenant to associate with this credential.
    - [subscription]: (string) Optional. A new Azure subscription to associate with this credential.
    - [clientId]: (string) Optional. A new Azure application client id to associate with this credential.
    - [clientSecret]: (string) Optional. A new Azure application client secret to associate with this credential
  - when "type" is "swift":
    - [tenant]: (string) Optional. A new Swift tenant to associate with this credential.
    - [user]: (string) Optional. A new Swift user to associate with this credential.
    - [password]: (string) Optional. A password to associate with this credential.
    - [auth_url]: (string) Optional. A new authentication service URL to associate with this credential.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.corefiler.modifyCredential('testCred',
  {'name': 'updatedCred', 'note': 'Updated information'})
success
```

corefiler.modifyKeyManagementType

NAME

corefiler.modifyKeyManagementType

SYNOPSIS

corefiler.modifyKeyManagementType(filerName, newType, newTypeArgs) => status

DESCRIPTION

Modifies the core filer key management type of encryption keys.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer
- newType: (string) The name of the new key management type (simpleKey or kmip).
- newTypeArgs: An XML-RPC struct containing the following name:value pairs:
  - [name]: (string) Required if the newType is kmip. The name of the new kmip server
  - [passphrase]: (string) Required if the newType is simpleKey. It used to encrypt the key recovery file

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.corefiler.modifyKeyManagementType('filername','kmip', {'name':'kmip0'})
success
```

NAME
corefiler.modifyNetwork

SYNOPSIS
corefiler.modifyNetwork(filerName, networkName) => status

DESCRIPTION
Configures which network a core filer should use for communication.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer.
- networkName: (string) The name of the network to use for communication.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.corefiler.modifyNetwork('filername','testnetwork')
success

corefiler.modifyNfs

NAME

corefiler.modifyNfs

SYNOPSIS

corefiler.modifyNfs(filerName, settingsStruct) => status

DESCRIPTION

Modifies the NFS attributes of the core filer; that is, whether Kerberos is used or not.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer
- settingsStruct: An XML-RPC struct that contains the following name:value pair:
  - kerberos: (string) Whether the NFS system can work with Kerberos ('yes') or not ('no')

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.corefiler.modifyNfs('grape', {'kerberos': 'yes'})
success
```

corefiler.modifySnapshotName

NAME
corefiler.modifySnapshotName

SYNOPSIS
corefiler.modifySnapshotName(filerName, snapshotName) => status

DESCRIPTION
Changes the snapshot directory name for the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer
- snapshotName: (string) The new directory name (the default is .snapshot)

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.corefiler.modifySnapshotName('thor', '.decSnapshot')
success
```

NAME

corefiler.modifyWriteMode

SYNOPSIS

DEPRECATED

DESCRIPTION

This RPC is deprecated. It is replaced by cachePolicy.modify, where you can modify the cache policy settings for a single cache policy that applies to N core filers. You can also use cachePolicy.create to create a new cache policy with specific cache policy settings and then use that cache policy on a core filer using corefiler.setCachePolicy.

NAME
corefiler.removeCredential

SYNOPSIS
corefiler.removeCredential(credentialName) => status

DESCRIPTION
Deletes a cloud credential.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- credentialName: (string) The name of the credential to remove

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.corefiler.removeCredential('cloudCred')
success
```

NAME
corefiler.rename

SYNOPSIS
corefiler.rename(filerName, newName) => status

DESCRIPTION
Renames the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer
- newName: (string) The new name for the core filer

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.corefiler.rename('oldName', 'newName')
success

NAME

corefiler.setCachePolicy

SYNOPSIS

corefiler.setCachePolicy(filerName, cachePolicyName) => status

DESCRIPTION

Set a new cache policy for a core filer.

NOTE: This immediately impacts the way that the cluster caches data for the core filer, and may cause damage to the file system when transitioning between certain cache policies if used improperly. Be sure that you are aware of the proper procedures required when changing cache policies on a core filer. Using the GUI can help walk you through this process. It is recommended that you only use this RPC under the recommendation of Avere Global Services.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the core filer
- cachePolicyName: (string) The name of the cache policy that you would like the cluster to start using when caching data for this core filer.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.corefiler.setCachePolicy('myFiler','Clients Bypassing the Cluster')
success
```

NAME
corefiler.setCredential

SYNOPSIS
corefiler.setCredential(filerName, credentialName) => status

DESCRIPTION
Selects which credential a cloud core filer uses when authenticating to a cloud provider.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the cloud core filer
- credentialName: (string) The name of the credential the filer should use

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.corefiler.setCredential('blueAngel','updatedCred')
success

NAME
corefiler.unsuspend

SYNOPSIS
corefiler.unsuspend(filerName) => status

DESCRIPTION
Unsuspends the specified core filer.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS
- filerName: (string) The name of the core filer

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.corefiler.unsuspend('gentoo')
success
```

corefiler.uploadKeyRecoveryFile

NAME

corefiler.uploadKeyRecoveryFile

SYNOPSIS

corefiler.uploadKeyRecoveryFile(filerName, passphrase, recoveryFile) => status

DESCRIPTION

Installs master keys from a recovery file into a cloud core filer. This operation is useful when a cloud core filer needs to be attached to a cloud containing encrypted data previously stored by an Avere FXT cluster, such as during disaster recovery.

NOTE: It is only valid to upload a recovery file to a new cloud core filer; this must be done before invoking the generateMasterKey method.

NOTE: "mass" is also accepted as a deprecated alias for the "corefiler" module.

PARAMETERS

- filerName: (string) The name of the cloud core filer
- passphrase: (string) The recovery file passphrase which was supplied to the corefiler.generateMasterKey method
- recoveryFile: (string) The new master key, represented as a base64-encoded string

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

The first of the following Python examples provides the recoveryFile value directly. The second example assumes that 'result' was generated using the corefiler.generateMasterKey method:

```
print clientHandle.corefiler.uploadKeyRecoveryFile('blueAngel',
  'supersecret', 'U2FsdGVkX18fePMC0ZLCaGYKUTvgHo7JTD+...iV7y6DX
  1jzbGwBUrdbQLWIw2nL2ykeLF1urHU2OC6d/59qrMrBgAAAAEA\n'}
success

print clientHandle.corefiler.uploadKeyRecoveryFile('cloudFiler1',
  'supersecret', result['recoveryFile'])
success
```

NAME

dirServices.adLookup

SYNOPSIS

dirServices.adLookup(dir_service, ADS_FQDN) => AD_info_struct

DESCRIPTION

Returns information about the Active Directory domain.

PARAMETERS

- dir_service: (string) The name of a directory services instance; the only currently valid value is 'default'
- ADS_FQDN: (string) The fully qualified domain name (for example, 'AD_controller.example.com') of the AD domain controller

RETURNS

- AD_info_struct: An XML-RPC struct that contains the following name:value pairs for an Active Directory (AD) domain:

  - FXTsite: (string) The geographic location of the FXT cluster
  - DCsite: (string) The geographic location of the domain controller
  - DCname: (string) The domain controller's name
  - ADforestName: (string) The name of the AD forest
  - ADworkgroupDiscovered: (string) If a a workgroup is discovered by the software, and is too long for operating systems previous to Windows 2000, this parameter is a name that can be used by pre-Windows 2000 clients

EXAMPLE

```
print clientHandle.dirServices.adLookup('default', 'ad.company.com')
{'FXTsite': 'company_location', 'DCsite': 'company_location',
'DCname': 'HQ-AD.ad.company.com', 'ADforestName': 'ad.company.com',
'ADworkgroupDiscovered': 'DEV'}
```

dirServices.addAdOverride

NAME

dirServices.addAdOverride

SYNOPSIS

dirServices.addAdOverride(dir_service, netbios, fqdn, addresses) => status

DESCRIPTION

Creates an Active Directory (AD) override for a domain.

PARAMETERS

- dir_service: (string) The name of a valid directory service. Currently the only valid name is 'default'.
- netbios: (string) The NetBIOS name of the AD domain
- fqdn: (string) The fully qualified domain name (for example, company.domain.com) of the AD domain
- addresses: (string) A space-separated list of one or more IPv4 addresses

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.addAdOverride('default', 'ops',
  'ops.directory.com', '10.0.0.40')
success
```

NAME
dirServices.downloadCert

SYNOPSIS
dirServices.downloadCert(dir_service) => job_UUID

DESCRIPTION
Starts a background job to download an SSL server certificate to be used to validate secure LDAP connections.

PARAMETERS
- dir_service: (string) The name ('default' or 'login') of a directory service instance

RETURNS
- job_UUID: (string) The UUID of the background job that downloads and distributes the SSL certificate

EXAMPLE
print clientHandle.dirServices.downloadCert('default')
ebf0f521-0000-11e3-816a-000c29159544

NAME
dirServices.get

SYNOPSIS
dirServices.get(dir_service, [component]) => attributes_struct

DESCRIPTION
Returns a struct containing the attributes of the specified directory service.

PARAMETERS
- dir_service: (string) The directory service. There are currently only two valid values, 'default' and 'login'.

- [component]: (string) Optional. One of the following aspects of the directory service. If this is specified, the method only returns information about that component, which is one of the following:
  - 'LDAP'
  - 'NIS'
  - 'AD'
  - 'netgroup'
  - 'username'
  - 'activedirectory'
  - 'usernameMap'
  - 'kerberos'

RETURNS
- attributes_struct: An XML-RPC struct that includes one or more of the following name:value pairs, depending on the directory service specified by the 'dir_service' parameter:

  - rev: (deprecated) The revision number of the configuration

  - netgroupSource:
    (string) The netgroup source, one of the following:
    - None
    - NIS
    - LDAP
    - File

  - NISdomain: (string) The NIS domain name, returned when NIS is used to obtain user information. If NIS is not used, this parameter is empty.

  - LDAPsecureAccess:
    (string) Whether LDAP secure access is 'enabled' or 'disabled' (the default)

  - dnsDomains: (string) A space-separated list of domains, used for DNS completion

  - dnsDomainDiscovery:
    (string) 'yes' or 'no' (default); if yes, the realm of a host will be determined dynamically using DNS records

  - id: (deprecated) The UUID of the configuration

  - usernameMapSource:

(string) The source for UNIX-to-Windows username conversions,
either 'None' or 'File'

- realm: (string) The Kerberos realm (domain) that contains the principal names
in the Kerberos server database.

- usernamePolled:
(integer) The time of the last username poll,
given in epoch seconds (UNIX timestamp), the number of seconds
since January 1, 1970

- usernameMapPolled:
(integer) The time that the last username map source was polled,
given in epoch seconds (UNIX timestamp), the number of seconds
since January 1, 1970

- usernameConditions:
(string) 'enabled' (default) or 'disabled'; if enabled,
conditions are raised when there are username issues
discovered while processing username directory entries

- usernamePollNow:
(string) Seconds since the UNIX epoch of the last username poll operation

- netgroupPollNow:
(string) Seconds since the UNIX epoch of the last netgroup poll operation

- nfsDomain: (string) The name of an NFSv4 domain used for NFSv4 ACLs,
which get translated into CIFS ACLs for CIFS-enabled vservers

- netgroupPollPeriod:
(string) The polling period for netgroup information.
A setting of 0 (zero) disabled polling; the default is 3600 (1 hour)

- NISserver: (string) A space-separated list of the fully-qualified domain names or IP addresses
of up to three NIS servers. If NIS is not used, this parameter is empty.

- usernamePollPeriod:
(string) The polling period for the username (password and group) files.
A setting of 0 (zero) disables polling; the default is 3600 (1 hour).

- LDAPbasedn: (string) The base name of the LDAP domain. If LDAP is not used, this
parameter is empty.

- usernameSource:
(string) The source for username-to-UID conversions, one of the following:
- None
- NIS
- LDAP
- File
- AD

- LDAPcertificateURI:
(string) The URI for the LDAP domain's security certificate.
If LDAP is not used, this parameter is empty.

- kdcDnsDiscovery:
  (string) If Kerberos is enabled for the cluster,
  whether the key distribution center is automatically searched for;
  the default is 'no'. The 'yes' value of this parameter is only
  valid if the 'kdc' parameter is not set.
- LDAPcredentials:
  (string) 'enabled' or 'disabled' (default); if enabled, LDAP
  connection credentials are enabled and the LDAPbinddn bind name
  is used for secure LDAP connections
- netgroupFileURI:
  (string) The Uniform Resource Identifier from which to read a
  netgroup file

- LDAPbinddn: (string) The bind name used for secure LDAP connections. If LDAP is
  not used, this parameter is empty.

- kdc: (string) The fully-qualified domain name or IP address of the Kerberos
  key distribution center. This parameter is only valid if the 'kdcDnsDiscovery'
  parameter is not set, or is set to 'no'.

- usernameMapPollPeriod:
  (integer) The polling period, in seconds, for the username map file

- LDAPserver: (string) The name of the LDAP server. If LDAP is not used, this parameter is empty.

- LDAPrequireCertificate:
  (string) 'enabled' or 'disabled'; when enabled, LDAP connections will
  require a valid server certificate to be used. If the LDAP server uses
  a self-signed certificate, then LDAPcertificateURI should be the URI of
  the LDAP domain's security certificate

- usernameMapFileURI:
  (string) The URI of a username map file
- usernameGroupURI:
  (string) The URI of a username group file (in /etc/group format)
- ADdomainName: (string) The fully qualified domain name of an Active Directory server

- ADtrusted: (string) The names of trusted Active Directory domains to download user/group
  data from. This value is only used when the usernameSource is set to AD. Allowed
  values are:
  - '' (default; only download user/group data from AD domain cluster is joined to)
  - '*' (download user/group data from all trusted AD domains)
  - A space-separated list of trusted domain names

- usernamePasswdURI:
  (string) The URI of a username password file (in /etc/passwd format)

EXAMPLE

```
print clientHandle.dirServices.get('default')
{'rev': 'd6cb5fd1-3cdd-11e3-a1ce-000c2908d193', 'netgroupSource': 'None
', 'NISdomain': '', 'LDAPsecureAccess': 'disabled', 'dnsDomains': '.ker
beros_co.net kerberos_co.net .company.com company.com', 'dnsDomainDisco
very': '', 'id': 'faa36547-dedf-11e2-8807-000c2994f1f5', 'usernameMapSo
urce': 'None', 'realm': 'KERBEROS_CO.NET', 'usernamePolled': '138264092
7', 'usernameMapPolled': '', 'usernameConditions': 'enabled', 'username
PollNow': '1381945322', 'netgroupPollNow': '1374003266082', 'nfsDomain'
```

: 'test', 'netgroupPollPeriod': '3600', 'NISserver': '', 'usernamePollP
eriod': '3600', 'LDAPbasedn': 'ou=test,dc=company,dc=company,dc=com', '
usernameSource': 'LDAP', 'LDAPcertificateURI': '', 'kdcDnsDiscovery': '
', 'LDAPcredentials': 'disabled', 'netgroupFileURI': ' ', 'LDAPbinddn':
'cifstest@test-company.com', 'kdc': '10.0.0.23', 'usernameMapPollPeriod
': '3600', 'LDAPserver': 'ldap.company.com', 'LDAPrequireCertificate':
'enabled', 'usernameMapFileURI': '', 'usernameGroupURI': 'http://compan
y.com/user_file', 'ADdomainName': 'ad.company.com', 'usernamePasswdURI'
: 'http://company.com/user_file'}

print clientHandle.dirServices.get('default', 'kerberos')
{'realm': 'KERBEROS_CO.NET', 'rev': 'd6cb5fd1-3cdd-11e3-a1ce-000c2908d1
93','kdcDnsDiscovery': '', 'kdc': '10.0.0.23', 'dnsDomains': '.cc.examp
le.com .example.com', 'dnsDomainDiscovery': '', 'id': 'faa36547-dedf-11
e2-8807-000c2994f1f5'}

dirServices.listAdOverrides

NAME

dirServices.listAdOverrides

SYNOPSIS

```
dirServices.listAdOverrides(dir_service) => array_of_structs
```

DESCRIPTION

Returns the Active Directory overrides for all AD domains.

PARAMETERS

- dir_service: (string) The name of a valid directory service.
  Currently the only valid name is 'default'.

RETURNS

- array_of_structs: An array of XML-RPC structs that contain the following
  name:value pairs:

  - netbios: (string) The NetBIOS name of the AD domain
  - fqdn: (string) The fully qualified domain name of the AD domain,
    such as company.domain.com
  - addresses: (string) A space-separated list of one or more IPv4 addresses

EXAMPLE

```
print clientHandle.dirServices.listAdOverrides('default')
[{'netbios': 'ops', 'addresses': '10.0.0.40', 'fqdn': 'ad.company.com'}]
```

NAME
dirServices.loginPoll

SYNOPSIS
dirServices.loginPoll(dir_service) => status

DESCRIPTION
Triggers an immediate poll of the directory service's login source, used to verify cluster credentials.

PARAMETERS
- dir_service: (string) The name of a directory service instance. Currently the only valid name is 'login'.

RETURNS
- status: (string) Either 'success' or a reason for failure. methods.

EXAMPLE
print clientHandle.dirServices.loginPoll('login')
success

NAME
dirServices.modify

SYNOPSIS
dirServices.modify(dir_service, attrs) => status

DESCRIPTION
Modifies one or more attributes on the cluster's directory-service configuration.

PARAMETERS
- dir_service: (string) The name ('default' or 'login') of a directory service.

- attrs: An XML-RPC struct that specifies one or more of the following name:value pairs:

  - LDAPserver: (string) The name of the LDAP server, if LDAP is the netgroup source

  - LDAPbasedn: (string) The base name of the LDAP domain, if LDAP is the netgroup/username source

  - LDAPcertificateURI:
    (string) The URI for the LDAP domain's security certificate, if LDAP is the netgroup source

  - netgroupSource:
    (string) The netgroup source, one of the following:
    - None
    - NIS
    - LDAP
    - File

  - LDAPbasednNetgroup:
    (string) The optional base name of the LDAP domain for netgroup download. LDAPbasedn is used if this parameter is not specified.

  - NISdomain: (string) The NIS domain name, if NIS is the netgroup source

  - netgroupPollPeriod:
    (integer) The polling period for the netgroup information. A setting of 0 (zero) disables polling; the default is 3600 (1 hour).

  - LDAPbinddn: (string) The distinguished name, such as "user@example.com", used to securely bind to the LDAP server,if LDAP is the netgroup source

  - NISserver: (string) A space-separated list of the fully qualified domain names or IP addresses of up to three NIS servers, if NIS is the netgroup source

  - loginSource: (string) The source to use for authenticating user login, either 'Local' or 'Local/LDAP'

  - usernameSource:
    (string) The source for username-to-UID conversions, one of the following:
    - None
    - NIS

- LDAP
- File
- AD

- LDAPbasednUser:
(string) The optional base name of the LDAP domain for username download.
LDAPbasedn is used if this parameter is not specified.

- LDAPbasednGroup:
(string) The optional base name of the LDAP domain for group download.
LDAPbasedn is used if this parameter is not specified.

- nfsDomain: (string) The name of an NFSv4 domain used for NFSv4 ACLs that get translated into CIFS ACLs for CIFS-enabled vservers
- LDAPsecureAccess:
(string) Whether LDAP secure access is 'enabled' or 'disabled'

- ADdomainName: (string) The fully qualified domain name of an Active Directory server to be used by CIFS-enabled vservers

- ADtrusted: (string) The names of trusted Active Directory domains to download user/group data from. This value is only used when the usernameSource is set to AD. Allowed values are:
- '' (default; only download user/group data from AD domain cluster is joined to)
- '*' (download user/group data from all trusted AD domains)
- A space-separated list of trusted domain names

- usernameMapSource:
(string) The source for UNIX-to-Windows username conversions, either 'None' or 'File'
- usernameMapFileURI:
(string) The URI of a username map file
- usernameMapPollPeriod:
(integer) The polling period for the username map file.
A setting of 0 (zero) disables polling.
- usernamePollPeriod:
(integer) The polling period for the username (password and group) files.
A setting of 0 (zero) disables polling; the default is 3600 (1 hour).
- usernameGroupURI:
(string) The URI of a username group file (in /etc/group format)
- usernamePasswdURI:
(string) The URI of a username password file (in /etc/passwd format)

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.dirServices.modify('default', {'netgroupSource': 'None'})
success
```

dirServices.modifyAdOverride

NAME

dirServices.modifyAdOverride

SYNOPSIS

dirServices.modifyAdOverride(dir_service, netbios, fqdn, addresses) => status

DESCRIPTION

Modifies an Active Directory (AD) override for a domain.

PARAMETERS

- dir_service: (string) The name of a valid directory service.
  Currently the only valid name is 'default'.
- netbios: (string) The NetBIOS name of the AD domain
- fqdn: (string) The fully qualified domain name of the AD domain,
  such as company.domain.com
- addresses: (string) A space-separated list of one or more IPv4 addresses

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.modifyAdOverride('default', 'ops',
  'company.domain.com', '10.0.0.40 10.0.0.41')
success
```

NAME
dirServices.netgroupPoll

SYNOPSIS
dirServices.netgroupPoll(dir_service) => status

DESCRIPTION
Triggers an immediate poll of the netgroup source for a specified directory service configuration.

PARAMETERS
- dir_service: (string) The name of a directory service.
Currently the only valid name is 'default'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.dirServices.netgroupPoll('default')
success

dirServices.removeAdOverride

NAME

dirServices.removeAdOverride

SYNOPSIS

dirServices.removeAdOverride(dir_service, netbios, fqdn) => status

DESCRIPTION

Removes an Active Directory (AD) override for a domain.

PARAMETERS

- dir_service: (string) The name of the directory service. Currently the only valid name is 'default'.
- netbios: (string) The NetBIOS name of the AD domain
- fqdn: (string) The fully qualified domain name (for example, company.domain.com) of the AD domain

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.removeAdOverride('default', 'ops', 'ad.company.com')
success
```

dirServices.setLdapPassword

NAME

dirServices.setLdapPassword

SYNOPSIS

dirServices.setLdapPassword(dir_service, new_pass) => status

DESCRIPTION

Updates the password for an LDAP configuration.

PARAMETERS

- dir_service: (string) The name ('default' or 'login') of a directory service
- new_pass: (string) The new password for the LDAP configuration

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.setLdapPassword('default', 'supersecret')
success
```

NAME

dirServices.usernameMapPoll

SYNOPSIS

dirServices.usernameMapPoll(dir_service) => status

DESCRIPTION

Triggers an immediate poll of the directory service's UNIX-to-Windows username map. Use the dirServices.get method to examine the 'usernameMapPolled' attribute.

PARAMETERS

- dir_service: (string) The name of a directory service. Currently the only valid name is 'default'.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.usernameMapPoll('default')
success
```

NAME

dirServices.usernamePoll

SYNOPSIS

dirServices.usernamePoll(dir_service) => status

DESCRIPTION

Triggers an immediate poll of the directory service's username-to-UID source. Use the dirServices.get method to examine the 'usernamePolled' attribute.

PARAMETERS

- dir_service: (string) The name of a directory service. Currently the only valid name is 'default'.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.dirServices.usernamePoll('default')
success
```

NAME
ha.listPartners

SYNOPSIS
ha.listPartners() => haPartner_struct

DESCRIPTION
List the HA partners of all nodes in the cluster.

PARAMETERS
- return_type: (string) How do you want the HA partners to be represented? The options are: 'externalNodeNames','internalNodeNames','haNodeSetUUIDs'
- [verbose] (boolean) Optional. If True, the cfsSpace policy name being used on the HA partners will be described. For the primary 'SmallCache' is the only cfsSpace policy. For the userDataSecondary and userDataMirrorB node the cfsSpace policy can be either 'SecondaryCache' or 'MirrorBCache'.

RETURNS
- haPartner_struct: An XML-RPC struct that includes one or more of the following name:value pairs:

  - userDataPrimary: The node acting as primary user data space.

  - userDataSecondary: The node acting as the mirror for the primary.

  - userDataMirrorB: The node acting as the backup mirror.
    NOTE: Only returned if 3nodeHA is enabled.

  If the verbose boolean is set to True the value of userDataPrimary, userDataSecondary and userDataMirrorB will be a dictionary of type name/ID:cfsSpacePolicy

EXAMPLE
```
print clientHandle.ha.listPartners('externalNodeNames')
{'4800ENG001':  {'userDataMirrorB':   'cloudtest02',
                 'userDataPrimary':   '4800ENG001',
                 'userDataSecondary': 'cloudperf01'},
 'cloudperf01': {'userDataMirrorB':   '4800ENG001',
                 'userDataPrimary':   'cloudperf01',
                 'userDataSecondary': 'cloudtest02'},
 'cloudtest02': {'userDataMirrorB':   'cloudperf01',
                 'userDataPrimary':   'cloudtest02',
                 'userDataSecondary': '4800ENG001'}}
                 OR
print clientHandle.ha.listPartners('externalNodeNames', True)
{'4800ENG001': {'userDataMirrorB':   {'cloudperf01': 'SecondaryCache'},
                'userDataPrimary':   {'4800ENG001':  'SmallCache'},
                'userDataSecondary': {'cloudtest02': 'SecondaryCache'}},
 'cloudperf01': {'userDataMirrorB':   {'cloudtest02': 'MirrorBCache'},
                 'userDataPrimary':   {'cloudperf01': 'SmallCache'},
                 'userDataSecondary': {'4800ENG001':  'MirrorBCache'}},
```

'cloudtest02': {'userDataMirrorB': {'4800ENG001': 'SecondaryCache'},
'userDataPrimary': {'cloudtest02': 'SmallCache'},
'userDataSecondary': {'cloudperf01': 'MirrorBCache'}}}
OR
averecmd --pretty ha.listPartners externalNodeNames
averecmd --pretty ha.listPartners externalNodeNames True

keyMgmt.createKmipServer

NAME
keyMgmt.createKmipServer

SYNOPSIS
keyMgmt.createKmipServer(args) => status

DESCRIPTION
Create a kmip server.

PARAMETERS
- args: An XML-RPC struct containing the following name:value pair
  - name: (string) The administrative name of the kmip server.
  - host: (string) The host IP of the kmip server.
  - port: (string) The port number of the kmip server.
  - [username]: (string) Optional. The username used to login to the kmip server.
  - [password]: (string) Optional. The password associated with the username to the kmip server.
  - CACertName: (string) The CA certificate name used to connect to the kmip server.
  - CACertIssuer: (string) The CA certificate issuer used to connect to the kmip server.
  - CACertSerial: (string) The CA certificate serial number used to connect to the kmip server.
  - clientCertName: (string) The client certificate name used to connect to the kmip server.
  - clientCertIssuer:(string) The client certificate issuer used to connect to the kmip server.
  - clientCertSerial:(string) The client certificate serial number used to connect to the kmip server.
  - [note]: (string) Optional. Any text description added to the certificate

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE

print clientHandle.keyMgmt.createKmipServer({'name':'kmipA', 'host':'10.50.3.94' ,'port':'5696', 'username':'avere', 'password':'1234', 'CACertName':'myca', 'CACertIssuer':'avere_CA', 'CACertSerial':'00', 'clientCertName':'certA', 'clientCertIssuer':'avere_CA', 'clientCertSerial':'1697'})
success

keyMgmt.deleteKmipServer

NAME
keyMgmt.deleteKmipServer

SYNOPSIS
keyMgmt.deleteKmipServer(name, [force]) => status

DESCRIPTION
Deletes a kmip server.

PARAMETERS
- name: (string) The administrative name of the kmip server.
- [force]: (boolean) Optional. Whether the kmip server will be forcibly modified. Default is False.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.keyMgmt.deleteKmipServer(kmipA)
success
```

keyMgmt.getKmipServer

NAME
keyMgmt.getKmipServer

SYNOPSIS
keyMgmt.getKmipServer(name) => kmip_info_struct

DESCRIPTION
Gets a kmip server associated with the cluster.

PARAMETERS
- name: (string) The administrative name of the kmip server.

RETURNS
- kmip_info_struct: (struct) An XML-RPC struct containing the following name:value pair
  - name: (string) The administrative name of the kmip server.
  - host: (string) The host IP of the kmip server.
  - port: (string) The port number of the kmip server.
  - username: (string) The username used to login to the kmip server.
  - password: (string) The password associated with the username to the kmip server.
  - CACertName: (string) The CA certificate name used to connect to the kmip server.
  - CACertIssuer: (string) The CA certificate issuer used to connect to the kmip server.
  - CACertSerial: (string) The CA certificate serial number used to connect to the kmip server.
  - clientCertName: (string) The client certificate name used to connect to the kmip server.
  - clientCertIssuer:(string) The client certificate issuer used to connect to the kmip server.
  - clientCertSerial:(string) The client certificate serial number used to connect to the kmip server.
  - note: (string) Any text description added to the certificate

EXAMPLE
print clientHandle.keyMgmt.getKmipServer(kmipA)
{'username': 'avere', 'CACertName': 'myca', 'clientCertIssuer': 'avere', 'clientCertSerial': '1697', 'name': 'kmipA', 'CACertIssuer': 'avere_CA', 'clientCertName': 'certA', 'port': '5696', 'host': '10.50.3.94', 'CACertSerial': '00', 'corefilers': []}

NAME
keyMgmt.listKmipServers

SYNOPSIS
keyMgmt.listKmipServers() => array_of_struct

DESCRIPTION
Lists kmip servers associated with the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_struct: (array) An array of kmip server structs, including name, host, port, CACert information, clientCert information, associated corefilers.

EXAMPLE
print clientHandle.keyMgmt.listKmipServers()
[{'username': 'avere', 'CACertName': 'myca', 'clientCertIssuer': 'avere', 'clientCertSerial': '1697', 'name': 'kmipA', 'CACertIssuer': 'avere_CA', 'clientCertName': 'certA', 'port': '5696', 'host': '10.50.3.94', 'CACertSerial': '00', 'corefilers': []},
{'username': 'avere', 'CACertName': 'myca', 'clientCertIssuer': 'avere', 'clientCertSerial': '1697', 'name': 'kmipB', 'CACertIssuer': 'avere_CA', 'clientCertName': 'certA', 'port': '5696', 'host': '10.50.3.94', 'CACertSerial': '00', 'corefilers': []}]

keyMgmt.modifyKmipServer

NAME
keyMgmt.modifyKmipServer

SYNOPSIS
keyMgmt.modifyKmipServer(name, args, [force]) => status

DESCRIPTION
Modifies a kmip server.

PARAMETERS
- name: (string) The administrative name of the kmip server.
- args: An XML-RPC struct containing the following name:value pair
  - [name]: (string) Optional. The administrative name of the kmip server.
  - [host]: (string) Optional. The host IP of the kmip server.
  - [port]: (string) Optional. The port number of the kmip server.
  - [username]: (string) Optional. The username used to login to the kmip server.
  - [password]: (string) Optional. The password associated with the username to the kmip server.
  - [CACertName]: (string) Optional. The CA certificate name used to connect to the kmip server.
  - [CACertIssuer]: (string) Optional. The CA certificate issuer used to connect to the kmip server.
  - [CACertSerial]: (string) Optional. The CA certificate serial number used to connect to the kmip server.
  - [clientCertName]: (string) Optional. The client certificate name used to connect to the kmip server.
  - [clientCertIssuer]:(string) Optional. The client certificate issuer used to connect to the kmip server.
  - [clientCertSerial]:(string) Optional. The client certificate serial number used to connect to the kmip server.
  - [note]: (string) Optional. Any text description added to the certificate

- [force]: (boolean) Optional. Whether the kmip server will be forcibly modified. Default is False.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.keyMgmt.modifyKmipServer(kmipA, {'name':'kmipC', 'host':'10.50.3.9' ,'port':'569', 'username':'avere1', 'password':'4566'})
success

NAME
keyMgmt.testKmipServer

SYNOPSIS
keyMgmt.testKmipServer(name, type) => test result

DESCRIPTION
Tests the given kmip server.

PARAMETERS
- name: (string) The administrative name of the kmip server.
- type: (string) The type of testing the kmip server. One of the followings:
- 'cert': Get the server certificate. This tests connectivity and the CA certificate match.
- 'protocolDiscover': Get the supported protocol version. Our Client version is 1.1. This tests connectivity, username authentication, and both CA and client certificate match.
- 'query': Get the supported operations, objecttypes and vendor information from the server. This tests connectivity, username authentication, and both CA and client certificate match.

RETURNS
- test result One of the following:
- (string) 'certificate text': If type is 'cert', the text information of the kmip server certificate.
- (string) 'protocol versions': If type is 'protocolDiscover', the supported version. example: "protocolVersion(major version, minor version)"
- (XML-RPC struct) 'query': If type is 'query', the supported operations, objecttypes and vendor information.

EXAMPLE
print clientHandle.keyMgmt.testKmipServer(kmipA, cert)

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 1926 (0x786)
   Signature Algorithm: sha256WithRSAEncryption
      Issuer: C=US, ST=PA, L=Pittsburgh, O=avere, OU=Lab,
CN=avere_CA/emailAddress=avere@averesystems.com
      Validity
         Not Before: Nov  4 19:29:22 2014 GMT
         Not After : Nov  1 19:29:22 2024 GMT
      Subject: C=US, ST=PA, L=Pittsburgh, O=avere, OU=Lab,
CN=kmip_server/emailAddress=avere@averesystems.com
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
               00:ce:90:28:54:1a:0d:e1:c0:89:d4:61:9f:7f:2d:
                              .....
                              .....
               28:75:50:75:93:0f:b9:5c:71:03:b6:0f:01:f4:91:
               36:a3
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         X509v3 Basic Constraints:
            CA:FALSE
         Netscape Cert Type:
```

SSL Server
Signature Algorithm: sha256WithRSAEncryption
a4:39:55:c7:13:54:0c:41:7e:3b:6a:80:c5:d8:bb:0d:9b:de:
.....
.....
e3:87:90:91:7c:bc:21:5c:d7:c6:b6:cf:14:de:6b:c6:5c:55:
99:53:f8:39

print clientHandle.keyMgmt.testKmipServer(kmipA, protocolDiscover)
protocolVersion(1,1),protocolVersion(1,0)

print clientHandle.keyMgmt.testKmipServer(kmipA, query)
operation = '['QUERY', 'LOCATE', 'DESTROY', 'GET', 'CREATE', 'REGISTER', 'GET_ATTRIBUTES', 'GET_ATTRIBUTE_LIST', 'ADD_ATTRIBUTE', 'MODIFY_ATTRIBUTE', 'DELETE_ATTRIBUTE', 'ACTIVATE', 'REVOKE', 'POLL', 'CANCEL', 'CHECK', 'GET_USAGE_ALLOCATION', 'CREATE_KEY_PAIR', 'REKEY', 'ARCHIVE', 'RECOVER', 'OBTAIN_LEASE', 'REKEY_KEY_PAIR', 'CERTIFY', 'RECERTIFY', 'DISCOVER_VERSIONS', 'NOTIFY', 'PUT']'
vendor = '['Thales e-Security, Inc']'
objecttype = '['CERTIFICATE', 'SYMMETRIC_KEY', 'SECRET_DATA', 'PUBLIC_KEY', 'PRIVATE_KEY', 'TEMPLATE', 'OPAQUE_DATA', 'SPLIT_KEY']'

NAME

maint.isAccessSuspended

SYNOPSIS

maint.isAccessSuspended() => isSuspended

DESCRIPTION

Returns the state of client access to the cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

isSuspended: (boolean) Whether client access to the cluster is suspended (True) or not (False)

EXAMPLE

```
print clientHandle.maint.isAccessSuspended()
False
```

NAME
maint.unsuspendAccess

SYNOPSIS
maint.unsuspendAccess() => status

DESCRIPTION
Unsuspends client access, enabling clients to access the FXT cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.maint.unsuspendAccess()
success
```

mcd.addNode

NAME

mcd.addNode

SYNOPSIS

mcd.addNode(cluster-name, nodes=1) => status

DESCRIPTION

Create nodes and add them to the cluster.

PARAMETERS

- name: (string) Then name of the cluster.
- nodeCnt: (integer) The number of nodes to add to the cluster.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

Add a node to the cluster.

clientHandle.mcd.addNode('cluster-name')

NAME

mcd.addProxyServer

SYNOPSIS

mcd.addProxyServer(settings) => success or failure

DESCRIPTION

PARAMETERS

An XML-RPC struct containing proxy server settings.

- name (string) Name of this proxy server entry.
- note (string) An optional descriptive note.
- networkAddress (string) hostname:portnumber The network address and port number of the proxy server.

The allowable values of the network address will vary from cloud to cloud. In the Google cloud the proxy server host can be an IP address, a fully qualified domain name (FQDN), a host name or a google object instance name. In the Amazon cloud the hostname must be the Amazon EC2 private domain name.

RETURNS

success or failure

EXAMPLE

```
print clientHandle.mcd.createProxyServer({ 'name' : 'proxyServer-1', 'networkAddress' : '10.59.0.10:12345' })
'success'
```

NAME

mcd.addUser

SYNOPSIS

mcd.addUser(user, permission, password) => status

DESCRIPTION

Add an administrative user to the cluster manager.

PARAMETERS

- user: (string) The user name
- permission: (string) One of the following:
  - 'rw' for read-write administrative access, the default
  - 'ro' for read-only administrative access
- password: (string) The user's password

RETURNS

- status: (string) Either 'success' or a reason for failure. Note that this method also returns 'Failed' without a reason given if it is unable to find the user's password record.

EXAMPLE

```
print clientHandle.mcd.addUser('juser','rw','gobbledygookpassword')
User juser already exists
print clientHandle.mcd.addUser('newjuser','rw','gobbledygookpassword')
success
```

NAME

mcd.changePassword

SYNOPSIS

mcd.changePassword(user, oldPassword, newPassword) => status

DESCRIPTION

Changes the password of a cluster manager administrative user.

PARAMETERS

- user: (string) The administrative user name
- oldPassword: (string) The user's old password
- newPassword: (string) The user's new password

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.mcd.changePassword('juser', 'oldSillyPassword', 'newSillyPassword')
success
```

NAME
mcd.create

SYNOPSIS
mcd.create() => status

DESCRIPTION
Add a cluster to the list of clusters being monitored on the multi-cluster dashboard.

PARAMETERS
- [cluster name] (string) Optional. The name of the cluster. If the name is not set then the multi-cluster dashboard will retrieve it from the cluster itself. The cluster will be named 'Cluster' until the name is retrieved. There is no requirement for cluster names to be unique and duplicate names will have a digit appended to the name to uniquify it. The default value is 'Cluster'.
- [login] (string) Optional. The login name to the cluster. The default value is 'admin'.
- [password] (string) Optional. The password associated with the login name. The default value is an empty string ''.
- server (string) The name of the cluster management address or IP.
- note (string) A descriptive note.
- adminStatus (string) The admin status. The possible values are 'ENABLED' and 'DISABLED'. The default value is 'ENABLED'. If the adminStatus value is set to 'DISABLED' the multi-cluster dashboard manager will add the cluster to the list of clusters but it will not poll the cluster for status.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
clusters =  clientHandle.mcd.addNode('cluster-1',4)
'success'
```

mcd.createCluster

NAME

mcd.createCluster

SYNOPSIS

mcd.createCluster(cluster-name, settings) => status

DESCRIPTION

Create a new vFXT cluster in the cloud provider.

PARAMETERS

- name: (string) The name of the cluster. It must be a unique cluster name.
- password: (string) The password for ssh and GUI admin access to the cluster.
- region: (string) The cloud provider region.
- instanceType: (string) In virtual machine instance type.
- nodes: (string) The number of nodes in the cluster.
- cacheSize: (string) The cache size in gigabytes (with disk type identifier where required).
- useProxyServer: (boolean) Optional. When useProxyServer is set to true the cluster will be created behind one of the proxy servers configured in the MCD. For more information see mcd.createProxyServer.
- createCloudFiler: (boolean) Optional. When createCloudFiler is set to true a cloud filer with a junction mount point will automatically be created during vFXT cluster creation. If this option is set to false a cloud filer will not be created. If this option is omitted it will default to true.

Run mcd.guiInfo to view the current set of default values for region, instance type, and cache size.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

Cache Size option examples:

- AWS 1000, 4000, 8000
- GCE 375-local-ssd, 1000-persistent-SSD, 1500-local-SSD, 3000-local-SSD, 4000-persistent-SSD, 8000-persistent-SSD
- Azure 1000, 4000, 8000

The following example will create a three node cluster in the gce environment.

```
clientHandle.mcd.createCluster('tknaus-cluster', { 'password' : 'xxxxxxxx', 'region': 'us-central1-b',
'instanceType' : 'n1-highmem-8', 'nodes' : '3', 'cacheSize' : '2000', useProxyServer: true })
```

mcd.deleteProxyServer

NAME
    mcd.deleteProxyServer

SYNOPSIS
    mcd.deleteProxyServer(name|uuid) => success or failure

DESCRIPTION
PARAMETERS
    - name|uuid   (string) Name or uuid of the proxy server configuration entry to delete.
RETURNS
    success or failure
EXAMPLE
    print clientHandle.mcd.m.deleteProxyServer('proxyServer-5')
    'success'

NAME
mcd.deleteUser

SYNOPSIS
mcd.removeUser(user) => status

DESCRIPTION
Removes an administrative user.

PARAMETER
- user: (string) The administrative user name

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.mcd.removeUser('dante')
success

NAME
mcd.destroyCluster

SYNOPSIS
mcd.destroyCluster() => status

DESCRIPTION
Destroys every node in a cluster managed by the MCD.

PARAMETERS
- name: (string) The name of the cluster

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.mcd.destroyCluster()
    "success"
```

NAME

mcd.fetchJobLog

SYNOPSIS

mcd.fetchJobLog('cluster-name') => a string containing the log of the most recent job.

DESCRIPTION

Return job logs from the vFXT cloud management shepherd.

PARAMETERS

- clusterName: (string) The name of the cluster or cluster uuid.

RETURNS

- A job log report.

EXAMPLE

```
print clientHandle.mcd.fetchJobLog('my-cluster')
07-Aug-2015 20:27:37: 1 of 15: Reading defaults.
07-Aug-2015 20:27:37: 3 of 15: Connected to Google cloud APIs.
07-Aug-2015 20:27:37: Connected to Google cloud APIs
```

mcd.fetchStatus

NAME
mcd.fetchStatus

SYNOPSIS
mcd.list() => clusterStatsDictionary

DESCRIPTION
Retrieve the statistics used by the multi-cluster dashboard to monitor cluster status.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- clusterStatsDictionary - A dictionary where cluster names are the keys and the cluster statistics are the values.

EXAMPLE
clusters = clientHandle.mcd.fetchStats()
clusters['Cluster(1)'] = {u'status': u'UP', u'adminStatus': u'ENABLED', u'name': u'shadow_Cluster', u'lastMaxConditionSeverity': 0, ... }

mcd.getProxyServer

NAME

mcd.getProxyServer

SYNOPSIS

mcd.getProxyServer(name|uuid) => a proxy server config struct

DESCRIPTION

PARAMETERS

The name or uuid of the proxy server configuration to retrieve.

RETURNS

An XML-RPC struct containing proxy server identifiers and network parameters.

- name (string) The name of the proxy server.
- note (string) A descriptive note.
- status (string) Indicates whether or not the proxy server is reachable from the shepherd. The possible return values are 'reachable', 'unreachable', and 'unknown'.
- networkAddress (string) hostname:portnumber The network address and port number of the proxy server.

  The allowable values of the network address will vary from cloud to cloud. In the Google cloud the proxy server host can be an IP address, a fully qualified domain name (FQDN), a host name or a GCE instance name. In the Amazon cloud the hostname must be the Amazon EC2 private domain name.
- id (string) The uuid for identifying this configuration entry.

EXAMPLE

```
print clientHandle.mcd.getProxyServer('proxyServer-5')
{'note': '', 'status': 'unknown', 'networkAddress': 'proxy:12345', 'name': 'proxyServer-5',
 'uuid': '081f18f3-e7cc-11e5-b420-42010a340002'}
```

NAME

mcd.guiInfo

SYNOPSIS

mcd.guiInfo() => struct of cloud configuration parameters

DESCRIPTION

Return the allowable cluster create configuration parameters for the cloud that the vFXT cluster management shepherd is running it.

PARAMETERS

None

RETURNS

- clusterConfigStruct: An XML-RPC struct containing the following name:value pairs:

  - regions: (array) Regions in which you can create vFXT clusters.
  - cacheSizes (array) The set of selectable cluster cache sizes in megabytes (MB).
  - instanceTypes (array) The set of selectable node instance types.
  - machineTypes (structure) The set of selectable machine types that can be used as instance types by region.

EXAMPLE

```
print clientHandle.mcd.guiInfo()
  { 'region' : ['us-central1-b', 'asia-east1-a', 'asia-east1-b', 'asia-east1-c', 'europe-west1-b', 'europe-west1-c', 'europe-west1-d', 'us-central1-a', 'us-central1-c', 'us-central1-f'],
    'cacheSizes' : ['1000','4000'],
    'instanceTypes' : ['n1-highmem-8', 'n1-highmem-32'],
    'machineTypes' : { 'us-central1-a' : ['n1-highmem-8'], us-central-b [...], ... }
  }
```

NAME
mcd.list

SYNOPSIS
mcd.list() => clusterNameArray

DESCRIPTION
List clusters monitored by the multi-cluster dashboard.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- clusterNameArray (array) An array of cluster names.

EXAMPLE
```
print clientHandle.mcd.list()
['Cluster','Cluster(1)','Cluster(2)']
```

mcd.listProxyServers

NAME

mcd.listProxyServers

SYNOPSIS

mcd.listProxyServers => array of proxy server configuration entries

DESCRIPTION

List the proxy servers that can be used to create clusters that are contained within a network proxy environment.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- proxy server definition array.

EXAMPLE

print clientHandle.mcd.listProxyServers()

[{'note': '', 'status': 'reachable', 'networkAddress': 'ff:342', 'name': 'foo', 'uuid': '9ff9e27a-e7c1-11e5-92ca-42010a340002'}, {'note': '', 'status': 'reachable', 'networkAddress': '10.59.0.10:12345', 'name': 'proxyServer-1', 'uuid': 'f84808c5-e7cb-11e5-91da-42010a340002'}, {'note': '', 'status': 'reachable', 'networkAddress': 'proxy:12345', 'name': 'proxyServer-5', 'uuid': '081f18f3-e7cc-11e5-b420-42010a340002'}, {'note': '', 'status': 'reachable', 'networkAddress': 'proxy:12345', 'name': 'proxyServer-5', 'uuid': '502f5242-e7cc-11e5-bdd4-42010a340002'}]

NAME
mcd.listUsers

SYNOPSIS
mcd.listUsers() => array_of_structs

DESCRIPTION
Lists an administrative users of the cluster manager.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - id: (string) The UUID of the user
  - name: (string) The user name
  - perm: (string) One of the following:
    - 'rw' for read-write administrative access
    - 'ro' for read-only administrative access

EXAMPLE
```
print clientHandle.mcd.listUsers()
  {'id': '615eeccf-be47-11e6-be2c-42010a340011', 'perm': 'rw', 'name': 'admin'}
  {'id': '7604eecf-be47-11e6-a501-42010a340011', 'perm': 'ro', 'name': 'rouser'}
  {'id': '7cb5ba66-be47-11e6-b382-42010a340011', 'perm': 'rw', 'name': 'rwuser'}
```

NAME
mcd.modify

SYNOPSIS
mcd.modify('name', newSettings) => status

DESCRIPTION
Change the settings of an existing multi-cluster cluster monitoring entry.

PARAMETERS
- cluster name (string) Name of the entry to be modified.
- newSettings (struct) A dictionary of optional name value pairs.
  - [name] (string) The new name.
  - [login] (string) The login name to the cluster.
  - [password] (string) The password associated with the login name.
  - server (string) The name of the cluster management address or IP.
  - note (string) A descriptive note.
  - adminStatus (string) The admin status. The possible values are 'ENABLED' and 'DISABLED'.
    If the adminStatus value is set to 'DISABLED' the the
    multi-cluster dashboard manager will not poll the cluster for status.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
clusters = clientHandle.mcd.modify('Cluster(1)', {'note' : 'This is a descriptive note.'})
'success'

NAME

mcd.powerdown

SYNOPSIS

mcd.powerdown() => status

DESCRIPTION

Powers down every node in a cluster managed by the MCD. Client access is terminated until power is restored. No committed data is lost. See cluster.powerdown.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.mcd.powerdown()
success
```

NAME
mcd.powerup

SYNOPSIS
mcd.powerup() => status

DESCRIPTION
Powers up every node in a cluster managed by the MCD.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.mcd.powerup()
success
```

NAME
mcd.remove

SYNOPSIS
mcd.remove(cluster_name) => status

DESCRIPTION
Remove a cluster from the list of clusters being monitored on the multi-cluster dashboard.

PARAMETERS
- cluster name (string) Name of the cluster to delete from the multi-cluster dashboard.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
clusters = clientHandle.mcd.remove('Cluster(1)')
'success'

mcd.setCredential

NAME

mcd.setCredential

SYNOPSIS

mcd.setCredential(name, args) => status

DESCRIPTION

Install shepherd administrative and vFXT cluster principal credentials into an Avere shepherd instance running in the Microsoft Azure cloud.

The Microsoft Azure cloud has no infrastructure for VM authentication such as Amazon EC2 roles or Google GCE instance scope. In order for a shepherd instance to create vFXT clusters shepherd administrative and cluster principal credentials must first be installed into the shepherd. The shepherd administrative credential is used to create clusters.

The cluster principal credential is passed to the clusters created by the shepherd. This credential is used by the clusters for moving IP addresses around the cluster.

PARAMETERS

- name: (string) The name of the credential.
- args: An XML-RPC struct containing the following name:value pairs:
  - type: (string) The credential type. The values are either: 'azAdmin' which identifies the shepherd administrative credential or 'azPrincipal' which identifies the cluster principal credential.
  - public: (string) Shepherd service principal name or the cluster application id.
  - private: (string) Shepherd or cluster application password.
  - public2: (string) The shepherd's tenant id or the cluster service principal object id.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.mcd.setCredential('Azure Admin Credentials', { 'type': 'azAdmin',
      'public': '86f4481b-abcd-1234-9e37-767d5153efb5',
      'private': 'ShepherdAppPassword',
      'public2': '6a3a0d20-abcd-1234-9d0d-70b92290ad08' })
'success'
print s.mcd.setCredential('Azure Cluster Principal Credentials', { 'type': 'azPrincipal',
      'public': '9df91da5-abcd-1234-864c-6245f11efee2',
      'private': 'ClusterAppPassword',
      'public2': '9f2d00ae-abcd-1234-aeb0-69463ef1257b' })
'success'
```

NAME
migration.abort

SYNOPSIS
migration.abort(jobId) => status

DESCRIPTION
Deletes the specified migration job.

PARAMETERS
- jobID: (string) The job number of the migration

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.migration.abort('2')
success
```

NAME

migration.create

SYNOPSIS

migration.create(srcFiler, srcExportPath, srcExportSubdir, tgtFiler, tgtExportPath, tgtExportSubdir, [type], [overwriteMode], [startMode], [advancedSettings]) => jobInfoArray

DESCRIPTION

Creates a migration with the given parameters. The migration will not start until migration.start is called, unless the 'startNow' parameter is set to 'startNow'.

PARAMETERS

- srcFiler: (string) The core filer containing the data to be migrated
- srcExportPath: (string) The export on the source core filer
- srcExportSubdir: (string) The subdirectory on the source core filer that will be migrated

- tgtFiler: (string) The core filer to which the data will be migrated
- tgtExportPath: (string) The export on the target core filer
- tgtExportSubdir: (string) The subdirectory on the target core filer where the data will be placed

- [type]: (string) Optional. The migration type, either 'move' (the default) or 'mirror'
- [overwriteMode]: (string) Optional. How the migration treats files that already exist on the target core filer, one of the following:

  - 'always' (the default): Target files are always replaced with copies from the source
  - 'filehandleChanged': Overwrite the target file only if the filehandle, size, or modification time is different
  - 'timeChanged': Overwrite the target file only if the modification time or size is different

  Only use 'filehandleChanged' or 'timeChanged' if the target is known to be a (possibly out of date) copy of the source export.

- [startMode]: (string) Optional. Automatically starts the migration when the value is 'startNow'. Otherwise, the migration will not run until migration.start is executed.

- [advancedSettings]:
  If CIFS is enabled, an XML-RPC struct that specifies one or more of the following name:value pairs:

  - srcShareName: (string) The name of the CIFS share on the source core filer, which should access the same directory as the NFS export 'srcExport'
  - srcAdminUsername: (string) A username that has administrative access to read all CIFS ACLs on the source

  - tgtShareName: (string) The name of the CIFS share on the target core filer, which should access the same directory as the NFS export 'tgtExport'
  - tgtAdminUsername: (string) A username that has administrative access to write all CIFS ACLs on the target

  - [vserver]: (string) Optional. The name of a CIFS-enabled vserver which has the following properties:

- Protocol Transition is enabled in Active Directory
- Constrained delegation is configured for both the source and the target core filers.

If this parameter is not specified, then a CIFS-enabled vserver is selected automatically.

- [sparseFileCheck]: (string) Optional. 'enabled' (default) causes the mover to check each source file for sparsely written data to avoid moving empty file system blocks. 'disabled' may provide a slight cluster performance advantage if you are certain that you don't have sparse files.

- [transferExportPolicy]: (string) Optional. Valid values: 'no' (default) or 'yes'. Controls whether the cluster will transfer the source export policy to the target. The export policy is transferred when either a transition or reverse occurs. You may not specify 'yes' if either the source or the target is a subdirectory in the export since an export policy cannot apply to a subdirectory within an export.

- [node]: (string) Optional. The admin name of the primary node that the mover should run on. If this option is not supplied, or you use the word 'auto' then the default behavior is used. The default behavior is to pick the node with the fewest active migration jobs.

- [syncPolicy]: (string) 'flexible' to allow the mirror to go temporarily out-of-sync if the destination core filer is having problems or 'strict' to keep the mirror synchronized even at the cost of client-side operations. This setting is only valid on mirror job types.

- setRootTgtAclToSrcAcl:
(string) When set to 'yes', sets the target ACL at root to same source ACL at root. The default is 'no'.

RETURNS

- jobInfoArray: An array of the following information about the migration:

- jobID: (string) The job number of the migration

- jobInfo: An XML-RPC struct that specifies one or more of the following name:value pairs about the migration. Note that parameters containing "mass" are deprecated, and only present for backward "Corefiler" should be used for all new applications.

- srcCoreFilerID | srcMassid:
(string) The ID of the core filer containing the data being migrated
- srcCoreFiler | srcMass:
(string) The name of the core filer containing the data being migrated

- node: (string) Name of the node on which the migration is happening

- name: (string) Name of the migration, which will be 'migration<jobID>'

- tgtCoreFiler | tgtMass:
(string) The name of the core filer to which the data is being migrated
- tgtCoreFilerID | tgtMassid:
(string) The ID of the core filer to which the data is being migrated

- overwriteMode: (string) How the migration is treating files that already exist on the target core filer, one of the following:
  - 'always': Target files are always replaced with copies from the source.
  - 'filehandleChanged': Overwrite the target file only if the filehandle, size, or modification time is different.
  - 'timeChanged': Overwrite the target file only if the modification time or size is different.

- note: (string) Any text description added to the migration

- state: (string) The current state of the migration. Possible values are:
  - 'stopped': The migration is stopped. In this state, the migration can be:
    - started, using the migration.start method
    - aborted, using the migration.abort method
  - 'moving': The migration is moving data. In this state, the migration can be:
    - stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
    - paused, using the migration.pause method with the 'stop' parameter set to 'pause'
    - aborted, using the migration.abort method
  - 'synchronized': The source and target are synchronized. In this state, the migration can be:
    - stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
    - reversed, using the migration.transition method with the 'reverse' parameter set to 'reverse'
    - aborted, using the migration.abort method
    - transitioned, using the migration.transition method without the 'reverse' parameter set
  - 'paused': The migration is paused. In this state, the migration can be:
    - (re)started, using the migration.start method
    - stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
    - aborted, using the migration.abort method
  - 'transitioning': The migration job is transitioning from one state to another (for example, from 'moving' to 'complete'). Administrative commands are blocked in this state.
  - 'complete': The migration job is finished. In this state, the migration ID can be dismissed with the migration.dismiss method, removing the migration from the list of migrations.
- type: (string) The migration type, either 'move' or 'mirror'
- status: (string) A more detailed description of the 'state' parameter
- completionTime: (integer) Time at which the migration job ended, in epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970. This is only returned if the job has completed.
- startTime: (integer) Time at which the migration job was last started, in epoch seconds (UNIX timestamp). This is only returned if the job has been started.
- createTime: (integer) Time at which the migration was created, given in epoch seconds (UNIX timestamp)

- srcExport: (string) The export path on the source core filer
- srcSubDir: (string) The subdirectory on the source core filer that is being migrated
- tgtSubDir: (string) The subdirectory on the target core filer where the data is being placed

- status: (string) If the migration creation is complete, either 'success' or a reason for failure. If the creation is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

NOTE: The jobID and jobInfo parameters might be empty if the activity times out.

EXAMPLE

```
print clientHandle.migration.create('thor', '/vol0/a_arrow', 'holding',
  'thor', '/vol0/b_arrow', 'final_store')
['2', {'srcCoreFilerId': '6', 'srcCoreFiler': 'thor', 'node': 'ligo-new
g3', 'name': 'migration2', 'tgtMass': 'thor','tgtCoreFilerId': '6','src
Massid': '6', 'srcSubDir': 'holding', 'overwriteMode': 'always', 'srcMa
ss': 'thor', 'tgtSubDir': 'final_store', 'note': '', 'state': 'stopped'
, 'tgtMassid': '6', 'status':  'waiting to be started', 'tgtCoreFiler':
'thor', 'tgtExport': '/vol0/b_arrow', 'type': 'move', 'createTime': 138
2043286, 'srcExport': '/vol0/a_arrow', 'sparseFileCheck' : 'enabled'}, 'success']
```

NAME
migration.dismiss

SYNOPSIS
migration.dismiss(jobId) => status

DESCRIPTION
Removes a completed migration job from the migrations list.

PARAMETERS
- jobID: (string) The job number of the migration

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.migration.dismiss('2')
success
```

migration.getErrorReport

NAME
migration.getErrorReport

SYNOPSIS
migration.getErrorReport(jobId) => error report

DESCRIPTION
Gets the error report for this jobId, if one exists.

PARAMETERS
- jobID: (string) The job number of the migration

RETURNS
- error report: (string) A string which is the error report file for this job.
An error is returned if no error report exists or the jobId is not found.

EXAMPLE
migration.getErrorReport('3')
==> returns the error report file as a string

migration.getExcludeListByMigrationId

NAME

migration.getExcludeListByMigrationId

SYNOPSIS

migration.getExcludeListByMigrationId(jobId) => excludeList

DESCRIPTION

Gets the exclude list that was specified for the given data management job and returns it as a string in XML format

PARAMETERS

- jobID: (string) The job number of the migration

RETURNS

- excludeList: (string) A string in XML format if there is an exclude list for this data management job, or an empty string if there isn't one.

EXAMPLE

```
migration.getExcludeListByMigrationId('3')
<excludelist>
   <exclude>*.mp3</exclude>
   <exclude>foo/</exclude>
</excludelist>
```

NAME
migration.list

SYNOPSIS
migration.list() => array_of_structs

DESCRIPTION
Returns an array of active migration structs.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- struct_of_structs: An array of XML-RPC structs that include the following name:value pairs. Note that parameters containing "Mass" are deprecated, and only present for backward compatibility. "Corefiler" should be used for all new applications.

- srcCoreFiler | srcMass:
(string) The name of the core filer containing the data being migrated
- srcCoreFilerID | srcMassid:
(string) The ID of the core filer containing the data being migrated

- tgtCoreFiler | tgtMass:
(string) The name of the core filer to which the data is being migrated
- tgtCoreFilerID | tgtMassid:
(string) The ID of the core filer to which the data is being migrated

- node: (string) Name of the node on which the migration is happening

- fileLoggingName: (string) The name of the file where file logging is stored; the default is '.avere_log_migration#'; otherwise, the name given by the user. The file is created in the destination export directory.

- fileLogging: (string) (what files have been transferred) Whether file logging 'yes' or 'no' (default)

- overwriteMode: (string) How the migration is treating files that already exist on the target core filer, one of the following:
- 'always': Target files are always replaced with copies from the source.
- 'filehandleChanged': Overwrite the target file only if the filehandle, size, or modification time is different.
- 'timeChanged': Overwrite the target file only if the modification time or size is different.
- note: (string) Any text description added to the migration

- state: (string) The current state of the migration. Possible values are:
- 'stopped': The migration is stopped. In this state, the migration can be:
- started, using the migration.start method
- aborted, using the migration.abort method
- 'moving': The migration is moving data. In this state, the migration can be:
- stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
- paused, using the migration.pause method with the 'stop' parameter set to 'pause'
- aborted, using the migration.abort method
- 'synchronized': The source and target are synchronized. In this state,

the migration can be:
- stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
- reversed, using the migration.transition method with the 'reverse' parameter set to 'reverse'
- aborted, using the migration.abort method
- transitioned, using the migration.transition method without the 'reverse' parameter set

- 'paused': The migration is paused. In this state, the migration can be:
- (re)started, using the migration.start method
- stopped, using the migration.pause method with the 'stop' parameter set to 'stop'
- aborted, using the migration.abort method

- 'transitioning': The migration job is transitioning from one state to another (for example, from 'moving' to 'complete'). Administrative commands are blocked in this state.
- 'complete': The migration job is finished. In this state, the migration ID can be dismissed with the migration.dismiss method, removing the migration from the list of migrations.

- type: (string) The migration type, either 'move' or 'mirror'
- status: (string) A more descriptive explanation of the 'state' parameter
- exitStatusMsg: (string) Text explaining the 'exitStatus' meaning
- completionTime: (integer) Time at which the migration job ended, in epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970. This is only returned if the job has completed.
- startTime: (integer) Time at which the migration job was last started, in epoch seconds (UNIX timestamp)
- createTime: (integer) Time at which the migration was created, given in epoch seconds (UNIX timestamp)
- name: (string) Name of the migration, "migration#"

- currentTime: (integer) The current time, in epoch seconds (UNIX timestamp)
- srcExport: (string) The export path on the source core filer
- exitDetailsMsg: (string) If exitStatus exists, more information about the error
- srcSubDir: (string) The subdirectory on the source core filer that is being migrated
- tgtSubDir: (string) The subdirectory on the target core filer where the data is being placed
- exitStatus: (integer) If present, identifies an error condition with the migration job, one of the following:
- '-2' for a restartable pause
- '-1' for a mover error
- A positive number for NFS errors such as bad file handle and custom Avere errors related to protocols

EXAMPLE

```
print clientHandle.migration.list()
{'890':
 {'srcCoreFilerId': '78',
  'node': 'company11',
  'fileLoggingName': 'true',
  'fileLogging': True,
  'tgtCoreFiler': '(removed)',
  'overwriteMode': 'filehandleChanged',
  'note': '',
  'state': 'complete',
  'type': 'mirror',
  'status': 'aborted',
  'tgtCoreFilerId': '79',
  'exitStatusMsg': 'Issue encountered during transition',
  'completionTime': 1380830384,
```

'startTime': 1380830073,
'createTime': 1380830070,
'name': 'migration890',
'srcCoreFiler': '(removed)',
'tgtExport': '/vol/vol21',
'currentTime': 1380833011,
'exitStatus': 17,
'srcExport': '/vol/vol20',
'exitDetailsMsg': 'Could not initiate transition due to destination
export being inaccessible. Check destination core filer status and
try again.'},
'882': {'srcCoreFilerId': '78', 'node': 'company11', 'srcMassid': '78',
'fileLoggingName': 'true', 'fileLogging': True, 'tgtCoreFiler': '(remo
ved)', 'overwriteMode': 'filehandleChanged', 'note': '', 'state': 'com
plete', 'type': 'move', 'status': 'move completed', 'tgtCoreFilerId':
'79', 'exitStatusMsg': 'mover running','completionTime': 1380827694,
'startTime': 1380827657, 'createTime': 1380827610, 'name': 'migration8
82', 'srcCoreFiler': '(removed)', 'tgtMass': '(removed)', 'tgtExport':
'/vol/vol5', 'currentTime': 1380833011, 'tgtMassid': '79', 'srcExport'
: '/vol/vol4', 'exitDetailsMsg': 'VcmMoveReq failure'},'893': {'srcCor
eFilerId': '9', 'node': 'company11', 'srcMassid': '9', 'tgtCoreFiler':
'migration-lenparg', 'overwriteMode': 'always', 'note': '', 'state':
'complete', 'type': 'mirror', 'status': 'aborted', 'tgtCoreFilerId':
'4', 'exitStatusMsg': 'mover running', 'completionTime': 1380832232,
'startTime': 1380832182, 'createTime': 1380832179, 'name': 'migration8
93', 'srcCoreFiler': 'migration-grapnel', 'tgtExport':'/vol/loan1_migr
ation2', 'currentTime': 1380833011, 'srcExport':'/vol/loan1_migration1'}}

NAME
migration.modify

SYNOPSIS
migration.modify(jobId, settings) => status

DESCRIPTION
Changes the supplied parameters pertaining to the given data management job ID.
This can only be run when the data management job is stopped or paused.

PARAMETERS
- jobId: (string) The jobId of the data management job that you wish to modify.

- settings: (struct) A struct that may contain one or more of the following:

  - fileLogging: (string) 'yes' to turn it on, 'no' to turn it off

  - fileLoggingName: (string) Provide an optional file name to use for file logging
    If you don't supply a value, then it will use .avere_migration_logN

  - sparseFileCheck: (string) 'enabled' to turn it on, 'disabled' to turn it off

  - node: (string) Name of the primary node where you wish the mover to run.

  - note: (string) Free-formed note field

  - overwriteMode: (string) How the migration is treating files that already exist on the target
    core filer, one of the following:
    - 'always': Target files are always replaced with copies from the source.
    - 'filehandleChanged': Overwrite the target file only if the filehandle,
      size, or modification time is different.
    - 'timeChanged': Overwrite the target file only if the modification
      time or size is different.

  - srcAdminUsername: (string) The CIFS admin user name to use when reading from the source.

  - tgtAdminUsername: (string) The CIFS admin user name to use when writing to the destination.

  - syncPolicy: (string) 'flexible' to allow the mirror to go temporarily out-of-sync if
    the destination core filer is having problems or 'strict' to keep the
    mirror synchronized even at the cost of client-side operations. This setting
    is only valid on mirror job types.

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason
  for failure. If the activity is not complete, the activity UUID, which
  can be used as input for the cluster.getActivity and cluster.abortActivity
  methods.

EXAMPLE
print clientHandle.migration.modify('7', {'node': 'node1', 'fileLogging': 'yes', 'overwriteMode': 'always'})
'success'

NAME
migration.pause

SYNOPSIS
migration.pause(jobId, [stop]) => status

DESCRIPTION
Pauses or stops a specified migration.

PARAMETERS
- jobID: (string) The job number of the migration

- [stop]: (string) Optional. One of the following:
  - 'pause' (the default) continues mirroring, so files that have been migrated stay in sync
  - 'stop' disables mirroring, so that only operations that modify files sent to the source core filer.

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.migration.pause('3')
success
```

migration.setExcludeListByMigrationId

NAME

migration.setExcludeListByMigrationId

SYNOPSIS

migration.setExcludeListByMigrationId(jobId, excludeList) => status

DESCRIPTION

Sets the exclude list for the migration job. You can only set the exclude list when the data management job's operation state is "stopped" or "paused". A data management job may be stopped or paused due to admin action, or due to an error encountered by the mover.

The exclude list is specified as an XML string using the following format:

```
<excludelist>
  <exclude>rule0</exclude>
  <exclude>rule1</exclude>
  ...
  <exclude>ruleN</exclude>
</excludelist>
```

Each rule specifies a path to exclude using the following format:

- If a rule starts with a /, it specifies an absolute path starting with the migration root.
- If a rule does not start with a /, it specifies a relative path from the migration root.
- If the rule ends with a / it only matches directories.
- If a rule does not end with a / it only matches files.
- The star * and question mark ? characters are wild cards.
- * matches N characters. ? matches exactly one character.
- The wild card stops matching at the next / character.

PARAMETERS

- jobID: (string) The job number of the migration
- excludeList: (string) XML formatted string (as above)

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

Excludes all files with .mp3 extension and all directories named foo:

```
migration.setExcludeListByMigrationId('3',
'<excludelist><exclude>*.mp3</exclude><exclude>foo/</exclude></excludelist>')
```

NAME
migration.setNote

SYNOPSIS
migration.setNote(jobId, note) => status

DESCRIPTION
Creates a note about the migration.

PARAMETERS
- jobID: (string) The job number of the migration
- note: (string) A note about the migration, which can then be returned by the migration.list method

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.migration.setNote('3', 'This migration will be
  started for testing.')
success
print clientHandle.migration.list()
{'3':  {'srcCoreFilerId': '6', 'srcCoreFiler': 'thor', 'node': 'ligo3',
'name': 'migration3', 'tgtMass': 'thor', 'tgtCoreFilerId': '6', 'srcMas
sid': '6', 'srcSubDir': 'source', 'overwriteMode': 'always', 'srcMass':
'thor', 'tgtSubDir': 'tgtExportSubdir',
'note': 'This migration will be started for testing.',
'state': 'stopped', 'tgtMassid': '6', 'status': 'waiting to be started'
, 'tgtCoreFiler': 'thor', 'tgtExport': '/vol0/b_arrow', 'type': 'move',
'createTime': 1382119916, 'srcExport': '/vol0/a_arrow'}}
```

migration.start

NAME
migration.start

SYNOPSIS
migration.start(jobId) => status

DESCRIPTION
Starts the migration with the given ID. Use the migration.list method to find the jobID parameter.

PARAMETERS
- jobID: (string) The job number of the migration

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.migration.start('2')
17197c46-381f-11e3-b9ce-000c293a3789
print clientHandle.migration.start('2')
success
```

migration.transition

NAME
migration.transition

SYNOPSIS
migration.transition(jobId, [reverse]) => status

DESCRIPTION
Changes the state of a synchronized migration.

PARAMETERS
- jobID: (string) The job number of the migration

- [reverse]: (string) Optional. How the state of a mirror operation changes:
  - If 'reverse' is specified, the mirror is reversed, and
    the migration is changed to the 'moving' state.
  - If 'reverse' is not specified, the migration stops,
    and is changed to the 'complete' state.

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason
  for failure. If the activity is not complete, the activity UUID, which
  can be used as input for the cluster.getActivity and cluster.abortActivity
  methods.

EXAMPLE
```
print clientHandle.migration.transition('2', 'reverse')
success
```

monitoring.emailSettings

NAME
monitoring.emailSettings

SYNOPSIS
monitoring.emailSettings() => settings_struct

DESCRIPTION
PARAMETERS
- No input parameters are required for this method.

RETURNS
- settings_struct: An XML-RPC struct that contains the following email notification name:value pairs:

  - recipients: (string|struct) Either a comma-separated list of recipient email addresses for all email notifications, or a struct where the keys are the recipients and the values are an array of email notification categories. The email notification categories are:

    - 'all': all system-wide alerts
    - 'NFSCIFS': NFS and CIFS protocol alerts
    - 'clusterServices': FXT internal software, high availability, and cluster network alerts
    - 'coreFiler': core filer alerts
    - 'directoryServices': directory services alerts
    - 'hardwareFailure: hardware failure alerts
    - 'network': client and core filer network alerts
  - rev: (deprecated) The revision number of the email-notification configuration
  - moreContext: (string) Whether email alerts should contain additional lower-severity alerts as context for critical conditions ('true') or not ('false')
  - mailFromAddress: (string) The email address from which the cluster sends email notifications
  - id: (deprecated) The UUID of the email-notification configuration
  - mailServer: (string) The name or IP address of the SMTP server that the cluster uses to send email notifications, optionally followed by :port (defaults to 25).
  - enabled: (boolean) Indicates whether or not sending email is enabled

EXAMPLE
```
print clientHandle.monitoring.emailSettings()
{'recipients': 'support@company.com', 'rev': '846edbb8-1de9-11e
2-8eff-0025907bdb2c', 'moreContext': 'false', 'mailFromAddress':
'you@example.com', 'id': '4a52dbde-e2f7-11e1-960e-0025907a4420',
'mailServer': '10.22.123.456'}
```

monitoring.enableSyslogServer

NAME

monitoring.enableSyslogServer

SYNOPSIS

monitoring.enableSyslogServer(bool) => status

DESCRIPTION

Turn remote syslog forwarding on (True) or off (False)

PARAMETERS

enable: (bool) Whether to enable or disable remote syslog forwarding

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.enableSyslogServer(True)
success
print clientHandle.monitoring.enableSyslogServer(False)
success
```

monitoring.getSyslogServer

NAME
monitoring.getSyslogServer

SYNOPSIS
monitoring.getSyslogServer() => syslog server name string (hostname or hostname:port)

DESCRIPTION
Returns the name or IP address of the syslog server that collects alerts from the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- getSyslogServer: (string) The name or IP address of the syslog server

EXAMPLE
```
print clientHandle.monitoring.getSyslogServer()
syslog.company.com
```

monitoring.getSyslogSettings

NAME
monitoring.getSyslogSettings

SYNOPSIS
monitoring.getSyslogSettings() => settingStruct

DESCRIPTION
Returns the current syslog settings.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- syslogSettingsStruct: An XML-RPC struct that contains name:value configuration setting pairs.

  - remoteSyslogServer (string) The remote syslog server hostname or address
  - enable: (boolean) Whether remote syslog is enabled (True) or not (False)
  - forwarding_options: An XML-RPC struct with the following name:value pairs
    - auth: (boolean) log authentication events
    - xmlrpc: (boolean) log xmlrpc configuration events
    - filesystem: (boolean) log filesystem events

EXAMPLE
```
print clientHandle.monitoring.getSyslogSettings
    remoteSyslogServer = ''
    enable             = 'False'
    forwarding_options = '{'auth': True, 'xmlrpc': True, 'filesystem': True}'
```

monitoring.modifyEmailSettings

NAME

monitoring.modifyEmailSettings

SYNOPSIS

monitoring.modifyEmailSettings(settings_struct) => status

DESCRIPTION

Sets one or more parameters for email notification.

PARAMETERS

- settings_struct: An XML-RPC struct that contains one or more of the following email notification name:value pairs:

  - mailFromAddress: (string) The email address from which the cluster sends email notifications
  - mailServer: (string) The name or IP address of the SMTP server the cluster uses to send email notifications
  - mailServerUser: (string) Optional user id to log into the SMTP server
  - mailServerPassword: (string) Optional password to log into the SMTP server
  - recipients: (string|struct) Either a comma-separated list of recipient email addresses for all email notifications, or a struct where the keys are the recipients and the values are an array of email notification categories. The email notification categories are:

    - 'all': all system-wide alerts
    - 'NFSCIFS': NFS and CIFS protocol alerts
    - 'clusterServices': FXT internal software, high availability, and cluster network alerts
    - 'coreFiler': core filer alerts
    - 'directoryServices': directory services alerts
    - 'hardwareFailure: hardware failure alerts
    - 'network': client and core filer network alerts

  - moreContext: (string) Whether email alerts should contain additional lower-severity alerts as context for critical conditions ('true') or not ('false')

  - enable: (boolean) Whether or not to enable email alerts

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.modifyEmailSettings({'mailFromAddress':
  'cluster1@company.com', 'mailServer': '10.1.22.155', 'recipients':
  'support@company.com', 'moreContext': 'false'})
success

print clientHandle.monitoring.modifyEmailSettings({'mailFromAddress':
  'cluster1@company.com', 'mailServer': '10.1.22.155', 'recipients':
  { 'support@company.com' : [ 'all'], 'network@company.com' : ['NFSCIFS',
  'directoryServices, 'network' ] }, 'moreContext': 'false'})
success
```

monitoring.modifySnmpSettings

NAME

monitoring.modifySnmpSettings

SYNOPSIS

monitoring.modifySnmpSettings(snmp_struct) => status

DESCRIPTION

Sets one or more SNMP parameters.

PARAMETERS

- settings_struct: An XML-RPC struct that contains one or more of the following SNMP name:value pairs:
  - enable: (string) Whether SNMP is enabled on the cluster ('yes' or 'no')
  - contact: (string) A contact name for the cluster
  - location: (string) The physical location of the cluster
  - readCommunityString: (string) The community string used by SNMP for read-only access
  - trapHost: (string) The name or IP address of the SNMP monitor host (manager) to which the cluster sends trap notifications
  - trapPort: (string) The port number of the SNMP monitor host to which the cluster sends trap notifications

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.modifySnmpSettings({'enable':
  'yes', 'contact': 'support@company.com', 'location': 'Ohio'})
success
```

monitoring.setSyslogServer

NAME

monitoring.setSyslogServer

SYNOPSIS

monitoring.setSyslogServer(syslogServer [, forwarding_options]) => status

DESCRIPTION

Sets the name or IP address of a syslog server that is to collect cluster alerts.
Optionally, disable remote syslog for specific classes of events (default is to forward all supported events).

PARAMETERS

syslogServer: (string) The cluster's syslog server, specified as either a fully qualified domain name or an IP address, optionally followed by :port (defaults to 514).
forwarding_options (optional) An XML-RPC struct that contains the following name:value pairs:
- auth (boolean) Set to False to disable forwarding of OS login events.
- xmlrpc (boolean) Set to False to disable forwarding of XMLRPC calls and failures
- filesystem (boolean) Set to False to disable forwarding of FXT filesystem messages (high volume)

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.setSyslogServer('10.12.12.000')
success
print clientHandle.monitoring.setSyslogServer('10.12.12.000:5014')
success
### forwarded OS login and XMLRPC events, but not filesystem events
print clientHandle.monitoring.setSyslogServer('10.12.12.000:5014', '{"auth":True, "xmlrpc":True,
"filesystem":False}')
success
```

NAME
monitoring.snmpSettings

SYNOPSIS
monitoring.snmpSettings() => settings_struct

DESCRIPTION
Lists the SNMP settings for the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- settings_struct: An XML-RPC struct that contains one or more of the following SNMP name:value pairs:
  - enable: (string) Whether SNMP is enabled on the cluster ('yes' or 'no')
  - rev: (deprecated) The revision number of the SNMP-settings configuration
  - trapHost: (string) The name or IP address of the SNMP monitor host (manager) to which the cluster sends trap notifications
  - contact: (string) A contact name for the cluster
  - location: (string) A description of the physical location of the cluster
  - readCommunityString: (string) The community string used by SNMP for read-only access, if needed
  - trapPort: (string) The port number of the SNMP monitor host to which the cluster sends trap notifications
  - id: (deprecated) The UUID of the SNMP-settings configuration

EXAMPLE
```
print clientHandle.monitoring.snmpSettings()
{'enable': 'yes', 'trapHost': '10.12.0.111', 'rev': 'd2318d6b-c97
5-11e2-a2a3-00259014ce64', 'contact': 'support@company.com', 'loc
ation': 'Ohio', 'trapPort': 'e0', 'id': 'd104296a-ea46-11de-841a-
001517c01795'}
```

NAME

monitoring.syslogServer

SYNOPSIS

monitoring.syslogServer() => (syslogServer)

DESCRIPTION

Returns the name or IP address of the syslog server that collects alerts from the cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- syslogServer: (string) The name or IP address of the syslog server

EXAMPLE

```
print clientHandle.monitoring.syslogServer()
syslog.company.com
```

monitoring.syslogServerEnabled

NAME
monitoring.syslogServerEnabled

SYNOPSIS
monitoring.syslogServerEnabled() => bool

DESCRIPTION
Returns True if messages are forwarded to remote syslog servers; False otherwise.
NOTE: Call monitoring.getSyslogSettings() to see which message types are forwarded.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- syslogServer: (string) The name or IP address of the syslog server.
May be of the form hostname:port or IP:port

EXAMPLE
print clientHandle.monitoring.getSyslogServer()
syslog.company.com

NAME

monitoring.testEmail

SYNOPSIS

monitoring.testEmail([recpt, server, from [,user, password]]) => status

DESCRIPTION

Sends a test email notification.

PARAMETERS

This method can be specified with no parameters, in which case it uses the mail-from address, mail server, and recipient list specified for the cluster as shown by the monitoring.emailSettings method.

- [recpt, server, from]:
  Optional. An ordered list of parameters that specify the recipient, mail server, and mail-from address. If one of these parameters is specified, all three parameters must be specified.
- [user, password]:
  Optional. An ordered list of parameters that specify the user and password for the mail server.

RETURNS

- status: (string) Either 'A test email is sent.' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.testEmail()
A test email is sent.
```

monitoring.testSyslog

NAME

monitoring.testSyslog

SYNOPSIS

monitoring.testSyslog() => status

DESCRIPTION

Sends a test event message to the server listed by
the monitoring.syslogServer method.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- status: (string) Either 'A test syslog page is sent' or a reason for failure.

EXAMPLE

```
print clientHandle.monitoring.testSyslog()
A test syslog page is sent
```

network.addLinkAggregate

NAME

network.addLinkAggreagte

SYNOPSIS

network.addLinkAggregate(name,distribution) => status

DESCRIPTION

Adds a network link aggregate

PARAMETERS

- scope: (string) Scope for link aggregate; cluster is the only valid value in this release
- name: (string) Name of the link aggregate
- distribution: (string) Type of link aggregate distribution: LACP or loadbalance

RETURNS

- status: (string) 'success' if the link aggregate was added, otherwise a description of the failure

network.addPortGroup

NAME

network.addPortGroup

SYNOPSIS

network.addPortGroup(name) => status

DESCRIPTION

Adds a network port group

PARAMETERS

- scope: (string) Scope for port group; cluster is the only valid value in this release
- name: (string) Name of the port group

RETURNS

- status: (string) 'success' if the port group was added, otherwise a description of the failure

network.getLinkAggregates

NAME

network.getLinkAggregates

SYNOPSIS

network.getLinkAggregates([filter]) => dict

DESCRIPTION

Get link aggregates

PARAMETERS

- filter: (string) (optional) Regular expression to apply to link aggregate names; names matching the regular expression will be returned

RETURNS

- aggregates: (list of dicts string) Each returned dictionary contains {'scope':'cluster', 'distribution':('LACP' -or- 'loadbalance'), 'ports':csv string}

network.getPortBindings

NAME

network.getPortBindings

SYNOPSIS

network.getPortBindings(scope) => bindings

DESCRIPTION

Get network port bindings

PARAMETERS

- scope: (string) Scope for query; cluster is the only valid value in this release

RETURNS

- bindings: (bindingsStruct) A dict of {groupname:csv-interface-list} listing which interfaces are bound to each named port group.

network.getPortGroups

NAME

network.getPortGroups

SYNOPSIS

network.getPortGroups([filter]) => dict

DESCRIPTION

Get matching port groups

PARAMETERS

- filter: (string) (optional) Regular expression to apply to port group names; names matching the regular expression will be returned

RETURNS

- groups: (list of dicts) keys are 'scope' and 'ports'. In this release, the scope is always 'cluster', and ports is a CSV-string of the interfaces bound to the named port group.

network.modifyPortBindings

NAME

network.modifyPortBindings

SYNOPSIS

network.modifyPortBindings(scope,bindings) => status

DESCRIPTION

Modify network port bindings

PARAMETERS

- scope: (string) Scope for modification; cluster is the only valid value in this release
- bindings: (bindingsStruct) A XMLRPC struct with groupname:interfacelist key/value pairs. The interface list is a comma-separated list of interfaces to bind to the named port group. For example: {"YourGroupName":"e1a,e1b", "anothergroup":"e6a,e6b"}
- force: (boolean) (optional) True to override port link checks, otherwise False; default False

RETURNS

- status: (string) 'success' if the port binding modifications were succesful, otherwise a description of the failure

network.removeLinkAggregate

NAME

network.removeLinkAggregate

SYNOPSIS

network.removeLinkAggregate(name) => status

DESCRIPTION

Remove a link aggregate

PARAMETERS

- name: (string) Name of the port group to remove

RETURNS

- status: (string) 'success' if the link aggregate was removed, otherwise a description of the failure

network.removePortGroup

NAME

network.removePortGroup

SYNOPSIS

network.removePortGroup(name) => status

DESCRIPTION

Remove a port group

PARAMETERS

- name string Name of the port group to remove: ()

RETURNS

- status: (string) 'success' if port group was removed, otherwise a description of the failure

NAME
nfs.addPolicy

SYNOPSIS
nfs.addPolicy(vserverName, policyName) => status

DESCRIPTION
Creates an NFS export policy with the specified policy name on the specified vserver.

NOTE: This method replaced the deprecated nfs.createPolicy method.

PARAMETERS
- vserverName: (string) The name of the vserver on which the export policy should be created
- policyName: (string) The name of the export policy

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.nfs.addPolicy('vserver1', 'user_policy')
success
```

NAME
nfs.addRule

SYNOPSIS
nfs.addRule(vserverName, policyName, scope, filter, permission,
squash, anonuid, suid, subdir, settings_struct) => status

DESCRIPTION
Adds a new NFS export rule to an existing export policy.

PARAMETERS
- vserverName: (string) The name of the vserver on which the export policy is located

- policyName: (string) The name of the export policy on which the new NFS export rule is located

- scope: (string) One of the following:
  - 'host' if the policy should be applied to a specific host
  - 'network' if the policy should be applied to a host on a matching network
  - 'netgroup' if the policy should be applied to a host defined in a netgroup
  - 'default' if the policy should be applied to any host

- filter: (string) Permissible formats depend upon the value of the 'scope' parameter.
  - If the value is 'host', the format may be either an IP address or a fully qualified domain name
  - If the value is 'network', the format may be IP address/netmask (A.B.C.D/N1.N2.N3.N4) or CIDR style notation (A.B.C.D/N)
  - If the value is 'netgroup', the format is @NETGROUP
  - If the value is 'default', the value is always *

- permission: (string) One of the following:
  - 'rw' for read-write access to the export
  - 'ro' for read-only access to the export
  - 'no' for no access to the export

- squash: (string) One of the following:
  - 'root' to prevent clients with local root access from having root privileges on the export
  - 'all' to cause clients to only have read-only access to the export
  - 'no' if any client can access the export

- anonuid: (integer) The user and group ID of the anonymous user. Possible values include:
  -2 or 65534 (nobody)
  -1 or 65535 (no access)
  0 (root without superuser privileges)

- suid: (string) Either 'yes' or 'no' (default); if 'yes', this enables set-user and set-group ID bits for this export

- subdir: (string) 'yes' (default) or 'no'; if 'yes', enables subdirectory mounts within this export

- settings_struct: An XML-RPC struct that contains one or more of the following name:value pairs for authentication:

- authSys: (string) Use NFS auth_sys authentication ('yes' or 'no')
- authKrb: (string) Use Kerberos authentication ('yes' or 'no")

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.nfs.addRule('vserver3', 'user-policy', 'default',
  '*', 'ro', 'root', 0, 'yes', 'no', {'authSys': 'no', 'authKrb': 'yes'})
success
```

NAME
nfs.get

SYNOPSIS
nfs.get(vserverName) => settings

DESCRIPTION
Returns the NFS settings for a vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- settings: An XML-RPC struct that contains the following name:value pairs:
  - rwsize: (integer) Determines the maximum NFS read and write transfer sizes. Possible values are 65536 and 524288.
  - kerberos: (string) Either enables Kerberos ('yes') or not 'no' (the default)
  - extendedGroups (string) Either 'yes' or 'no'; if 'yes', this enables the extended groups feature, which resolves POSIX group membership from an external group database rather than the groups specified in an AUTH_SYS credential

EXAMPLE
```
print clientHandle.nfs.get('vserver1')
{'rwsize': '65536', 'kerberos': 'no', 'extendedGroups': 'no'}
```

nfs.getExportPolicy

NAME
nfs.getExportPolicy

SYNOPSIS
nfs.getExportPolicy(vserverName, [filerName], exportPath) => policyName

DESCRIPTION
Returns the name of an export policy.

PARAMETERS
- vserverName: (string) The name of the vserver on which the export policy is located

- filerName: (string) If the vserver is GNS enabled, this is the core filer on which the export policy is located

- exportPath: (string) The path to the exported directory

RETURNS
- policyName: (string) The name of the policy for the NFS export

EXAMPLE
print clientHandle.nfs.getExportPolicy('vserver1', 'grape', 'vol/juser')
default

nfs.getExportSettings

NAME

nfs.getExportSettings

SYNOPSIS

nfs.getExportSettings(vserverName, filerName, nfsExport) => settings_struct

DESCRIPTION

Returns the NFS export's settings.

PARAMETERS

- vserverName: (string) The name of the vserver on which the export is located
- filerName: (string) The core filer associated with the vserver
- nfsExport: (string) The path to the exported directory

RETURNS

- settingsStruct: An XML-RPC struct that contains the following name:value pairs:
  - policy: (string) The name of the export policy
  - qtree: (string) Whether qtree containers are enabled ('enabled' or 'disabled')

EXAMPLE

```
print clientHandle.nfs.getExportSettings('vserver1', 'grape', '/vol/juser')
{'policy': 'default', 'qtree': 'no'}
```

nfs.listExports

NAME

nfs.listExports

SYNOPSIS

nfs.listExports(vserverName, [filerName]) => array_of_structs

DESCRIPTION

Returns an array of a vserver's NFS exports that are available to clients.

PARAMETERS

- vserverName: (string) The name of the vserver on which the export policy is located.
- filerName: (string) If the vserver is GNS enabled, you must also list the core filer on which the export policy is located.

RETURNS

- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:
  - policy: (string) The name of the export policy
  - path: (string) NFS export path
  - qtree: (string) Whether qtree containers are enabled ('no' or 'yes')

EXAMPLE

```
print clientHandle.nfs.listExports('new_global', 'grapnel')
[{'policy': 'default', 'path': '/vol/user1', 'qtree': 'no'},
 {'policy': 'default', 'path': '/vol/next_user', 'qtree': 'no'},
 {'policy': 'default', 'path': '/vol/last_user', 'qtree': 'no'}}]
```

NAME
nfs.listPolicies

SYNOPSIS
nfs.listPolicies(vserverName) => policyNameArray

DESCRIPTION
Returns the names of all NFS export policies for a vserver.

PARAMETERS
- vserverName: (string) The name of the export policy's vserver

RETURNS
- policyNameArray: (array) The NFS export policy names. The names are strings.

EXAMPLE
```
print clientHandle.nfs.listPolicies('vserver1')
['default', 'user-policy']
```

NAME
nfs.listRules

SYNOPSIS
nfs.listRules(vserverName, policyName, clientFilter) => array_of_structs

DESCRIPTION
Returns the NFS export rules for an export policy.

PARAMETERS
- vserverName: (string) The name of the export policy's vserver
- policyName: (string) The name of the export policy
- clientFilter: (string) The IP address or DNS name of the client for which you want the rules

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - vserver_name: (string) The name of the vserver on which the export policy is located

  - policy: (string) The name of the export policy on which the new NFS export rule is located

  - scope: (string) One of the following:
    - 'host' if the policy should be applied to a specific host
    - 'network' if the policy should be applied to a host on a matching network
    - 'netgroup' if the policy should be applied to a host defined in a netgroup
    - 'default' if the policy should be applied to any host

  - filter: (string) Permissible formats depend upon the value of the 'scope' parameter.
    - If the value is 'host', the format may be either an IP address or a fully qualified domain name
    - If the value is 'network', the format may be IP address/netmask (A.B.C.D/N1.N2.N3.N4) or CIDR style notation (A.B.C.D/N)
    - If the value is 'netgroup', the format is @NETGROUP
    - If the value is 'default', the format is *

  - access: (string) One of the following:
    - 'rw' for read-write access to the export
    - 'ro' for read-only access to the export
    - 'no' for no access to the export

  - squash: (string) One of the following:
    - 'root' to prevent clients with local root access from having root privileges on the export
    - 'all' to cause clients to only have read-only access to the export
    - 'no' if any client can access the export

  - subdir: (string) 'yes' or 'no'; if 'yes', subdirectory mounts are enabled within this export
  - suid: (string) 'yes' or 'no'; if 'yes', set-user and set-group ID bits are enabled for this export
  - anonuid: (integer) The user and group ID of the anonymous user
  - id: (deprecated) The UUID of the configuration

- rev: (deprecated) The revision number of the configuration

EXAMPLE

```
print clientHandle.nfs.listRules('vserver1', 'default')
[{'filter': '*', 'authKrb': 'no', 'suid': 'yes', 'rev': 'f570aaf2-8909-1
1e3-bcd8-000c29159544', 'squash': 'no', 'authSys': 'yes', 'access': 'rw',
'subdir': 'yes', 'scope': 'default', 'id': 'f39073f1-8909-11e3-bcd8-000c2
9159544'}]
```

NAME
nfs.modify

SYNOPSIS
nfs.modify(vserverName, [settings]) => status

DESCRIPTION
Modifies the settings of a vserver.

PARAMETERS
- vserverName: (string) The name of the vserver.

- [settings]: An optional XML-RPC struct that contains the following name:value pairs:
  - rwsize: (integer) Sets the NFS read/write transfer size to '65536' or '524288'.
  - kerberos: (string) Enables ('yes') or disables ('no' - the default) Kerberos
  - extendedGroups (string) yes or no (default); if yes, enables the extended groups feature, which will resolve POSIX group membership from an external group database rather than the groups specified in an AUTH_SYS credential

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
print clientHandle.nfs.modify('vserver1', {'rwsize':'65536', 'kerberos': 'yes', 'extendedGroups': 'no'})
success

nfs.modifyExport

NAME

nfs.modifyExport

SYNOPSIS

nfs.modifyExport(vserverName, filerName, exportPath, settings) => status

DESCRIPTION

Sets an NFS export to use the given settings.

NOTE: This method is used to modify a single export. Use nfs.modifyExports to change several exports at the same time, to the same settings.

PARAMETERS

- vserverName: (string) The name of the vserver on which the export policy is located.
- filerName: (string) The core filer associated with the vserver.
- exportPath: (string) NFS export path

- settings: An XML-RPC struct that contains the following name:value pairs:
  - policy: (string) NFS export policy name
  - qtree: (string) Whether qtree containers are enabled ('no' or 'yes')

NOTE: The qtree element can only be set to 'yes' for paths of the format '/vol/[^\/$]'.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.nfs.modifyExport('vserver1', 'grape', '/vol/juser',
  {'policy': 'default', 'qtree': 'yes'})
success
```

nfs.modifyExports

NAME

nfs.modifyExports

SYNOPSIS

nfs.modifyExports(vserverName, filerName, nfs_exports) => status

DESCRIPTION

Sets NFS export settings.

PARAMETERS

- vserverName: (string) The name of the vserver on which the export policy is located

- filerName: (string) The core filer associated with the vserver

- nfs_exports: An XML-RPC struct that contains the following name:value pairs:
  - path: NFS export path that identifies each export
  - policy: NFS export policy
  - qtree: Whether qtree containers are enabled ('no' or 'yes')

NOTE: The qtree element can only be set to 'yes' for paths of the format '/vol/[^\/$]'.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.nfs.modifyExports('vserver1', 'grape', {'path':
  {'/vol/juser', 'policy': 'default', 'qtree': 'yes'})
success
```

NAME
nfs.modifyPolicy

SYNOPSIS
nfs.modifyPolicy(vserverName, [filerName], exportPath, policy) => status

DESCRIPTION
Sets a vserver's path to use a specified export policy.

PARAMETERS
- vserverName: (string) The name of the vserver
- filerName: (string) If the vserver is GNS enabled, you must also list the core filer on which the export policy is located.
- exportPath: (string) The NFS export path
- policy: (string) The new NFS export policy name

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.nfs.modifyPolicy('vserver1', 'grape', '/vol/juser',
  'user-policy')
success
```

NAME
nfs.modifyRule

SYNOPSIS
nfs.modifyRule(export_uuid, setting_struct_array) => status

DESCRIPTION
Modifies an existing NFS export rule. The 'scope' parameter is required in the setting_struct_array.

PARAMETERS
- rule_uuid: The UUID (id) of the export rule

- setting_struct_array:
An array of XML-RPC structs that contain the following name:value pairs:

- scope: (string) One of the following:
- 'host' if the policy should be applied to a specific host
- 'network' if the policy should be applied to a host on a matching network
- 'netgroup' if the policy should be applied to a host defined in a netgroup
- 'default' if the policy should be applied to any host

- filter: (string) Permissible formats depend upon the value of the 'scope' parameter.
- If the value is 'host', the format may be either an IP address or
a fully qualified domain name
- If the value is 'network', the format may be IP address/netmask
(A.B.C.D/N1.N2.N3.N4) or CIDR style notation (A.B.C.D/N)
- If the value is 'netgroup', the format is @NETGROUP
- If the value is 'default', the format is *

- access: (string) One of the following:
- 'rw' for read-write access to the export
- 'ro' for read-only access to the export
- 'no' for no access to the export

- squash: (string) One of the following:
- 'root' to prevent clients with local root access from having root
privileges on the export
- 'all' to cause clients to only have read-only access to the export
- 'no' if any client can access the export

- anonuid: (integer) The user and group ID of the anonymous user. Possible values include:
-2 or 65534 (nobody)
-1 or 65535 (no access)
0 (root without superuser privileges)

- suid: (string) yes or no (default); if yes, enables enables set-user and
set-group ID bits for this export
- subdir: (string) yes (default) or no; if yes, enables subdirectory mounts within the export

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason
for failure. If the activity is not complete, the activity UUID, which
can be used as input for the cluster.getActivity and cluster.abortActivity
methods.

EXAMPLE

```
print clientHandle.nfs.modifyRule('f390700f1-8909-11e3-bcd8-000c291ss594',
  {'scope': 'default', 'access': 'rw', 'anonuid': '65534'})
success
```

NAME
nfs.removePolicy

SYNOPSIS
nfs.removePolicy(vserverName, policyName) => status

DESCRIPTION
Removes the specified NFS export policy.

PARAMETERS
- vserverName: (string) The name of the vserver
- policyName: (string) The name of the NFS export policy

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.nfs.removePolicy('vserver1', 'user-policy')
success
```

NAME
nfs.removeRule

SYNOPSIS
nfs.removeRule(export_uuid) => status

DESCRIPTION
Removes an NFS export rule.

PARAMETERS
- export_uuid: The UUID of the export rule

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
print clientHandle.nfs.modifyRule('f390700f1-8909-11e3-bcd8-000c291ss594')
success

NAME

nfs.uploadKeytab

SYNOPSIS

nfs.uploadKeytab(vserverName, keytab) => status

DESCRIPTION

Appends a per-vserver keytab to the master keytab for the cluster.

PARAMETERS

- vserverName: (string) The name of the vserver
- keytab: (string) The contents of the vserver's kerberos keytab file, represented as a base64-encoded string

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.nfs.uploadKeytab('vserver1', 'BQIAAABXAAIAC0FWRVJFUUEuTkVU
  AARob3N0ABN4bGcxNy5jYy5hcnJpYWQuY29tAAAAAVKWQu4CABIAIPELQbtoW4+801pdPWovDZf
  9naRRzvxjpclEH0j69qFxdAAAARwACAAtBVkVSRVFBLk5FVAAEaG9zdAATeGxnMTcuY2MuYXJya
  ...
  WFkLmNvbQAAAAFSlkLuAgARABAj2FoySBV2fHQkAR12ZB/NAAAATwACAAtBVkVSRVFBLk5FVAAE')
success
```

NAME
node.allowToJoin

SYNOPSIS
node.allowToJoin(nodeName, [ignoreClientIPCheck]) => status

DESCRIPTION
Enables the specified node to join the cluster.

NOTE: Use the node.listUnconfiguredNodes method to obtain a list of currently available unconfigured nodes. Use the node.remove method to remove a joined node.

PARAMETERS
- nodeName: (string) Name of the node

- [ignoreClientIPCheck]:
(boolean) Optional. Determines if the node will be added whether there are enough IP addresses available in the cluster (True) or not (False)

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.node.allowToJoin('test-node')
success

NAME
node.cpu

SYNOPSIS
node.cpu(nodeName) => percentage

DESCRIPTION
Returns the specified node's CPU usage as a percentage.

PARAMETERS
- nodeName: (string) Name of the node

RETURNS
- percentage: (float) The percentage of CPU being used by the node

EXAMPLE
print clientHandle.node.cpu('busyNode')
55.00000

node.diskPerformance

NAME

node.diskPerformance

SYNOPSIS

node.diskPerformance(nodeName, [period]) => disk performance struct

DESCRIPTION

Returns node disk performance statistics.

PARAMETERS

- nodeName: (string) The name of the node. If node name is cluster then disk performance for all nodes will be returned in a dictionary keyed by the name name.

- [period]: (decimal) Optional. Specifies the period over which average performance information is returned, in minutes. The default value is 1.

RETURNS

- performanceStruct: An XML-RPC struct that contains the following name:value pairs, both for instantaneous and average values by disk. The instantaneous value is taken from the first value of the period. The average values are calculated from samples taken on ten second intervals for the given period.

  - busy_percent: (decimal) The disk busy statistic reported as a percent.
  - outstanding: (decimal) The number of disk OPs queued.
  - read_ops: (decimal) The number of read OPs per second.
  - write_ops: (decimal) The number of write OPs per second.
  - ops_latency_usec: (decimal) The ops latency in micro-seconds.
  - read_bytes: (deciman) The number of read bytes per second.
  - read_latency_usec: (decimal) The disk read latency in micro-seconds.
  - write_bytes: (deciman) The number of write bytes per second.
  - write_latency_usec:(decimal) The disk write latency in micro-seconds.

EXAMPLE

```
print client.node.diskPerformance('busyNode')
{ Drive2 = {
     'busy_percent': {'avg': 0.0, 'instant': 0.0},
     'write_bytes': {'avg': 618425.0, 'instant': 863109.0},
     'read_ops': {'avg': 12.0, 'instant': 21.0},
     'outstanding': {'avg': 0.0, 'instant': 0.0},
     'ops_latency_usec': {'avg': 0.0, 'instant': 0.0},
     'write_ops': {'avg': 4.0, 'instant': 6.0},
     'write_latency_usec': {'avg': 0.0, 'instant': 0.0},
     'read_bytes': {'avg': 566464.0, 'instant': 933232.0},
     'read_latency_usec': {'avg': 0.0, 'instant': 0.0}}
  Drive3 = { ... }
  Drive<N> = { ... }
}
```

NAME
node.get

SYNOPSIS
node.get(nodeName | nodeNameArray) => nodeInfoStruct

DESCRIPTION
Returns detailed information about a list of nodes.

PARAMETERS
One of the following:
- nodeName: (string) The name of the node
- nodeNameArray: (array) An array of node names. The names are strings.

RETURNS
NOTE: Some parameters apply only to advanced-networking configurations. In a non-advanced-networking configuration, the method returns the parameters with empty values.

- nodeInfoStruct: An XML-RPC struct that contains the following name:value pairs:

  - name: (string) The name of the node
  - alternateImage: (string) The alternate software image loaded but not running on the node
  - clusterIPs: (array) An array of structs containing the node's cluster IP addresses
  - rev: (string) The node's revision number
  - state: (string) The node's administrative state, one of the following:
    - 'up' if the node is working as part of the cluster
    - 'down' if the node is not currently accessible by the cluster
    - 'pending' if the node is in the process of moving to another state
    - 'removing' if the node is in the process of being removed
    - 'removing / down' if the node has been flagged for removal using the node.remove method, and is not currently accessible by the cluster
    - 'removing / pending' if the node has been flagged for removal using the node.remove method, but the process has not yet started
    - 'removing / dead' if the node has been either returned to factory specifications or forcibly removed with the node.remove method
    - 'reformatting / down' if all working drives on the node are being returned to factory specifications, and the node is not currently accessible by the cluster. The node automatically rejoins the cluster after this method finishes.
    - 'reformatting / pending' if the node has been flagged for reformatting using the node.reformat method, but the process has not yet started
    - 'suspended' if the node is still part of the cluster, but is not available to clients
    - 'offline' if the node is still part of the cluster, but all services are turned off
    - 'unknown' if the cluster cannot retrieve any information about the node
  - primaryClusterIP: (struct) A struct containing the node's primary cluster IP address
  - clientFacingIPs: (struct) A struct listing the IP addresses for each vserver (client-facing IPs) on the node
  - nodeMgmtIP: (string) The node-management IP address, if one has been set on the cluster using advanced-networking VLAN configuration options
  - id: (string) The node's UUID
  - activeImage: (string) The software image running on the node

  - [ipmi]: (struct) If the node's IPMI card is configured, an XML-RPC struct that contains

one or more of the following name:value pairs:
- configuration: (string) One of the following values:
  - per-node: The following configuration values of the node's IPMI card are returned
    - mode: The IPMI card's IP address type, either 'static' or 'dhcp'
    - address: If 'mode' is 'static', the IP address of the IPMI card, otherwise empty
    - netmask: If 'mode' is 'static', the netmask for the IPMI card, otherwise empty
    - router: If 'mode' is 'static", the default router for the IPMI card, otherwise empty
  - cluster-wide: Use the cluster.get method to obtain the common configuration of the IPMI card of each node in the cluster.

EXAMPLE

```
print clientHandle.node.get('thor1')
{'thor1': {'name': 'eval27', 'alternateImage': 'AvereOS_V3.0.0.4',
'ipmi': {'configuration': 'per-node', 'mode': 'DHCP'},
'clusterIPs': [{'IP': '10.1.27.67'}, {'IP': '10.1.27.68'}],
'rev': '65bad702-0b44-11e3-ab28-001517762565', 'state': 'up',
'primaryClusterIP': {'IP': '10.1.27.67'}, 'clientFacingIPs':
{'vserver1': [{'IP': '10.1.27.86'}], 'vserver2': [{'IP': '10.1.27.89'},
{'IP': '10.1.27.91'}]}, 'id': 'febfd432-a087-11e2-9e13-001517762565',
'activeImage': 'AvereOS_V3.0.0.4'}}
```

node.getHardwareInfo

NAME

node.getHardwareInfo

SYNOPSIS

node.getHardwareInfo(nodeName) => hardwareInfoStruct

DESCRIPTION

Returns hardware information about the specified node.

PARAMETERS

- nodeName: (string) The name of the node (this only takes a single node)

RETURNS

- hardwareInfoStruct: An XML-RPC struct that contains the following name:value pairs about the node hardware:

  - Storage type: (string) Type of storage media on the node
  - Storage: (string) Storage capacity on the node
  - System: (string) FXT model
  - CPU: (string) Number and type of CPUs
  - Serial Number: (string) Serial number
  - Memory: (string) Total amount of RAM, specified by the number and capacity of DIMMs
  - Port e0a: (string) MAC address, status, and speed of the listed port
  - Port e0b: (string) MAC address, status, and speed of the listed port
  - Port e0c: (string) MAC address, status, and speed of the listed port
  - Port e0d: (string) MAC address, status, and speed of the listed port

EXAMPLE

```
print clientHandle.node.getHardwareInfo('specialFXT')
{'Memory': '64 GB (16 x 4096 MB)', 'Storage': '1176 GB data (8 x 147 GB),
250 GB system', 'System': 'FXT2300', 'Port e3d': '00:15:17:76:25:66 UP,
no carrier', 'Port e3a': '00:15:00:76:25:65 UP, active, 1000baseT full-duplex',
'Port e3c': '00:15:00:76:00:67 UP, no carrier', 'Port e3b': '00:15:17:76:25:64
UP, no carrier', 'CPU': '2 (2 x Intel(R) Xeon(R) CPU E5420 2.50GHz)',
'Storage Type': 'Data is stored on HDD disks', 'Slot 2': 'Quad-Port Gigabit
Ethernet Controller', 'Slot 3': 'Quad-Port Gigabit Ethernet Controller',
'Serial Number': '0123456789', 'Slot 4': '1GB NVRAM Controller with Battery
Backup', 'Slot 5': 'Eight-Port SAS Controller', 'Port e2b':'00:15:17:76:23:dc
UP, no carrier', 'NVRAM': '1024 MB installed', 'Port e2a':'00:15:17:76:23:dd
UP, active, 1000baseT full-duplex', 'Port e2d':'00:15:17:76:23:de UP, no
carrier', 'IPMI LAN': '00:30:48:dc:67:34 10.1.41.107 DHCP', 'Port e0a':
'00:30:48:ca:3d:d4 UP, active, 1000baseT full-duplex', 'Port e0b':
'00:30:48:ca:3d:d5 UP, no carrier', 'Port e2c': '00:15:17:76:23:df UP, no
carrier'}
```

NAME

node.getLocatorInfo

SYNOPSIS

node.getLocatorInfo(nodeName) => locatorInfoArray

DESCRIPTION

Returns information about all locators for the specified node.

PARAMETERS

- nodeName: (string) The name of the node (this only takes a single node)

RETURNS

- locatorInfoArray: An array of XML-RPC structs that contains the following name:value pairs about the node locators:

  - Name: (string) Name of the locator
  - Status: (string) Whether the indicator is active ('enabled') or not ('disabled')

EXAMPLE

```
print clientHandle.node.getLocatorInfo('specialFXT')
[{'Status': 'disabled', 'Name': 'Drive2'}, {'Status': 'disabled', 'Name':
'Drive3'}, {'Status': 'disabled', 'Name': 'Drive0'}, {'Status': 'disabled
', 'Name': 'Chassis'}, {'Status': 'disabled', 'Name': 'Drive6'}, {'Status
': 'disabled', 'Name': 'Drive4'}, {'Status': 'disabled', 'Name': 'Drive5'
}, {'Status': 'disabled', 'Name': 'Drive1'}, {'Status': 'disabled', 'Name
': 'Drive7'}]
```

node.getSensorInfo

NAME
node.getSensorInfo

SYNOPSIS
node.getSensorInfo(nodeName) => sensorInfoArray

DESCRIPTION
Returns information about all sensors for the specified node.

PARAMETERS
- nodeName: (string) The name of the node (this only takes a single node)

RETURNS
- sensorInfoArray: An array of XML-RPC structs that contain the following name:value pairs about the sensors on the node:

  - sensor: (string) The name of the sensor.
  - reading: (string) The current reading of the sensor.
  - status: (string) The current status of the sensor.

EXAMPLE
```
print clientHandle.node.getSensorInfo('specialFXT')
[{'status': 'OK', 'reading': '3.288 Volts', 'sensor': '+3.3VSB'},
{'status': 'OK', 'reading': '5.088 Volts', 'sensor': '+5 V'}, {'status':
'OK', 'reading': '5.088 Volts', 'sensor': '+5VSB'}, {'status': 'OK',
'reading': '6889.0 RPM', 'sensor': 'Fan1'}, {'status': 'OK', 'reading':
'6889.0 RPM', 'sensor': 'Fan2'}, {'status': 'OK', 'reading':
'6889.0 RPM', 'sensor': 'Fan3'}, {'status': 'OK', 'reading':
'6889.0 RPM', 'sensor': 'Fan4'}, {'status': 'OK', 'reading': 'Normal',
'sensor': 'CPU1 Temp'}, {'status': 'OK', 'reading': 'Normal', 'sensor':
'CPU2 Temp'}]
```

NAME
node.list

SYNOPSIS
node.list() => nodeNameArray

DESCRIPTION
Lists the names of nodes in the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- nodeNameArray: (array) Names of nodes currently in the cluster. The names are strings.

EXAMPLE
```
print clientHandle.node.list()
['gentoo', 'thor']
```

NAME

node.listUnconfiguredNodes

SYNOPSIS

node.listUnconfiguredNodes() => array_of_structs

DESCRIPTION

Lists nodes that are not configured, including the host that want to join the cluster.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the unconfigured nodes:

  - status: (string) The node's status
  - id: (string) The UUID of the node.
  - name: (string) The name of the node.
  - address: (string) The node's advertised IP address.
  - softwareVersion: (string) The version of Avere OS running the node.

EXAMPLE

```
print clientHandle.node.listUnconfiguredNodes()
[{'status': 'Hardware fault: Memory fault', 'softwareVersion':
'V2.1.0.9.C4', 'id': 'cf300399-a77e-11e2-b1c0-0015177872e5',
'name': 'eval2', 'address': '10.1.1.255'},
{'status': 'wants to join', 'softwareVersion': 'V3.0.0.2.C1',
'id': '93858182-ab96-11e2-8473-000c297bbf24', 'name': 'traini
ng1-node3', 'address': '10.1.1.252'}]
```

NAME

node.manualNodeDiscover

SYNOPSIS

node.manualNodeDiscover(nodeIP, mode) => status

DESCRIPTION

Manually locates and adds a node to the cluster's unjoined node list (returnable using node.listUnconfiguredNodes). The node can then be added to the cluster. This is useful if the cluster setup does not allow multicast.

PARAMETERS

- nodeIP: (string) The IP address of the node to located. It must be on the same subnet as the cluster IPs.
- mode: (string) Optional. The Discover mode, either "n" or "c". The default value is "n", to use on any of the cluster nodes. "c" mode is used on unconfigured node.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.node.manualNodeDiscover('10.1.4.0')
success
```

NAME
node.modifyIPMI

SYNOPSIS
node.modifyIPMI(nodeName, mode, [staticOptions]) => status

DESCRIPTION
Configures a node's IPMI card.

PARAMETERS
- nodeName: (string) Name of the node whose IPMI card is to be configured.
- mode: (string) Configuration mode for the IPMI card, one of the following:
  - 'none' for no configuration. This value clears the configuration on a cluster whose IPMI cards have previously been configured.
  - 'static' configures the IPMI card with a static IP address
  - 'dhcp' configures the IPMI with a DHCP-assigned IP address

- staticOptions: An XML-RPC struct that contains the following name:value pairs. If the 'mode' parameter is specified as 'none' or 'dhcp', this parameter is optional and has no effect.
  - address: (string) IP address for the IPMI card
  - netmask: (string) The netmask for the IPMI card's IP addresses
  - router: (string) The default router for the IPMI card's IP addresses

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.node.modifyIPMI('newg0', 'dhcp')
success
```

NAME
node.offline

SYNOPSIS
node.offline(nodeName) => status

DESCRIPTION
Takes the specified node offline. Taking a node offline is the first step in various testing and maintenance procedures. This operation stops all services on the node, but does not shut down the node.

You can put the node back online by using the node.online method.

PARAMETERS
- nodeName: (string) The name of the node to be put offline

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.node.offline('gentoo')
success
```

NAME
node.online

SYNOPSIS
node.online(nodeName) => status

DESCRIPTION
Puts a node online that was previously taken offline with the node.offline method.

PARAMETERS
- nodeName: (string) The name of the node to be put back online

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.node.online('training')
xmlrpc.Fault: <Fault 105: 'Node is not offlined administratively'>
print clientHandle.node.online('gentoo')
success
```

node.performance

NAME

node.performance

SYNOPSIS

node.performance(nodeName, [period]) => performanceStruct

DESCRIPTION

Returns node performance information.

PARAMETERS

- nodeName: (string) The name of the node

- [period]: (decimal) Optional. Specifies the period for which performance information is returned, in minutes. The default value is 1.

RETURNS

- performanceStruct: An XML-RPC struct that contains the following name:value pairs, both for 'Instantaneous' and <period> min average', the average performance for the given period.

  - Latency MS: (decimal) Average latency of each operation, in microseconds
  - Cache Hit Rate: (integer) Average number of client requests handled by the cluster, instead of being forwarded to the core filer
  - Ops Per Second: (integer) Average number of operations per second

EXAMPLE

```
print clientHandle.node.performance('thor', 5)
{'5 min average': {'Latency MS': '2.0', 'Cache Hit Rate': 35,
'Ops Per Second': 31}, 'Instantaneous': {'Latency MS': '5.0',
'Cache Hit Rate': 22, 'Ops Per Second': 28}}
```

NAME
node.powerdown

SYNOPSIS
node.powerdown(nodeName) => status

DESCRIPTION
Powers down the specified node. Client access can be temporarily disrupted, depending on the state of other nodes in the cluster; no committed data is lost.

PARAMETERS
- nodeName: (string) Either 'local' or the name of the node

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.node.powerdown('thor')
success

NAME
node.reboot

SYNOPSIS
node.reboot(nodeName) => status

DESCRIPTION
Reboots the specified node. Client access is temporarily disrupted; no committed data is lost.

PARAMETERS
- nodeName: (string) Either 'local' or the name of the node

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.node.reboot('gentoo')
success

NAME
node.remove

SYNOPSIS
node.remove(nodeName, [force]) => status

DESCRIPTION
Removes a node from the cluster.

NOTE: Before this method can return successfully, you must enable the maintenance level of XML-RPC APIs using the system.enableAPI method.

PARAMETERS
- nodeName: (string) The name of the node

- [force]: (boolean) Optional. Whether the node will be forcibly removed (True) or not (False - the default). If True, some client connections may be interrupted, and data can be lost.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.system.enableAPI('maintenance')
success
print clientHandle.node.remove('myOldNode', True)
success
```

NAME
node.rename

SYNOPSIS
node.rename(nodeName, newName) => status

DESCRIPTION
Renames the specified node.

PARAMETERS
- nodeName: (string) The current name of the node
- newName: (string) The new name for the node

RETURNS
- status: (string) Either 'success' or a reason for failure. The method will also return 'success' if the new name is the same as the old name.

EXAMPLE
```
print clientHandle.node.rename('nodeName1', 'nodeName2')
success
print clientHandle.node.rename('nodeName2', 'nodeName2')
success
```

node.restartService

NAME
node.restartService

SYNOPSIS
node.restartService(nodeName) => status

DESCRIPTION
Restarts all services on the specified node. Client access is temporarily disrupted; no committed data is lost.

PARAMETERS
- nodeName: (string) Either 'local' or the name of the node

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.node.restartService('slowNode')
success
```

NAME
node.safeRemoveTest

SYNOPSIS
node.safeRemoveTest(nodeName) => isSafe

DESCRIPTION
Runs a test on the cluster's high availability services to verify that it is safe to forcibly remove ("force remove") a node.

Removing a node from the cluster that is down or out of communication with the cluster requires a force remove. A user may also force remove a node that is up and in the process of being removed in order to speed the removal process.

PARAMETERS
- nodeName: (string) The name of the node to test

RETURNS
- isSafe: (boolean) Whether the node is safe to force remove (True) or not (False). This method will return False if high availability is off.

NOTE: Force removing a node when 'isSafe' is False will be likely to result in user DATA loss, metadata corruption (files and directories are missing), or cluster configuration corruption. DATA loss is also likely if a node is force removed and high availablity is off.

EXAMPLE
```
print clientHandle.node.safeRemoveTest('grape')
False
```

node.setLocatorLed

NAME

node.setLocatorLed

SYNOPSIS

node.setLocatorLed(nodeName, locatorName, state) => status

DESCRIPTION

Set a new state for a locator LED on the specified node.

PARAMETERS

- nodeName: (string) The name of the node (this only takes a single node)
- locatorName: (string) The name of a locator
- state: (string) Whether the indicator should be active ('enabled') or not ('disabled')

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.node.setLocatorLed('specialFXT', 'Drive0', 'enabled')
success
```

NAME
node.suspend

SYNOPSIS
node.suspend(nodeName) => status

DESCRIPTION
Suspends (that is, stops all services on) the specified node.

PARAMETERS
- nodeName: (string) The name of the node

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.node.suspend('slowNode')
*** system.login exception: [Errno 60] Operation timed out
print clientHandle.node.suspend('slowNode2')
success
```

NAME
node.unsuspend

SYNOPSIS
node.unsuspend(nodeName) => status

DESCRIPTION
Unsuspends (that is, restarts all services on) a node
that was previously suspended with the node.suspend method.

PARAMETERS
- nodeName: (string) The name of the node

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.node.suspend('slowNode')
success
```

node.updateHardwareInfo

NAME
node.updateHardwareInfo

SYNOPSIS
node.updateHardwareInfo() => status

DESCRIPTION
Updates the hardware information returned by node.getHardwareInfo() for the nodes in the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.node.updateHardwardInfo()
success
```

NAME
node.uptime

SYNOPSIS
node.uptime(nodeName) => uptime

DESCRIPTION
Returns the uptime for the specified node.

PARAMETERS
- nodeName: (string) The name of the node

RETURNS
- uptime: (string) How long the node has been running, given in the form of "dd days hh:mm:ss"

EXAMPLE
```
print clientHandle.node.uptime('myNode')
16 days 21:41:40
```

NAME

snapshot.create

SYNOPSIS

snapshot.create(corefiler, name, [options]) => status

DESCRIPTION

Creates a snapshot

PARAMETERS

- corefiler: (string) The name of a cloud core filer
- name: (string) The name of the snapshot
- options: (struct) Optional. An XML-RPC struct that must include one or more of the following name:value pairs:
  - note: (string) The descriptive text of this snapshot.
  - adminState: (string) The administrative state of this snapshot. Possible values are:
    - hold
    - none

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.snapshot.create('s3CloudFiler', 'mySnapShot', {'note':'A descriptive note', 'adminState':'hold'});
success
```

ERRORS

- Cannot find core filer s3CloudFiler.
- The snapshot name 'mySnapShot' already exists.

NAME
snapshot.createPolicy

SYNOPSIS
snapshot.createPolicy(name,policy,[options]) => status

DESCRIPTION
Creates a snapshot policy. Snapshot policies are used to control the creation and limits of snapshots on cloud core filers.

PARAMETERS
- name: (string) The name of the snapshot policy

- policy: An XML-RPC struct that contains the following name:value pairs, which define the snapshot schedules and limit policies:

  - hourly: (struct) The time periods and limit policy for hourly snapshots. Hourly snapshots will be taken on the hour as specified. If limit number is set then it will be the number of snapshots taken during the day.

  - daily: (struct) The limit policy for daily snapshots. Daily snapshots are taken at 3:00AM in the local timezone.

  - weekly: (struct) The limit policy for weekly snapshots. Weekly snapshots are taken at 3:00AM in the local timezone.

  - monthly: (struct) The limit policy for monthly snapshots. Monthly snapshots are taken at 3:00AM in the local timezone.

- options: (struct) Optional. Additional snapshot policy configuration parameters, currently consisting of the following name:value pair:

- note: (string) Optional. User-supplied descriptive text
- defaultTime: (string) Optional. The default hour to be used for all monthly, weekly, and daily snapshots. This must be a number between 0 to 23, including 0 and 23. If it's not set, it will be set to 3 by default.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.snapshot.createPolicy('examplePolicy',
{'hourly' : {'time' : '3,6,12,18', 'limit' : '4'}, 'daily' : {'limit' : '20'}, 'weekly' :
{'limit' : '8'}, 'monthly' : {'limit' : '3'} },
{'note':'A descriptive note'});
success
```

ERRORS
- The snapshot policy name 'examplePolicy' is already in use.
- The schedule period must be one of 'hourly', 'daily', 'weekly', or 'monthly'.
- The hourly snapshots must be made on the hour. For example: 7 for 7 AM and 23 for 11 PM.

NAME

snapshot.delete

SYNOPSIS

snapshot.delete(corefiler, name) => status

DESCRIPTION

Delete a snapshot.

PARAMETERS

- corefiler: (string) The name of the snapshot's cloud core filer
- name: (string) The name of the snapshot

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.snapshot.delete('s3CloudFiler', 'mySnapShot')
success
```

ERRORS

- Cannot find core filer s3cloudfiler.
- Cannot find snapshot mySnapShot.

NAME

snapshot.deletePolicy

SYNOPSIS

snapshot.deletePolicy(name, [force]) => status

DESCRIPTION

Delete a snapshot policy.

The default snapshot policy cannot be deleted. However, deleting it will reset it to its original values.

PARAMETERS

- name: (string) The name of the snapshot policy
- force: (boolean) Optional. If set to true, delete the policy even if it is associated to cloud core filers.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.snapshot.deletePolicy('examplePolicy');
success
print clientHandle.snapshot.deletePolicy('examplePolicy1', true);
success
```

ERRORS

- The 'examplePolicy' is in use by cloud filer 'able' and 'baker'. Set 'force' to true to delete it anyway. The snapshot policy for cloud filers using this policy will be changed to 'none'.

NAME
snapshot.getFilerPolicy

SYNOPSIS
snapshot.getFilerPolicy(corefiler) => struct

DESCRIPTION
List the snapshot policy for a filer. Snapshot policies are used to control the creation and limit of snapshots on cloud core filers.

PARAMETERS
- corefiler: (string) The name of the cloud core filer.

RETURNS
- struct: A XML-RPC struct containing the following name:value pairs:
  - name: (string) The name of the snapshot policy.
  - note: (string) A descriptive note (if it was set).
  - policy: (struct) A dictionary defining the snapshot schedules and limit policies.
    The snapshot scheduling periods are defined as:
    hourly, daily, weekly, monthly
    - hourly: (struct) A dictionary defining the time periods and limit policy for hourly snapshots. Hourly snapshots will be taken on the hour as specified in time from 0 to 23.
      Limit is the number of snapshots taken during the day.
    - daily: (struct) A dictionary limit policy for daily snapshots.
      Daily snapshots are taken at 3:00AM in the local timezone.
    - weekly: (struct) A dictionary limit policy for weekly snapshots.
      Weekly snapshots are taken at 3:00AM in the local timezone.
    - monthly: (struct) A dictionary limit policy for monthly snapshots.
      Monthly snapshots are taken at 3:00AM in the local timezone.

EXAMPLE
```
print clientHandle.snapshot.getFilerPolicy('myCloudFiler');
policy = '{'Monthly': {'limit': '3'}, 'Hourly': {'limit': '4', 'time': '1,10,11,12,14,15'}, 'Daily': {'limit': '3'}, 'Weekly': {'limit': '2'}}'
note   = 'this is a policy'
name   = 'mypolicy'
```

snapshot.list

NAME

snapshot.list

SYNOPSIS

snapshot.list(corefiler) => array_of_structs

DESCRIPTION

List all snapshots for a filer defined in the cluster.

PARAMETERS

- corefiler: (string) The name of a cloud core filer

RETURNS

- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - corefiler: (string) The name of the snapshot's cloud core filer
  - name: (string) The name of the snapshot

  - state: (string) The state this snapshot is in. The possible values are:
    - init
    - deleting
    - active

  - type: (string) The Admin type of this snapshot. Possible values are:
    - Monthly
    - Weekly
    - Daily
    - Hourly
    - Custom

  - createTime: (string) The snapshot creation time
  - id: (string) The snapshot ID
  - adminState: (string) The administrative state of this snapshot. Possible values are:
    - hold
    - none
  - Visible: (string) The number of bytes a 'du' would return in this snapshot
  - Written: (string) The number of bytes written to this snapshot
  - FreeOnDel: (string) The number of bytes that would be free if you delete this snapshot
  - note: (string) The descriptive text of this snapshot.

EXAMPLE

print clientHandle.snapshot.list('coreFiler');

[{'name': 'Hourly.2014-08-28_1200', 'Visible': '48080', 'Written': '0', 'FreeOnDel': '544', 'adminState': 'none', 'id': '4294966681', 'note': '', 'state': 'active', 'type': 'Hourly', 'createtime': '1409241639'}
{'name': 'Hourly.2014-08-28_1100', 'Visible': '48080', 'Written': '27664', 'FreeOnDel': '0', 'adminState': 'hold', 'id': '4294966683', 'note': '', 'state': 'active', 'type': 'Hourly', 'createtime': '1409238029'}]

NAME
snapshot.listAll

SYNOPSIS
snapshot.listAll() => struct of array_of_structs

DESCRIPTION
List all snapshots defined in the cluster.

PARAMETERS
- None

RETURNS
- struct of array_of_structs:
Key by filer name

- array_of_structs: An array of XML-RPC structs that contain the following
name:value pairs:

- corefiler: (string) The name of the snapshot's cloud core filer
- name: (string) The name of the snapshot

- state: (string) The state this snapshot is in. The possible values are:
- init
- deleting
- active

- type: (string) The Admin type of this snapshot. Possible values are:
- Monthly
- Weekly
- Daily
- Hourly
- Custom

- createTime: (string) The snapshot creation time
- id: (string) The snapshot ID
- adminState: (string) The administrative state of this snapshot. Possible values are:
- hold
- none
- Visible: (string) The number of bytes a 'du' would return in this snapshot
- Written: (string) The number of bytes written to this snapshot
- FreeOnDel: (string) The number of bytes that would be free if you delete this snapshot
- note: (string) The descriptive text of this snapshot.

EXAMPLE
print clientHandle.snapshot.listAll();
c1 = [{'name': 'Hourly.2014-08-28_1200', 'Visible': '48080', 'Written': '0', 'FreeOnDel': '544', 'adminState': 'none', 'id': '4294966681', 'note': '', 'state': 'active', 'type': 'Hourly', 'createtime': '1409241639'}
{'name': 'Hourly.2014-08-28_1100', 'Visible': '48080', 'Written': '27664', 'FreeOnDel': '0', 'adminState': 'hold', 'id': '4294966683', 'note': '', 'state': 'active', 'type': 'Hourly', 'createtime': '1409238029'}]

NAME
snapshot.listPolicies

SYNOPSIS
snapshot.listPolicies() => array_of_structs

DESCRIPTION
Lists snapshot policies defined in the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - name: (string) The name of the snapshot policy
  - note: (string) User-supplied descriptive text

  - corefilers: (array) An array of cloud core filers using this snapshot policy

  - policy: An XML-RPC struct that contains the following name:value pairs, which define the snapshot schedules and limit policies. An empty value indicates no change:

    - hourly: (struct) The time periods and limit policy for hourly snapshots. Hourly snapshots will be taken on the hour as specified in time from 0 to 23. Limit is the number of snapshots taken during the day.

    - daily: (struct) The limit policy for daily snapshots. Daily snapshots are taken at 3:00AM in the local timezone.

    - weekly: (struct) The limit policy for weekly snapshots. Weekly snapshots are taken at 3:00AM in the local timezone.

    - monthly: (struct) The limit policy for monthly snapshots. Monthly snapshots are taken at 3:00AM in the local timezone.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.snapshot.listPolicies();
[{'name' : 's3SnapshotPolicy', 'cloudFilers' : [ 'cane', 'able'],
'policy' : { 'daily' : { 'limit' : 10 }, 'weekly' : { 'limit' : 23}} },
{'name' : 'AnotherSnapshotPolicy', 'cloudFilers' : [ 'charlie', 'baker'],
'policy' : { 'hourly' : { 'time' : [0,3,6,9,12,15,18,21] : 'limit' : 3 } ,
'weekly' : { 'limit' : 23} } }]
success
```

NAME

snapshot.modify

SYNOPSIS

snapshot.modify(corefiler, name, modifiables) => status

DESCRIPTION

Modify a snapshot.

PARAMETERS

- corefiler: (string) The name of the cloud core filer that this snapshot is located in.
- name: (string) The name of the snapshot.
- modifiables: An XML-RPC struct that must include one or more of the following name:value pairs:
  - name: (string) The name of the snapshot
  - adminState: (string) The administrative state of this snapshot. Possible values are:
    - hold
    - none
  - note: (string) The descriptive text of this snapshot.

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.snapshot.modify('s3CloudFiler', 'mySnapShot', {'name':'newName', 'adminState':'hold'});
success
```

ERRORS

- Cannot find core filer s3cloudfiler.
- Cannot find snapshot mySnapShot.

snapshot.modifyFilerPolicy

NAME

snapshot.modifyFilerPolicy

SYNOPSIS

snapshot.modifyFilerPolicy(corefiler,policyName) => status

DESCRIPTION

Modifies the snapshot policy for a filer. Snapshot policies are used to control the creation and limit of snapshots on cloud core filers.

PARAMETERS

- corefiler: (string) The name of the cloud core filer.
- policyName: (string) The name of the snapshot policy. Empty or None means delete the policy for this cloud core filer

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.snapshot.modifyFilerPolicy('myCloudFiler', 'examplePolicy');
success
print clientHandle.snapshot.modifyFilerPolicy('myCloudFiler', '');
success
```

snapshot.modifyPolicy

NAME
snapshot.modifyPolicy

SYNOPSIS
snapshot.modifyPolicy(name,policy,options) => status

DESCRIPTION
Modifies a snapshot policy. Snapshot policies are used to control the creation and limit of snapshots on cloud core filers.

PARAMETERS
- name: (string) The name of the snapshot policy

- policy: An XML-RPC struct that contains the following name:value pairs, which define the snapshot schedules and limit policies. An empty value indicates no change:

  - hourly: (struct) The time periods and limit policy for hourly snapshots. Hourly snapshots will be taken on the hour as specified in time from 0 to 23. Limit is the number of snapshots taken during the day.

  - daily: (struct) The limit policy for daily snapshots. Daily snapshots are taken at 3:00AM in the local timezone.

  - weekly: (struct) The limit policy for weekly snapshots. Weekly snapshots are taken at 3:00AM in the local timezone.

  - monthly: (struct) The limit policy for monthly snapshots. Monthly snapshots are taken at 3:00AM in the local timezone.

- options: (struct) Optional. Additional snapshot policy configuration parameters, currently consisting of the following name:value pair:

- note: (string) Optional. User-supplied descriptive text
- defaultTime: (string) Optional. The default hour to be used for all monthly, weekly, and daily snapshots.
This must be a number between 0 to 23, including 0 and 23.
If it's empty, it will be set to 3 by default.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.snapshot.modifyPolicy('examplePolicy', { 'hourly' : { time : [0,3,6,9,12,15,18,21] }});
success

ERRORS
- The schedule period must be one of 'hourly', 'daily', 'weekly', or 'monthly'.
- The hourly snapshots must be made on the hour. For example: 7 for 7 AM and 23 for 11 PM.
- The time for taking a daily, weekly, or hourly is 3:00 AM. It may not be changed using this API.

NAME
stats.activeClients

SYNOPSIS
stats.activeClients(connectorName, connectorType) => array_of_structs

DESCRIPTION
Returns an array of client connections.

PARAMETERS
- connectorName: (string) The name of the vserver or node used for the connection statistics
- connectorType: (string) One of the following:
  - 'vserver' to return active connections to each vserver
  - 'node' to return active connections to each node

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the connections:

  - node: (string) The name of the node to which the active client is connected. If the 'type' requested is 'vserver', this parameter will be empty.
  - vserver: (string) The name of the vserver to which the active client is connected. If the 'type' requested is 'node', this parameter will be empty.
  - clientAddress: (string) The IP address of the active client
  - clientType: (string) The type of the client. Current values are 'cifs', 'nfs', or 'unknown'.
  - numConnections: (integer) The number of connections from the active client
  - avereAddress: (string) The cluster address to which the active client is connected

EXAMPLE
```
print clientHandle.stats.activeClients('node_finder1','node')
[{'node': 'node_finder1',
  'vserver': 'gns',
  'clientAddress': 'company.com (10.1.1.14)',
  'numConnections': 3,
  'avereAddress': '10.1.26.144'},
{'node': 'node_finder1', 'vserver': 'gns', 'clientAddress': 'company.com (10.
 1.2.83)', 'numConnections': 1, 'avereAddress': '10.1.26.135'}]
```

stats.clientCounts

NAME
stats.clientCounts

SYNOPSIS
stats.clientCounts(node, vserver, [summary]) => array_of_structs

DESCRIPTION
Returns the current number of client connections by connection types.

PARAMETERS
- node: (string) The name of the node in which the clients are connected
  - 'cluster' or '' can be used to specify the entire cluster
- vserver (string) The name of the vserver in which the clients are connected
  - '' can be used to specify all vservers
- [summary] (bool) Aggregates all connected client statistics into a single result
  - Defaults to 'True'

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the connections:

  - [node]:
    (string) The name of the node
  - [vserver]:
    (string) The name of the vserver
  - connectedcifs:
    (integer) The number of connected CIFs clients
  - connectednfs:
    (integer) The number of connected NFS clients
  - connectednfstcp:
    (integer) The number of NFS clients connected using TCP
  - connectednfsudp:
    (integer) The number of NFS clients connected using UDP
  - connected:
    (integer) The total number of connected clients

EXAMPLE
print clientHandle.stats.clientCounts('','nfs', False)
{'node': 'node0', 'connectednfsudp': 1, 'connectednfstcp': 4, 'vserver': 'nfs', 'connected': 6, 'connectedcifs': 1, 'connectednfs': 5}
{'node': 'node1', 'connectednfsudp': 1, 'connectednfstcp': 1, 'vserver': 'nfs', 'connected': 2, 'connectedcifs': 0, 'connectednfs': 2}

print clientHandle.stats.clientCounts('','nfs')
{connectednfsudp': 2, 'connectednfstcp': 5, 'vserver': 'nfs', 'connected': 8, 'connectedcifs': 1, 'connectednfs': 7}

stats.disableHotCollection

NAME
stats.disableHotCollection

SYNOPSIS

stats.disableHotCollection(vserverName) => status

DESCRIPTION
Disables the collection of hot statistics for clients on the specified vserver.

Use the stats.list method to obtain a list of collections.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
print clientHandle.stats.disableHotCollection('newserver')
success

NAME

stats.enableHotCollection

SYNOPSIS

stats.enableHotCollection(vserverName, [hotClientPeriod], [hotClientLimit]) => status

DESCRIPTION

Enables the collection of hot statistics for clients on the specified vserver.

Use the stats.list method to obtain a list of collections.

PARAMETERS

- vserverName: (string) The name of the vserver
- [hotClientPeriod]: (integer) Optional. The polling period for collecting hot client statistics on this vserver. The default collection period is 60 seconds.
- [hotClientLimit]: (integer) Optional. The maximum number of hot clients tracked on the vserver. The default value is 10 clients.

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.stats.enableHotCollection('newserver', 10, 20)
success
```

NAME
stats.get

SYNOPSIS
stats.get([scope], collection) => stat_value_struct

DESCRIPTION
Returns the current values of statistics in a collection.
Use the stats.list method to obtain a list of collections.

NOTE: You can also use the gperf collection to obtain statistics

PARAMETERS
- [scope]: (string) Optional. One of the following collection scopes:
  - 'local' to return statistics about the local node
  - 'cluster' (the default) to return cluster-wide statistics
  - '<node_name>' to return statistics about the specified node
- collection: (string) A collection name. Use the stats.list method to obtain a list of collections.

RETURNS
- stat_value_struct: An XML-RPC struct that contains the following name:value pairs:
  - <stat_name>: (string) The name of the statistic
  - <stat_value>:
    (integer) The value of the statistic

EXAMPLE
```
print clientHandle.stats.get('acl')
{'ruleEvalSquashRoot': 0.0, 'ruleEvalSquashAll': 0.0, 'ruleEvalAcces
 sNO': 0.0, 'caseInsensitiveGetFaclNfsv4Noent': 0.0, 'disable': 0.0,
 'ruleEvalSnapshotRO': 0.0, 'caseInsensitiveSetFaclNfsv4Noent': 0.0,
 'caseInsensitiveGetFacl': 0.0, 'ruleEvalDenyRoot': 0.0, 'caseInsens
 itiveGetFaclNfsv4': 0.0, 'caseInsensitiveSetFacl': 0.0, 'caseInsens
 itiveSetFaclNfsv4': 0.0, 'aclNoEnt': 0.0, 'ruleEvalAccessRO': 0.0,
 'ruleEvalErr': 0.0, 'aclNotSupp': 0.0}
```

stats.getHistoryConfig

NAME
stats.getHistoryConfig

SYNOPSIS
stats.getHistoryConfig(collection) => struct

DESCRIPTION
Returns information about the history configuration for a collection.
Use the stats.list method to obtain a list of collections.

PARAMETERS
- collection: (string) A collection name. Use the stats.list method to obtain a list of collections.

RETURNS
- stat_value_struct: An XML-RPC struct that contains the following name:value pairs:
  - rev: (deprecated) The revision number of the configuration
  - state: (string) Whether the configuration is 'active' or 'stopped'
  - collection: (string) The name of the statistics collection
  - step: (integer) How often statistics values will be gathered, in seconds
  - length: (integer) Total length of the log, in seconds; that is, how far back it will cover.
  - id: (deprecated) The UUID of the configuration

EXAMPLE
```
print clientHandle.stats.getHistoryConfig('acl')
{'rev': 'b20c2672-51ed-446b-a139-414b95eccbe2', 'state': 'active',
 'collection': 'acl', 'step': '30', 'length': '1800', 'id': 'bc79e
 541-03ca-4d93-9b15-8779a0852bdc'}
```

NAME

stats.history

SYNOPSIS

stats.history([scope], collection, stats, startingTime, endingTime) => stat_value_struct

DESCRIPTION

Returns historical values of the specified statistics.
Use the stats.list method to obtain a list of collections.

PARAMETERS

- [scope]: (string) Optional. One of the following collection scopes:
  - 'local' to return statistics about the local node
  - 'cluster' (the default) to return cluster-wide statistics
  - '<node_name>' to return statistics about the specified node
- collection: (string) The name of the statistics collection
- stats: (string) A comma-delimited list of statistics names, WITHOUT any spaces
- startingTime: (integer) The start time of the graph, given as one of the following:
  - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
  - A negative number defining the number of seconds before 'now'
- endingTime: (integer) The ending time for the returning the historical statistics, given as one of the following:
  - Epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
  - A negative number defining the number of seconds before 'now'
  - Zero (0), for 'now'

RETURNS

- stat_value_struct: An XML-RPC struct that contains the following set of name:value pairs for the collection:
  - <statName>:
    (string) The name of the statistic
  - <startingTime>:
    An XML-RPC struct that contains the following set of name:value pairs for each statistic:
    - <collection-time>:
      The time the statistic was collected, in epoch seconds
    - <statistic>:
      The value of the statistic at the time it was collected

EXAMPLE

```
print clientHandle.stats.history('local','summary',
    'nfs_front_call,nfs_back_call',0,-600)
{'nfs_back_call':{'1303520000': 3925.0, '1303500000': 28421.0,
 '1303510000':25391.0}, 'nfs_front_call':{'1303520000': 0.0,
 '1303500000': 0.0, '1303510000':0.0}}
```

NAME
stats.hotClients

SYNOPSIS
stats.hotClients(vserverName) => array_of_structs

DESCRIPTION
Returns hot clients for a vserver.

The stats.enableHotCollection method must be invoked before hot-client information can be returned.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the hot clients:
  - node: (string) The name of the node to which the hot client is connected.
  - clientAddress: (string) The name, IP address, and port number of the hot client, in the form of "name.company.com (ip-address:port)"
  - clientType: (string) The type of the client. Current values are 'cifs', 'nfs', or 'unknown'.
  - serverAddress: (string) The cluster address or vserver address to which the hot client is connected.
  - ops: (float) Number of operations from the hot client.

EXAMPLE
```
print clientHandle.stats.hotClients('gns')
[{'node': 'boiL33',
'clientAddress': 'tenth.company.com (10.1.2.83:888)',
'serverAddress': '10.1.26.126',
'ops': 2861006.0},
{'node': 'wanboi13', 'clientAddress': 'first.cc.company.com (10.1
.4.202:886)', 'serverAddress': '10.1.26.126', 'ops': 51012442.0}]
```

NAME
stats.hotFiles

SYNOPSIS
stats.hotFiles(fileType, [consolidated]) => array_of_structs

DESCRIPTION
Lists hot files of a specified type.

This method can be called whether or not the stats.enableHotCollection method has been invoked.

PARAMETERS
- fileType: (string) The type of action on the file, a comma-separated list of any of the following:
  - 'OPS'
  - 'READ'
  - 'WRITE'
  - 'UPDATES'

- [consolidated]: (boolean) Optional. Whether duplicate entries are merged into single entries (True) or not (False - the default)

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs:

  - node: (string) The name of the primary node that is serving the file
  - name: (string) The path to the file, its FSID and FILEID, and whether the ids are still being resolved to the name
  - <type> (string) The number of actions of <type> on the file, determined by the 'fileType' input parameter.
  - rank: (float) The rank of the file in the list of hot files, in ascending order of amount of activity
  - fkey: (string) A concatenation of the path to the file, the FSID, and the FILEID, in the form "name:fsid:fileid"
  - type: (string) Whether the file is a "regular" file ('REG') or a directory ('DIR')
  - fsid: (float) The file system ID
  - fileid: (float) The file's file ID

EXAMPLE
```
print clientHandle.stats.hotFiles('READ,OPS')
[{'node': 'maxout10',
  'name': 'filer.company.com:Resolving (FSID:1500069006 FILEID:64)',
  'ops': 0.0,
  'rank': 1.0,
  'fkey': 'filer.company.com:1500069006:64',
  'type': 'DIR',
  'fsid': 1500069006.0,
  'fileid': 64.0},
 {'node': 'maxout11', 'name': 'filer2.company.com:Resolving (FSID:15000
 69006 FILEID:64)', 'ops': 0.0, 'rank': 1.0, 'fkey': 'filer2.company.co
 m:1500069006:64', 'type': 'REG', 'fsid': 150069006.0, 'fileid': 64.0}]
```

stats.isHotCollectionEnabled

NAME
stats.isHotCollectionEnabled

SYNOPSIS
stats.isHotCollectionEnabled(vserverName) => collectionEnabled

DESCRIPTION
Checks whether collection of hot statistics is enabled for clients on the specified vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- collectionEnabled: (boolean) Whether statistics collection is enabled (True)
or not (False)

EXAMPLE
print clientHandle.stats.isHotCollectionEnabled('vserver1')
True

NAME

stats.list

SYNOPSIS

stats.list([collection]) => collectionNameArray | statNameArray

DESCRIPTION

Returns either the names of available collections, or the statistics for a single specified collection. Use the stats.list method to obtain a list of collections.

NOTE: You can also use the gperf collection to obtain statistics.

PARAMETERS

- [collection]: (string) Optional. The collection for which you want statistics

RETURNS

- collectionNameArray:
  (array) If 'collection' is not specified, an array of all the collection names for the cluster. The names are given as strings.
- statNameArray:
  (array) If 'collection' is specified, an array of statistics names in that collection. The names are given as strings.

EXAMPLE

```
print clientHandle.stats.list()
['acl', 'ctc', 'nsm', 'nlm', 'mount', 'nfs', 'zcSocket_RecvErrors',
 'zcSocket_SendErrors', 'xdr', 'cfsMalloc', 'extGroup', 'nfsdb', 'c
 mdctlHiPri', 'cmdctl', 'dirmgr_intentlog', 'dirmgr_inflightlog', '
 dirmgr_flushdirlog', 'all_token_mgrs', 'rollingtrace', 'dispatche
 r',...'nlm_back_proc_vserver1', 'nlm_back_rpc_vserver1', 'nlm_back
 _vdisk_vserver1', 'vcm_flush_vserver1', 'vcm_server_vserver1']

print clientHandle.stats.list('acl')
['aclNoEnt', 'ruleEvalErr', 'ruleEvalAccessRO', 'ruleEvalAccessNO',
 'ruleEvalSnapshotRO', 'ruleEvalSquashRoot', 'ruleEvalDenyRoot', 'r
 uleEvalSquashAll', 'disable', 'caseInsensitiveSetFacl', 'caseInsen
 sitiveSetFaclNfsv4', 'caseInsensitiveSetFaclNfsv4Noent', 'caseInse
 nsitiveGetFacl', 'caseInsensitiveGetFaclNfsv4', 'caseInsensitiveGe
 tFacNfsv4Noent', 'aclNotSupp']
```

stats.listConnections

NAME
stats.listConnections

SYNOPSIS
stats.listConnections(name, type, [resolveClientIP]) => array_of_structs

DESCRIPTION
Returns connection statistics for a specified node, vserver, or core filer.

PARAMETERS
- name: (string) The name of the node, vserver, or core filer
- type: (string) One of the following:
  - 'node' to return connection statistics for both vservers and core filers
  - 'node-vserver' to return connection statistics between the vserver and any connected clients
  - 'node-corefiler' to return connection statistics between the node and any connected core filers
  - 'corefiler' to return connection statistics between the cluster and any connected core filers
  - 'vserver' to return connection statistics for the specified vserver

- [resolveClientIP]: (boolean) Optional. Specifies whether the method makes a best-effort attempt to resolve client hostnames from the client IP addresses (True) or not (False - the default)

  NOTE: Client hostname resolution can reduce the performance of the method, especially if a large number of clients are connected.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the connections:

  - node: (string) The name of the node on which the connection is located
  - protocol: (string) The network protocol used for the connection, generally 'tcp tcp' or 'udp'
  - avereAddress: (string) The IP address and port of the cluster connection
  - clientAddress: (string) The IP address and port of the client connection
  - vserver_name | corefiler: (string) Whether the connection is a vserver connection or a core filer connection, with the name of the vserver or core filer.
  - type: (string) The type of connection, one of the following:
    - 'vserver' (the default)
    - 'node'
    - 'corefiler'
    - 'node-corefiler'
    - 'node-vserver'

EXAMPLE
```
print clientHandle.stats.listConnections('wantwo', 'node-corefiler')
[{'node': 'wantwo',
  'protocol': 'tcp tcp',
  'avereAddress': '10.1.26.17:676',
  'clientAddress': '10.1.18.41:2049',
  'corefiler': 'mainFiler',
  'type': 'nfs_backend'},
```

{'node': 'wantwo', 'protocol': 'tcp tcp', 'avereAddress': '10.1.00.
00:655', 'clientAddress': '10.1.00.00:2049', 'mass': 'mainFiler',
'corefiler': 'mainFiler', 'type':'nfs_backend'}]

NAME
stats.listHistoryConfigs

SYNOPSIS
stats.listHistoryConfigs() => array_of_structs

DESCRIPTION
Displays the statistics-history configuration.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the connections:

  - id: (deprecated) The UUID of the configuration
  - rev: (deprecated) The revision number of the configuration
  - collection: (string) The name of the statistics collection
  - length: (integer) Total length of the log, in seconds; that is, how far back it will cover.
  - step: (integer) How often statistics values will be gathered, in seconds
  - state: (string) Whether the configuration is 'active' or 'stopped'

NAME

support.executeAdvancedMode

SYNOPSIS

support.executeAdvancedMode(scope, mode, duration, [filter], [comment]) => status

DESCRIPTION

Starts an advanced information gathering mode.

PARAMETERS

- scope: (string) One of the following:
  - 'cluster' (the default) to return cluster-wide information
  - 'node' to return information about the local node
  - '<node_name>' to return information about the specified node
- mode: (string) Obtains valid modes from the struct provided by the support.listAdvancedModes method
- duration: (integer) How long, in seconds, the gathering process should execute before exiting
  - A duration of '0' will cause gathering to occur until the support.stopAdvancedMode method is called.
  - Only one advanced gather can be active at a time.

  WARNING: Only use large (> 3600) values for duration with the guidance of Avere Global Services.
- [filter]: (string) A valid tcpdump filter specification. Further information can be found in the tcpdump man page
- [comment]: (string) Additional information about the gathering mode, such as the problem that you were seeing when the gather mode was executed.

RETURNS

- status: (string) One of the following:
  - A handle that can be passed to the support.taskIsDone or support.taskComplete methods
  - A reason for failure

EXAMPLE

```
print clientHandle.support.executeAdvancedMode('cluster', 'perfmin', 60)
a7536f6e-a0c9-11e3-b818-000c29159544
```

support.executeNormalMode

NAME

support.executeNormalMode

SYNOPSIS

support.executeNormalMode(scope, mode, [comment], [advancedParameters]) => status

DESCRIPTION

Starts a normal information gathering mode.

PARAMETERS

- scope: (string) One of the following:
  - 'cluster' (the default) to return cluster-wide information
  - 'node' to return information about the local node
  - '<node_name>' to return information about the specified node
- mode: Obtain valid modes from the struct provided by the support.listNormalModes method.

- [comment]: (string) Additional information about the gathering mode, such as the problem that you were seeing when the gather mode was executed.

- [advancedParameters]:
  An XML-RPC struct that contains advanced settings for trace file upload.
  The system will upload trace files with modification times from eventtime-beforemin to eventtime+aftermin. The parameters include the following:
  - eventtime: (integer) The time of the event for which trace files will be uploaded, given in epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970.
  - beforemin: (string) The number of minutes before the event time specified by 'eventtime' to start uploading files. Defaults to 10 minutes.
  - aftermin: (string) The number of minutes after the event time specified by 'eventtime' to stop uploading files. Defaults to 2 minutes.

RETURNS

- status: (string) One of the following:
  - A handle that can be passed to the support.taskIsDone or support.taskComplete methods
  - A reason for failure

EXAMPLE

```
print clientHandle.support.executeNormalMode('cluster', 'gsicurstats')
32d75617-a0c5-11e3-910f-000c29159544
```

NAME

support.get

SYNOPSIS

support.get() => supportInfoStruct

DESCRIPTION

PARAMETERS

- No input parameters are required for this method.

RETURNS

-supportInfoStruct:

- crashInfo: (string) Whether error report summaries (backtraces) are uploaded ('yes') or not ('no" - the default)

- corePolicy: (string) The method for handling error reports (core dumps) when space is limited on the core partition, one of the following:
  - 'overwriteOldest' (default): The oldest error report is removed
  - 'overwriteNewest': The most recent error report is removed
  - 'saveUntilFilled': Error reports are saved until the core partition runs out of space, at which time new error reports will not be saved. Use this option only under the direction of Avere Global Services.

- tested: (string) Whether the upload path to Avere Global Services has been validated, either 'yes' or 'no'

  A value of 'no' is returned when the support.testUpload method was not able to successfully upload a test file, and generally means that there is some issue with the upload settings (for example, the proxy settings) that is inhibiting the upload from completing successfully.

- statsMonitor: (string) Whether to periodically gather cluster statistics and upload daily to Avere Global Services, either 'yes' or 'no'

- rollingTrace: (string) Whether to gather detailed tracing information for cluster processes, either 'yes' or 'no'

- traceLevel: (string) A number (in the form of a hexidecimal string) that specifies what information about AvereOS is included in a rollingTrace, for possible later upload to Avere Global Services

- memoryDebugging: (string) Whether detailed memory information is included in error reporting ('no' - the default, or 'yes'). This can slow performance while it is enabled.

- generalInfo: (string) Whether to upload a snapshot of the cluster configuration and logs to Avere Global Services daily, either 'yes' or 'no'

- customerId: (string) The customer id used when interacting with the Avere upload process. Currently this is the cluster name by default.
  NOTE: If the name is changed using the support.modify method, you need to make sure it is unique across all clusters, including those of other customers. Otherwise, information might be inaccurate. Please consult with Avere Global Systems for a customer id.

- SPSLinkEnabled: (string) Whether to periodically upload status information over a secure link to Avere Global Services, either 'yes' or 'no'

- SPSLinkInterval: (integer) The interval, in seconds, that the cluster will upload data over the SPSLink, between 30 and 86400 (1 day). The default is 300.

- remoteCommandEnabled: (string) Specifies whether Avere Global Services is allowed to send remote commands to the cluster over the SPSLink, either 'no' (the default) or 'yes'

EXAMPLE

```
print clientHandle.support.get()
{'crashInfo': 'yes', 'corePolicy': 'overwriteOldest', 'tested': 'no',
'statsMonitor': 'no', 'rollingTrace': 'no', 'traceLevel': '0x1',
'memoryDebugging': 'no', 'generalInfo': 'no', 'customerId': 'ligo'}
```

NAME

support.getAddress

SYNOPSIS

support.getAddress() => addressStruct

DESCRIPTION

Returns a list of shipping address attributes.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- addressStruct: An XML-RPC struct that contains detailed settings for the shipping address module.
  - address1: (string) The first part of the street address
  - address2: (string) The second part of the street address
  - city: The name of the city
  - state: The name of the state/province
  - zip: The appropriate ZIP or postal code
  - country: The name of the country
  - contact: The name of the contact person
  - telephone: The telephone number for the contact

EXAMPLE

```
print clientHandle.support.getAddress()
{'city': 'Pittsburgh', 'zip': '15333', 'address1': '125 Everywhere Street',
'address2': '', 'telephone': '412-555-1212', 'state': 'FL', 'contact':
'Company', 'country': 'USA'}
```

support.listAdvancedModes

NAME

support.listAdvancedModes

SYNOPSIS

support.listAdvancedModes() => modeArray

DESCRIPTION

Lists valid advanced gathering modes, with the default text used in the GUI.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- modeArray: (array) An array with the following two attributes:

  -validModesStruct: An XML-RPC struct that contains the following name:value pairs:
  - perfin: (string) 'Performance information'
  - perfrwtrace: (string) 'Read/Write Trace'
  - perfplumber': (string) 'Memory Debugging'
  - perfvcmtrace': (string) 'Trace'
  - perfflushtrace':
    (string) 'Writeback Trace'
  - perfthreadedstats':
    (string) 'Stats gathering'
  - perffullpacketring':
    (string) 'Full packet capture (50GB Ring buffer)'
  - perfdirtrace': (string) 'Directory Trace'
  - perfvcmpartialpacket':
    (string) 'Trace/partial packet'
  - perffullpacket':
    (string) 'Full packet capture'
  - perftimingtrace':
    (string) 'Timing Trace'

  - key: (string) The default mode, which is passed to the
    support.executeAdvancedMode() method

EXAMPLE

```
print clientHandle.support.listAdvancedModes()
[{'perfmin': 'Performance information', 'perfrwtrace': 'Read/Write
Trace', 'perfplumber': 'Memory Debugging', 'perfvcmtrace': 'Trace',
'perfflushtrace': 'Writeback Trace', 'perfthreadedstats': 'Stats ga
thering', 'perffullpacketring': 'Full packet capture (50GB Ring buf
fer)', 'perfdirtrace': 'Directory Trace', 'perfvcmpartialpacket': '
Trace/partial packet', 'perffullpacket': 'Full packet capture', 'pe
 rftimingtrace': 'Timing Trace'}, 'perffullpacket']
```

NAME
support.listCores

SYNOPSIS
support.listCores(scope) => coreInfoArray

DESCRIPTION
Lists available error reports (core dumps) for the specified nodes.

PARAMETERS
- scope: (string) One of the following:
  - 'cluster' to return cluster-wide error report information
  - 'node' to return error report information about the local node
  - '<node_name>' to return error report information about the specified node

RETURNS
- coreInfoStruct: (struct) An XML-RPC struct that contains the following name:value pairs:

  - <nodeName>: (string) The name of each node

  - <coreInfoArray>:
    An array of structs, each of which contains the following elements:

    - disabled: (boolean) Whether the error report is in the process of being dumped or uploaded (True) or not (False)
    - size: (integer) The actual size of the error report
    - displaysize: (string) A human-readable representation of the file size, displayed as MB, GB, and so forth
    - filename: (string) The name of the error report. This name can be passed to the support.uploadCores or support.removeCores methods.

EXAMPLE
```
print clientHandle.support.listCores('cluster')
{'ligo': [{'disabled': False, 'size': '1779494912', 'displaysize':
'1 GB', 'filename': 'company.66798.1375387294'}, {'disabled': False
, 'size': '16384', 'displaysize': '16 KB', 'filename': 'company.667
98.1375387294.bt'}]}
```

NAME

support.listCustomSettings

SYNOPSIS

support.listCustomSettings() => settingsArray

DESCRIPTION

Lists all custom settings that have been applied to the cluster.

RETURNS

- settingsArray: (array) An array of XML-RPC structs that contain some or all of the following name:value pairs:

  - name: (string) The name of the setting
  - checkCode: (string) A two-letter string provided by Avere
  - value: (string) The value for the setting
  - note: (string) A note describing why the setting was applied. This is blank if it was not originally provided.
  - default: (string) The default value for this setting. This is blank if it cannot be determined.
  - modtime: (string) The time at which the custom setting was applied, in the format 'yyyy-mm-dd hh:mm'

EXAMPLE

The following Python example will list all of the current custom settings applied to the cluster.

```
print support.listCustomSettings()
[{'modtime': '2013-10-10 10:55', 'name': 'gsiInfo.test_setting', 'checkC
ode': 'MU', 'default': '','value': 'true', 'note': 'testing a setting'}]
```

NAME

support.listNormalModes

SYNOPSIS

support.listNormalModes() => modeArray

DESCRIPTION

Returns the valid gathering modes for a cluster or node.

RETURNS

- modeArray: (array) An array with the following two attributes:

-validModesStruct: An XML-RPC struct that contains the following name:value pairs:

- gsidirmgr: (string) 'Local directory information'
- gsimid: (string) 'Data dump information'
- gsicleanup: (string) 'Remove failed statistics uploads (Use under direction of Avere Global Services)'
- gsilog: (string) 'Minimal log collection'
- gsicurstats: (string) 'Current statistic information'
- gsicatchup: (string) 'Retry failed statistics uploads (Use under direction of Avere Global Services)'
- gsitrace: (string) 'Current trace information'
- gsirollingtrace: (string) 'Rolling trace information'
- gsirollingtracemgmtd: (string) 'Rolling management trace information'
- gsihiststats: (string) 'Historical statistic information'
- gsimin: (string) 'Normal support information'

- key: (string) The default mode, which is passed to the support.executeNormaldMode() method

EXAMPLE

```
print clientHandle.support.listNormalModes()
[{'gsidirmgr': 'Local directory information', 'gsimid': 'Data dump
information', 'gsicleanup': 'Remove failed statistics uploads (Use
under direction of Avere Global Services)', 'gsilog': 'Minimal log
collection', 'gsicurstats': 'Current statistic information', 'gsic
atchup': 'Retry failed statistics uploads (Use under direction of
Avere Global Services)', 'gsitrace': 'Current trace information',
'gsirollingtrace': 'Rolling trace information', 'gsirollingtracemg
mtd': 'Rolling management trace information', 'gsihiststats': 'His
torical statistic information', 'gsimin': 'Normal support informat
ion'}, 'gsimin']
```

NAME

support.modify

SYNOPSIS

support.modify(attrsStruct) => attrsArray

DESCRIPTION

Sets one or more support attributes. Use the support.get method to obtain the current attributes.

PARAMETERS

- attrsStruct: (struct) An XML-RPC struct that contains the following name:value pairs:

  - crashInfo: (string) Whether error report summaries (backtraces) are uploaded ('yes') or not ('no" - the default)

  - corePolicy: (string) The method for handling error reports (core dumps) when space is limited on the core partition, one of the following:
    - 'overwriteOldest' (default): The oldest error report is removed
    - 'overwriteNewest': The most recent error report is removed
    - 'saveUntilFilled': Error reports are saved until the core partition runs out of space, at which time new error reports will not be saved. Use this option only under the direction of Avere Global Services.

  - tested: (string) Whether the upload path to Avere Global Services has been validated, either 'yes' or 'no'

    A value of 'no' is returned when the support.testUpload method was not able to successfully upload a test file, and generally means that there is some issue with the upload settings (for example, the proxy settings) that is inhibiting the upload from completing successfully.

  - statsMonitor: (string) Whether to periodically gather cluster statistics and upload daily to Avere Global Services, either 'yes' or 'no'

  - rollingTrace: (string) Whether to gather detailed tracing information for cluster processes, either 'yes' or 'no'

  - traceLevel: (string) A number (in the form of a hexidecimal string) that specifies what information about AvereOS is included in a rollingTrace, for possible later upload to Avere Global Services

  - memoryDebugging: (string) Whether detailed memory information is included in error reporting ('no' - the default, or 'yes'). This can slow performance while it is enabled.

  - generalInfo: (string) Whether to upload a snapshot of the cluster configuration and logs to Avere Global Services daily, either 'yes' or 'no'

  - customerId: (string) The customer id used when interacting with the Avere upload process. Currently this is the cluster name by default.
    NOTE: If the name is changed using the support.modify method, you need to make sure it is unique across all clusters, including those of other customers. Otherwise, information might be inaccurate. Please consult with Avere Global Systems for a customer id.

- SPSLinkEnabled: (string) Whether to periodically upload status information to Avere Global Services, either 'yes' or 'no'

- SPSLinkInterval: (integer) The interval, in seconds, that the cluster will upload data over the SPSLink, between 30 and 86400 (1 day). The default is 300.

- remoteCommandEnabled:
(string) Specifies whether Avere Global Services is allowed to send remote commands to the cluster over the SPSLink, either 'no' (the default) or 'yes'

RETURNS
- newAttrsArray: (array) An array of the following two parts:
  - status: (string) success or a reason for failure
  - newAttrsStruct: A struct that includes all of the name:value pairs of the attrsStruct, including values that have changed

EXAMPLE
```
print clientHandle.support.modify({'crashInfo': 'yes', 'statsMonitor':
  'yes', 'generalInfo': 'no', 'rollingTrace': 'yes'})
['success', {'remoteCommandEnabled': 'no', 'crashInfo': 'yes', 'tested
': 'no', 'statsMonitor': 'yes', 'rollingTrace': 'yes', 'SPSLinkEnabled
': 'no', 'corePolicy': 'overwriteOldest', 'memoryDebugging': 'no', 'ge
neralInfo': 'no', 'traceLevel': '0x1', 'SPSLinkInterval': '300', 'cust
omerId': 'ligo'}]
```

support.modifyAddress

NAME

support.modifyAddress

SYNOPSIS

support.modifyAddress(addressStruct) => addressArray

DESCRIPTION

Sets one or more mailing address attributes. The current values of these attributes can be obtained by using the support.getAddress method.

PARAMETERS

- addressStruct: An XML-RPC struct that contains one or more of the following name:value pairs:
  - address1: (string) The first part of the street address
  - address2: (string) The second part of the street address
  - city: (string) The name of the city
  - state: (string) The name of the state/province
  - zip: (string) The appropriate ZIP or postal code
  - country: (string) The name of the country
  - contact: (string) The name of the contact person
  - telephone: (string) The telephone number for the contact

RETURNS

- addressArray: (array) An array of the following two parts:
  - status: (string) success or a reason for failure
  - newAddressStruct: A struct that includes all of the name:value pairs of the addressStruct, including values that have changed

EXAMPLE

```
print clientHandle.support.modifyAddress({'contact':'J.User', 'state':'PA'})
['success', {'city': 'Pittsburgh', 'zip': '15333', 'address1': '125
Everywhere Street', 'address2': '', 'telephone': '412-555-1212',
'state': 'PA', 'contact': 'J. User', 'country': 'USA'}]
```

support.removeCores

NAME

support.removeCores

SYNOPSIS

support.removeCores(scope, filename) => removedReportsStruct

DESCRIPTION

Removes specified error reports (core dumps) from the specified nodes.

PARAMETERS

- scope: (string) One of the following report removal scopes:
  - 'local' to remove error reports from the local node
  - 'cluster' (the default) to remove error reports from the cluster
  - '<node_name>' to remove error reports from the specified node
- filename: (string) The name of the error report

RETURNS

- removedReportsStruct:
  An XML-RPC struct that contains the following name:value pairs:

  - <filename>: (string) The name of the error report (core dump) that was to be removed.

  - nodeStatus: A struct that includes the following name:value pairs:
    - nodeName: (string) The name of each node in the 'scope'
    - statusStruct: A struct that includes the following:
      - status: (string) Either 'Success' or the reason for failure to remove the core

EXAMPLE

```
print clientHandle.support.removeCores('cluster', 'smbd.61061.map')
{'smbd.61061.map': {'node3': {'status': 'Success'},'node2':
{'status': 'Failed - /support/cores/smbd.61061.map does not
exist'}, 'node1': {'status': 'Failed - /support/cores/smbd.
61061.map does not exist'}}}
```

NAME

support.removeCustomSetting

SYNOPSIS

support.removeCustomSetting(name) => status

DESCRIPTION

Removes a custom setting from the cluster.

PARAMETERS

- name: (string) The name of the setting that is to be removed

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

The following Python example will remove a custom setting which is currently applied to the cluster.

```
print support.removeCustomSetting("gsiInfo.test_setting")
success
```

NAME

support.setCustomSetting

SYNOPSIS

support.setCustomSetting(name, checkCode, value, [note]) => status

DESCRIPTION

Applies a custom setting to the cluster.

PARAMETERS

- name: (string) The name of the setting to be applied
- checkCode: (string) A two-letter string provided by Avere
- value: (string) The value for the setting being applied
- [note]: (string) Optional. A note describing why the setting is being applied

RETURNS

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

The following Python examples will apply a custom setting which will not affect the operation of the cluster.

```
print clientHandle.support.setCustomSetting('gsiInfo.test_setting',
  'MU', 'true')
success

print clientHandle.support.setCustomSetting('gsiInfo.test_setting',
  'MU', 'true', 'testing')
success
```

support.stopAdvancedMode

NAME

support.stopAdvancedMode

SYNOPSIS

support.stopAdvancedMode(scope) => status

DESCRIPTION

Stops advanced information gathering mode on the specified nodes.

PARAMETERS

- scope: (string) One of the following:
  - 'cluster' (the default) to halt information gathering on the cluster
  - 'node' to halt information gathering on the local node
  - '<node_name>' to halt information gathering on the specified node

RETURNS

- status: (struct) An XML-RPC struct that contains the following name:value pairs, containing status results for each node specified by the 'scope' parameter.
  - <node_name>: (string) The name of the node, with whether the mode was stopped or not. If you specified 'cluster', there will probably be more than one node.

EXAMPLE

```
support.stopAdvancedMode('my_node')
{'my_node': 'success'}
```

NAME
support.supportMode

SYNOPSIS
support.supportMode() => status

DESCRIPTION
Support options used by Avere support.

PARAMETERS
- key: (string) support provided key

RETURNS
- status: (string) Informational status message

support.taskComplete

NAME

support.taskComplete

SYNOPSIS

support.taskComplete(handle) => exitStatusStruct

DESCRIPTION

Retrieves the return code from a background job (task).

Task completion status can only be retrieved from the node where the background job was originally submitted.

NOTE: IP addresses, particularly the cluster management IP, can move amongst nodes in the cluster.

PARAMETERS

- handle: (string) A handle that can be retrieved from one of the following methods:
  - support.executeAdvancedMode
  - support.executeNormalMode
  - support.updatePackage
  - support.uploadCores

RETURNS

- exitStatusStruct: (struct) An XML-RPC struct that contains the following name:value pairs,
  - <node_or_file>: (string) The name of the node or a file passed to the original method
  - status: (struct) A struct with the name of the node, and either 'Success or 'Failed', with a failure status sometimes returning additional information. Occasionally there is an encoded string in a failed result that might need to be sent to Avere Global Services for analysis.

EXAMPLE

```
print clientHandle.support.taskComplete('2d7617-a0c5-11e3-910f-000c59544')
{'node1': {'status': 'Success'}, 'node7065': {'status': 'Success'}}
```

NAME

support.taskIsDone

SYNOPSIS

support.taskIsDone(handle) => completionStatus

DESCRIPTION

Checks the status of the specified background job.

Task completion status can only be retrieved from the node where the background job was originally submitted.

NOTE: IP addresses, particularly the cluster management IP, can move amongst nodes in the cluster.

PARAMETERS

- handle: (string) A handle that can be retrieved from one of the following methods:
  - support.executeAdvancedMode
  - support.executeNormalMode
  - support.updatePackage
  - support.uploadCores

RETURNS

- completionStatus: (string) One of the following:
  - 'True' if the background task is complete
  - 'False' if the background task is still running
  - 'Unknown' if the 'handle' parameter indicates a background job that doesn't exist at this time

EXAMPLE

```
print clientHandle.support.taskIsDone('2d7617-a0c5-1e3-910f-000c59544')
Unknown
```

NAME

support.testUpload

SYNOPSIS

support.testUpload() => status

DESCRIPTION

Uploads a test file to Avere Global Services and validates that the test file was uploaded properly.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- testUploaded: (boolean) Either True, if the uploaded test succeeded, or False if it did not.

  If the upload succeeds, the 'tested' attribute returned by the support.get method will be 'yes', otherwise it will be 'no'.

  A value of False (or 'no') generally means that there is some issue with the upload settings (for example, the proxy settings) that is inhibiting the upload from completing successfully.

EXAMPLE

```
print clientHandle.support.testUpload()
True
```

NAME

support.updatePackage

SYNOPSIS

support.updatePackage(pkgurl) => status

DESCRIPTION

Upgrades the support package by:
- Downloading the upgrade to one of the nodes in the cluster,
- distributing the updated package to all nodes, and then
- installing the package on all nodes in the cluster.

Updating the support package should only be done (infrequently) under the guidance of Avere Global Services. Upgrading the support package will not interrupt cluster services.

PARAMETERS

- pkgurl: (string) The URL for the upgraded software

RETURNS

- status: (string) One of the following:
  - A handle that can be passed to the support.taskIsDone or support.taskComplete methods
  - A reason for failure

EXAMPLE

```
print clientHandle.support.updatePackage
  ('https://download.averesystems.com/software/support.pkg')
ec9ae2ab-a5a-11e3-9463-000c291544
```

NAME

support.uploadCores

SYNOPSIS

support.uploadCores(scope, {reportName | reportNameArray}, [comment]) => uploaded_struct

DESCRIPTION

Uploads specified error reports (core dumps) from the specified nodes to Avere Global Services.

PARAMETERS

- scope: (string) One of the following report removal scopes:
  - 'local' to remove error reports from the local node
  - 'cluster' (the default) to remove error reports from the cluster
  - '<node_name>' to remove error reports from the specified node

- One of the following:
  - reportName: (string) The name of the error report. You can obtain these names using the support.listCores method.
  - reportNameArray: (array) An array of error report names. The names are strings.

- [comment]: (string) Additional information about the corefile, such as the workload that was being executed when the error report was generated.

RETURNS

- uploaded_struct:
  (struct) A struct with the name of the report, and a handle that can be passed to the support.taskIsDone or support.taskComplete methods.

EXAMPLE

```
print clientHandle.support.uploadCores('node1', 'disabled')
{'disabled': '46070ceb-a4b7-11e3-9b58-000c29159544'}
```

NAME
system.disableAPI

SYNOPSIS
system.disableAPI(api) => status

DESCRIPTION
Disables an advanced level of XML-RPC methods (APIs). This can be used to disable the following methods:

- corefiler.remove
- corefiler.invalidate
- maint.unsetDataRepository
- maint.suspendAccess
- maint.invalidateCache
- maint.reformatCluster
- maint.wipeCluster
- node.remove
- vserver.remove

PARAMETERS
- api: (string) One of 'maintenance' or 'preview'

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.system.disableAPI('maintenance')
success
```

NAME

system.enableAPI

SYNOPSIS

system.enableAPI(api) => status

DESCRIPTION

Enables a previously disabled advanced level of XML-RPC methods (APIs).
This can be used to enable the following methods:

- corefiler.remove
- corefiler.invalidate
- maint.unsetDataRepository
- maint.suspendAccess
- maint.invalidateCache
- maint.reformatCluster
- maint.wipeCluster
- node.remove
- vserver.remove

PARAMETERS

- api: (string) One of 'maintenance' or 'preview'

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.system.enableAPI('maintenance')
success
```

NAME

system.enabledAPIs

SYNOPSIS

system.enabledAPIs() => array_of_APIs

DESCRIPTION

Returns a list of any enabled methods that could be disabled, any of the following:

- corefiler.remove
- corefiler.invalidate
- maint.unsetDataRepository
- maint.suspendAccess
- maint.invalidateCache
- maint.reformatCluster
- maint.wipeCluster
- node.remove
- vserver.remove

PARAMETERS

- No input parameters are required for this method.

RETURNS

- array_of_APIs: An array of any enabled methods that could be disabled.

EXAMPLE

```
print clientHandle.system.enabledAPIs
['corefiler.remove', 'corefiler.invalidate']
```

NAME

system.listMethods

SYNOPSIS

system.listMethods() => methodArray

DESCRIPTION

Returns a list of methods available through the Avere OS XML-RPC interface.

NOTE: Methods that require advanced levels of maintenance will only be listed if those levels have been enabled using the system.enableAPI method.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- methodArray: An array of any methods available. Note that the methods are listed by modules, and not in alphabetical order.

EXAMPLE

```
print clientHandle.system.listMethods()
['system.listMethods', 'system.methodHelp', 'system.methodSignature',
'system.multicall','system.login', 'system.logout',  'system.listModu
les', ... 'node.rename' ... 'node.listUnconfiguredNodes', 'dirService
s.modify' ... 'vserver.rename' ...]
```

NAME

system.listModules

SYNOPSIS

system.listModules() => moduleArray

DESCRIPTION

Returns a list of modules available through the Avere OS XML-RPC interface. "Modules" are the groups of methods; for example, "maint" and "node" are modules.

PARAMETERS

- No input parameters are required for this method.

RETURNS

- moduleArray: A list of any available modules.

EXAMPLE

```
print clientHandle.system.listModules()
['system', 'node', 'dirServices', 'stats', 'cifs', 'analytics', 'admin',
'support', 'maint', 'alert', 'vserver', 'cluster', 'nfs', 'migration',
'monitoring', 'corefiler']
```

NAME

system.login

SYNOPSIS

system.login(user_name, password) => status

DESCRIPTION

Logs an XML-RPC session in to the Avere cluster. A valid login session is required before other XML-RPC operations can be performed.

PARAMETERS

- user_name: (string) The user name for the cluster, represented as a base64-encoded string

- password: (string) The password for the cluster, represented as a base64-encoded string

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.system.login('YWRtaW4=','c3VwZXJzZWNyZXQ=')
success
```

NAME
system.logout

SYNOPSIS
system.logout() => status

DESCRIPTION
Logs an active XML-RPC session out of the Avere cluster. This is the recommended method for closing XML-RPC sessions to an Avere cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
```
print clientHandle.system.logout()
success
```

system.methodHelp

NAME
system.methodHelp

SYNOPSIS
system.methodHelp(methodName) => helpText

DESCRIPTION
Returns help text for the specified method.

NOTE: Methods that require advanced levels of maintenance will indicate that they are not available unless those levels have been enabled using the system.enableAPI method.

PARAMETERS
- methodName: (string) The method for which you want help.

RETURNS
- helpText: (string) Text that displays help for the method.

Adds or updates an ssh public key

EXAMPLE
print clientHandle.system.methodHelp('admin.addSshKey')
USAGE
admin.addSshKey(user,keyname,keyvalue) => status

PARAMETERS
- user: (string) The administrative account for which the key is being added. Only 'admin' is currently supported.
- keyname: (string) A unique keyname
- keyvalue: (string) An ssh key, consisting of the key-type, the key, and an optional identifier, separated by a space

RETURNS
- status: (string) Either 'success' or a reason for failure

EXAMPLE
print clientHandle.admin.addSshKey('admin','juser',
'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQ...XVwubT4SgTcqoP juser@machine')
success

NAME

system.methodSignature

SYNOPSIS

system.methodSignature(method) => signature

DESCRIPTION

Returns the signature of the specified method; that is, a text string listing the types of objects returned by the method.

EXAMPLE

```
print clientHandle.system.methodSignature('system.multicall')
string, array
```

NAME
system.multicall

SYNOPSIS
system.multicall(array_of_structs) => status

DESCRIPTION
Submits several XML-RPC methods as a single request.

PARAMETERS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs
  - methodName: The name of the method being called
  - params:

RETURNS
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.system.multicall([{'methodName': 'system.enableAPI',
  params: '['maintenance']'}, {methodName: 'node.get', params: 'myNewNode'}])
[{'myNewNode': {'name': 'myNewNode', 'alternateImage': 'AvereOS_V3.0.0
.4-d8808fc', 'clusterIPs': [{'IP': '10.1.16.000'}, {'IP': '10.1.16.001
'}], 'rev': '0859b7d5-f9ef-11e2-9875-000c299a83be', 'state': 'pending',
'primaryClusterIP': {'IP': '10.1.16.003'}, 'clientFacingIPs': {'current
-vserver': [{'IP': '10.1.125.004'}, {'IP': '10.1.125.005'}], 'vserver1'
: [{'IP': '10.1.16.006'}, {'IP': '10.1.16.007'}], 'new-vserver': [{'IP'
: '10.1.22.008'}, {'IP': '10.1.22.009'}, {'IP': '10.1.22.010'}]}, 'id':
'985c1cd2-9710-11e2-90b3-000c299a83be', 'activeImage': 'AvereOS_V3.0.0.
4-d51e414'}}]
```

system.setInt64Representation

NAME

system.setInt64Representation

SYNOPSIS

system.setInt64Representation(type) => status

DESCRIPTION

Sets the default type of integer representation for the current XML-RPC session.
If an XML-RPC session sets a value other than default and the session
is then closed, the integer representation is set to this default when the
next XML-RPC login session is started.

PARAMETERS

- type: (string) The default type, one of the following:
  - 'default': Eight-byte signed integer (i8)
  - 'int': Four-byte signed integer (i4)
  - 'i4': Four-byte signed integer
  - 'i8': Eight-byte signed integer (the default; can also be specified by the value 'default')
  - 'double': Double-precision signed floating-point number
  - 'float': Signed floating-point number
  - 'string': String (that is, the number represented as a string)

RETURNS

- status: (string) Either 'success' or a reason for failure

EXAMPLE

```
print clientHandle.system.setInt64Representation('i8')
success
```

vserver.addClientIPs

NAME

vserver.addClientIPs

SYNOPSIS

vserver.addClientIPs(vserverName, range) => status

DESCRIPTION

Adds a range of client-facing IP addresses to a vserver in
an advanced-networking VLAN configuration. The new range's name is set by the system.

PARAMETERS

- vserverName: (string) The name of the vserver

- range: (struct) Must include all of the following name:value pairs:

  - firstIP: (string) The first address in the range of client-facing IP addresses
  - netmask: (string) The netmask for the client VLAN
  - lastIP: (string) The last address in the range of client-facing IP addresses
  - vlan: (string) The name of the VLAN on which the IP addresses are located

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason
  for failure. If the activity is not complete, the activity UUID, which
  can be used as input for the cluster.getActivity and cluster.abortActivity
  methods.

EXAMPLE

```
print clientHandle.vserver.addClientIPs('bunny', {'firstIP': '10.1.25.226',
  'netmask': '255.255.255.0', 'lastIP': '10.1.25.229', 'vlan': '23'})
f83c2c2d-a649-11e3-9bd8-00259014ce64
```

vserver.addJunction

NAME
vserver.addJunction

SYNOPSIS
vserver.addJunction(vserverName, junctionPath, filerName, nfsExport, [advancedParameters]) => status

DESCRIPTION
Creates a junction between filesystems on a GNS-enabled vserver.

PARAMETERS
- vserverName: (string) The name of the GNS-enabled vserver on which the junction is to be created
- junctionPath: (string) The junction path that is to be created
- filerName: (string) The name of the core filer that is the source of the junction
- nfsExport: (string) The name of the NFS export that is the source of the junction on the core filer specified by the 'filerName' parameter

- [advancedParameters]:
An optional XML-RPC struct that contains the following name:value pairs for advanced junction settings:

- subdir: (string) The subdirectory that is the source of the junction within the specified NFS export specified by the 'export' parameter

- createSubdirs: (string) Optional. Determines whether any missing subdirectories specified in the 'subdir' parameter will be created ('yes') or not ('no' which is the default).

- access: (string) The access-control mechanism for the share, one of the following:
- 'posix' for POSIX mode bits (the default)
- 'nfsv4' for NFSv4 ACLs
- 'cifs' for CIFS ACLs

- sharename: (string) The core filer CIFS share name that provides access to the junction
This parameter is optional when 'access' is set to 'posix' or 'nfsv4'.

- [sharesubdir]: (string) Optional. The core filer subdirectory within the CIFS share that provides access to the junction

- [permissions]: (string) Optional. Permission change for the share, one of the following:
- 'preserve' so that no changes are made to the current POSIX mode bits or ACL (the default for junctions with POSIX Mode bit access or NAS core filer)
- 'modebits' to force the NFS owner to be 0, the NFS group owner to be 0, and the mode bits to be 0777
- 'cifsacl' to force the CIFS owner to be an "Administrator", and to use a default access control of "Everyone Full Control", which is the default for CIFS ACL junctions on a cloud core filer. This permission can only be used with cloud filers with CIFS enabled.

- [inheritPolicy]: (string) Optional. Determines the method used to assign an NFS export policy to the junction.
- 'yes' to use the NFS export policy associated with the containing NFS export
- 'no' to specify the NFS export policy in the 'policy' parameter

- [policy]: (string) Optional. The NFS export policy to apply to the junction when 'inheritPolicy' is set to 'no'.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.addJunction('gns', '/workspace', 'thor',
  '/vol0/juser', {'subdir': 'project1', 'createSubdirs':'no', 'policy':'user_policy', 'inheritPolicy':'no'})
success
```

NAME

vserver.clientSuspendJunction

SYNOPSIS

vserver.clientSuspendJunction(vserverName, path, clientSuspend) => status

DESCRIPTION

Suspend client access to the junction.
See the vserver.clientSuspendStatus method for client suspend response options.

PARAMETERS

- vserverName: (string) The name of the GNS-enabled vserver on which the junction is located
- path: (string) The junction path
- clientSuspend: (string) Set to 'yes' to suspend client access to the junction.

RETURNS:

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.vserver.clientSuspendJunction('vserver1', '/gns', 'yes')
success
```

vserver.clientSuspendStatus

NAME

vserver.clientSuspendStatus

SYNOPSIS

vserver.clientSuspendStatus(vserverName, status) => status

DESCRIPTION

This method determines the response for client operations that access a junction or core filer that has client suspend enabled. See the methods vserver.clientSuspendJunction() and corefiler.modify() with the clientSuspend option for client suspend enable details.

PARAMETERS

- vserverName: (string) The name of the GNS-enabled vserver on which the junction is located
- status: (string) 'JUKEBOX' (default), 'IO', or 'SUSPEND'.
  A value of 'SUSPEND' sets the NFS client response to an immediate TCP disconnect.
  A value of 'JUKEBOX' sets the NFS client response to an immediate reply status of EJUKEBOX.
  A value of 'SUSPEND' or 'JUKEBOX' sets the CIFS client response for idempotent operations
  to a reply status of MORE_PROCESSING_REQUIRED after a 55 second delay.
  A value of 'SUSPEND' or 'JUKEBOX' sets the CIFS client response for non-idempotent and modifying
  operations to a reply status of IO_DEVICE_ERROR after a 55 second delay.
  A value of 'IO' sets the NFS client response to an immediate reply status of EIO.
  A value of 'IO' sets the CIFS client response to an immediate reply status of IO_DEVICE_ERROR.

RETURNS:

- status: (string) Either 'success' or a reason for failure.

EXAMPLE

```
print clientHandle.vserver.clientSuspendStatus('vserver1', 'JUKEBOX')
success
```

vserver.create

NAME
vserver.create

SYNOPSIS
vserver.create(vserverName, range) => status

DESCRIPTION
Adds a vserver to the cluster.

PARAMETERS
- vserverName: (string) The name of the vserver that is to be created

- range: An XML-RPC struct containing name:value pairs that depend on whether advanced networking is enabled on the cluster or not, and whether the vserver uses a global namespace (GNS).

  - If advanced networking is NOT enabled, the struct must contain the following name:value pairs:
    - firstIP: (string) The first address in the range of client facing IP addresses
    - lastIP: (string) The last address in the range of client facing IP addresses

  - If advanced networking IS enabled, 'range' must include an array of one or more structs, each of which contains the following name:value pairs:
    - firstIP: (string) The first address in the range of client facing IP addresses
    - lastIP: (string) The last address in the range of client facing IP addresses
    - vlan: (string) The name of the client facing VLAN
    - netmask: (string) The netmask for the client facing VLAN

  - If the vserver does NOT use a global namespace (GNS), you can use one of the following optional parameters to specify the name of the vserver's core filer:

    - [filerName]: (string) The name of the core filer, if it is already known to the cluster
    - [filerNetwork]: (string) The IP address or fully qualified domain name of the core filer, if it is not already associated with the cluster. If you specify the 'filerNetwork' parameter, also specify the 'filerKnown' boolean parameter.
    - [filerKnown]: (boolean) If 'filerNetwork' is specified, this value should be False, confirming that the core filer is not yet known to the cluster.

    WARNING: Do not specify this parameter under any other circumstances (for example, if the core filer is already known to the cluster or if the vserver enables a GNS), and do not specify this parameter with any value other than False.

  - If the vserver does NOT use a global namespace (GNS), you can specify the following optional settings for the vserver's core filer:

    - [settings]: An optional XML-RPC struct containing the following name:value pairs that may be specified for the vserver's core filer if the vserver does NOT use a global namespace (GNS).

      - extractFsidFromFilehandle: (string) A value of "yes" enables extracting the fsid from the filehandle for the newly created core filer associated with this non global namespace (non GNS) vserver. The default is "no".

WARNING: Only change this option from the default if the core filer does not return persistent FSID values in NFS attributes and the core filer does not provide a mechanism to assign persistent FSIDs.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.create('starTurn', {'firstIP': '10.1.25.229',
  'lastIP': '10.1.25.230'})
286383da-a89a-11e3-9bc6-001b21459eeb
```

vserver.get

NAME
vserver.get

SYNOPSIS
vserver.get(vserverArray) => vserverInfoStruct

DESCRIPTION
Returns detailed information for the specified vservers.

PARAMETERS
- vserverArray: An array of one or more vserver names

RETURNS
- vserverInfoStruct: An XML-RPC struct that contains name:value pairs about each vserver:

- <vserverName>:
(string) The name of the vserver

- infoStruct: An XML-RPC struct with the following name:value pairs, containing information about <vserverName>:
- hotClientPeriod:
(integer) The polling period for collection information on the most active clients on the vserver, in seconds. The default is 60 (1 minute).

- cifs: (string) Whether CIFS is enabled or not, 'no' (the default) or 'yes'
- rev: (deprecated) The vserver's revision number
- state: (string) Current operational state of the vserver, for example, 'subprocess reset' or 'Suspended'

- adminstate: (string) The vserver's administrative state, one of the following:
- 'online' if the vserver is available to the cluster
- 'offline' if the vserver is not available to the cluster, but still recognized by the cluster
- 'removing' if the vserver is in the process of being removed from the cluster
- 'removed' if the vserver was previously recognized by the cluster, but has been removed from the cluster configuration
- 'suspended' if the vserver is not available to the cluster, but is still recognized by the cluster
- 'flushing' if the cluster is transferring information from the cache to a core filer

- hotClientLimit: (integer) The maximum number of clients on the vserver for which hot client information will be collected. The default is 10.

- corefiler | mass:
(array) The core filer names associated with the vserver. The core filer names are strings. Note that parameters containing "mass" are deprecated, and only present for backward compatibility. "Corefiler" should be used for all new applications.

- hotClientEnabled:
(string) Whether information will be collected on hot clients, either 'no' (the default) or 'yes'

- clientFacingIPs:
(array) Each element is a struct with the following name:value pairs that depend on whether advanced networking is enabled on the cluster:

  - firstIP: (string) The first address in the range of client-facing IP addresses
  - netmask: (string) The netmask for the client-facing VLAN
  - [name]: (string) Advanced networking only. The name of the IP range
  - lastIP: (string) The last address in the range of client-facing IP addresses.
  - vlan: (string) Advanced networking only. The name of the VLAN.
  - group: (string) Advanced networking only. The name of the port group.
  - id: (deprecated) The vserver's UUID

- type: (string) The vserver type, either 'gns' or 'simple'
- name: (string) The name of the vserver

- clientSuspendStatus:
(string) The client suspend status. See the vserver.clientSuspendStatus method.

EXAMPLE

```
print clientHandle.vserver.get(['new_global', 'gns1'])
{'new_global':
 {'hotClientPeriod': '60',
  'cifs': 'yes',
  'rev': '6854df2a-1be5-11e3-8bef-002590208a30',
  'state': 'subprocess reset',
  'adminState': 'online',
  'hotClieentLimit': '10',
  'corefiler': ['read-now', 'thor', 'grape', 'read-jack'],
  'hotClientEnabled': 'no',
  'clientSuspendStatus': 'JUKEBOX',
  'clientFacingIPs': [
   {'firstIP': '10.1.27.189',
    'netmask': '255.255.224.0',
    'name': 'dataIP0',
    'lastIP': '10.1.27.196'}],
  'type': 'gns',
  'id': 'f2955baa-b48d-11e0-b94a-00259014ccb8',
  'name': 'new_global'},
  'gns1': {'hotClient
  Period': '90', 'cifs': 'yes','rev': '83391519-1be5-11e3-8bef-002
  590208a30', 'state': 'subprocess reset', 'adminState': 'online',
  'mass': ['grape'], 'hotClientEnabled': 'yes', 'clientFacingIPs':
  [{'firstIP': '10.1.30.98', 'netmask': '255.255.224.0', 'name': '
  dataIP0', 'lastIP': '10.1.30.100'}], 'type': 'simple', 'id': '09
  c5eb46-c747-11e2-a4ae-00259014ce64', 'name': 'gns1'}}
```

NAME

vserver.lastNetgroupUpdate

SYNOPSIS

vserver.lastNetgroupUpdate(vserverName, [raw]) => time

DESCRIPTION

Returns the time of the last netgroup update on a vserver .

PARAMETERS

- vserverName: (string) The name of the vserver

- [raw]: (boolean) Optional. Determines how the time is returned:
  - If True (the default), the time is returned as epoch seconds (UNIX timestamp), the number of seconds since January 1, 1970
  - If False, the returned time is formatted as 'DDD mmm dd hh:mm:ss yyyy'

RETURNS

- time: (integer | string) The time of the last netgroup update, an integer if epoch seconds, otherwise a string

EXAMPLE

```
print clientHandle.vserver.lastNetgroupUpdate('new_global', False)
Mon Mar 10 13:39:27 2014
```

NAME
vserver.list

SYNOPSIS
vserver.list() => vserverArray

DESCRIPTION
Returns the names of all vservers that are currently configured on the cluster.

PARAMETERS
- No input parameters are required for this method.

RETURNS
- vserverArray: - An array of currently configured vservers. Each vserver name is a string.

EXAMPLE
```
print clientHandle.vserver.list()
['rabbit', 'gns', 'gns1']
```

vserver.listClientIPHomes

NAME

vserver.listClientIPHomes

SYNOPSIS

vserver.listClientIPHomes(vserverName) => array_of_structs

DESCRIPTION

Lists information about client-facing IP addresses on a vserver.

PARAMETERS

- vserverName: (string) The name of the vserver

RETURNS

- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the client-facing IP addresses on the specified vserver:

  - current: Name of the node on which the IP address is currently located
  - ip: A client-facing IP address.
  - home: Name of the IP address's home node

EXAMPLE

```
print clientHandle.vserver.listClientIPHomes('rabbit')
[{'current': 'advantage32', 'ip': '10.1.25.228', 'home': 'None'},
 {'current': 'advantage32', 'ip': '10.1.27.202', 'home': 'advantage32'},
 {'current': 'advantage21', 'ip': '10.1.25.229', 'home': 'None'},
 {'current': 'advantage21', 'ip': '10.1.27.198', 'home': 'advantage21'}]
```

vserver.listCoreFilers

NAME
vserver.listCoreFilers

SYNOPSIS
vserver.listCoreFilers(vserverName) => array_of_structs

DESCRIPTION
Lists information about a core filer associated with a specified vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the core filer:
  - internalName: (string) A name used internally by the FXT cluster to identify the core filer
  - networkName: (string) The primary IP address or fully qualified domain name of the core filer
  - name: (string) The user-visible name of the core filer

EXAMPLE
```
print clientHandle.vserver.listCoreFilers('vserver1')
[{'internalName': 'mass1', 'networkName': 'grape.company.com', 'name': 'grape'}]
```

NAME
vserver.listJunctions

SYNOPSIS
vserver.listJunctions(vserverName) => array_of_structs

DESCRIPTION
Lists information about a junction associated with a GNS-enabled vserver.

PARAMETERS
- vserverName: (string) The name of the GNS-enabled vserver

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the junction on the specified vserver:

  - [sharesubdir]: (string) The optional subdirectory within the CIFS share that provides access to the junction

  - corefilerInternal | massInternal: (string) The internal name of the core filer that is the source of the junction. Note that parameters containing "mass" are deprecated, and only present for backward compatibility. "Corefiler" should be used for all new applications.

  - sharename: (string) The core filer CIFS share name that provides access to the junction This parameter is required when 'access' is set to 'cifs'

  - [access]: (string) The optional access-control mechanism for the share, one of the following:
    - 'posix' for POSIX mode bits (the default)
    - 'nfsv4' for NFSv4 ACLs
    - 'cifs' for CIFS ACLs

  - export: (string) The name of the NFS export that is the source of the junction on the specified core file

  - subdir: (string) The subdirectory on the NFS export for the junction

  - policy: (string) If the junction was added or modified with 'inheritPolicy' set to 'no' then 'policy' is the NFS export policy that was set. Otherwise this element is not returned.

  - inheritPolicy: (string) A value of 'no' indicates that an NFS export policy was specified during junction add or modify. Otherwise no value is returned and the NFS export policy of the containing NFS export applies to the junction.

  - path: (string) The junction path

  - clientSuspend: (string) Junctions suspended using vserver.clientSuspendJunction return 'yes' for this value.

  - corefiler | mass: (string) The name of the core filer that is the source of the junction. Note that parameters containing "mass" are deprecated, and

only present for backward compatibility. "Corefiler" should be used for all new applications.

EXAMPLE

```
print clientHandle.vserver.listJunctions('gns')
[{'sharesubdir': '',
  'corefilerInternal': 'mass20',
  'sharename': '',
  'access': 'posix',
  'export': '/vol/regression',
  'subdir': '',
  'inheritPolicy': 'no',
  'policy': 'vserver1.policy.user_policy',
  'path': '/my_filer',
  'mass': 'my_filer'},
{'sharesubdir': '', 'massInternal': 'mass39', 'sharename': '', 'access':
'posix', 'export': '/vol/toplevel', 'subdir': '', 'path': '/juser',
'mass': 'juser-read'}]
```

vserver.listJunctionsByCoreFiler

NAME
vserver.listJunctionsByCoreFiler

SYNOPSIS
vserver.listJunctionsByCoreFiler(filerName, nfs_export) => array_of_structs

DESCRIPTION
Lists information about a junction associated with an export directory on a specified core filer.

PARAMETERS
- filerName: (string) The name of the core filer
- nfs_export: (string) The name of the exported directory

RETURNS
- array_of_structs: An array of XML-RPC structs that contain the following name:value pairs about the junction on the specified core filer export directory:

- [sharesubdir]: (string) Optional. The subdirectory within the CIFS share that provides access to the junction

- [sharename]: (string) Optional. The CIFS share name that provides access to the junction This parameter is required when 'access' is set to 'cifs'

- [access]: (string) Optional. The access-control mechanism for the share, one of the following:
- 'posix' for POSIX mode bits (the default)
- 'nfsv4' for NFSv4 ACLs
- 'cifs' for CIFS ACLs

- vserver_name: (string) The name of the vserver

- export: (string) The name of the NFS export that is the source of the junction on the specified core filer

- subdir: (string) The subdirectory on the NFS export for the junction

- policy: (string) If the junction was added or modified with 'inheritPolicy' set to 'no' then 'policy' is the NFS export policy that was set. Otherwise this element is not returned.

- inheritPolicy: (string) A value of 'no' indicates that an NFS export policy was specified during junction add or modify. Otherwise no value is returned and the NFS export policy of the containing NFS export applies to the junction.

- path: (string) The junction path

- clientSuspend: (string) Junctions suspended using vserver.clientSuspendJunction return 'yes' for this value.
- corefiler | mass:
(string) The name of the core filer that is the source of the junction. Note that parameters containing "mass" are deprecated, and only present for backward compatibility. "Corefiler" should be used

for all new applications.

EXAMPLE

```
print clientHandle.vserver.listJunctionsByCoreFiler('grape', '/vol/nrob2')
[{'sharesubdir': 'test', 'sharename': 'nrob2', 'access': 'cifs', 'vserver':
'vserver1', 'export': '/vol/nrob2', 'subdir': 'test', 'path': '/nrob1/test',
'corefiler': 'grape', 'mass': 'grape'}]
```

NAME

vserver.makeCurrentHome

SYNOPSIS

vserver.makeCurrentHome(vserverName) => status

DESCRIPTION

Makes the FXT node on which each of the specified vserver's client-facing IP addresses are located the client-facing IP address's home node.

PARAMETERS

- vserverName: (string) The name of the vserver

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.listClientIPHomes('vserver8')
[{'current': 'node3', 'ip': '10.1.20.137', 'home': 'None'},
{'current': 'node765', 'ip': '10.1.20.138', 'home': 'None'}]

print clientHandle.vserver.makeCurrentHome('vserver8')
success

print clientHandle.vserver.listClientIPHomes('vserver8')
[{'current': 'node3', 'ip': '10.1.20.137', 'home': 'node3'},
{'current': 'node765', 'ip': '10.1.20.138', 'home': 'node765'}]
```

vserver.modifyClientIPHomes

NAME

vserver.modifyClientIPHomes

SYNOPSIS

vserver.modifyClientIPHomes(vserverName, settingStruct) => status

DESCRIPTION

Sets home nodes for client-facing IP addresses.

PARAMETERS

- vserverName: (string) The name of the vserver whose client-facing IP addresses are to be set
- settingsStruct: An XML-RPC struct that contains the following name:value pairs:

  - ip: The IP address that is to be assigned a home node.
  - home: The name of the home node. If an empty string or 'None' is specified, any existing home node is unset for the specified IP address.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.modifyClientIPHomes('vserver1',
  {'10.1.20.137': 'handynode', '10.1.20.138': 'node765'})
success
```

vserver.modifyClientIPs

NAME

vserver.modifyClientIPs

SYNOPSIS

vserver.modifyClientIPs(vserverName, range) => status

DESCRIPTION

Modifies the specified vserver's client-facing IPs addresses.

PARAMETERS

- vserverName: (string) The name of the vserver whose client-facing IP addresses are to be set

- range: An XML-RPC struct that contains the following name:value pairs, depending on whether advanced networking is enabled or not:

  - If advanced networking is NOT enabled, the 'range' parameter must include the following attributes:
    - firstIP: (string) The first IP address in the range of client-facing IP addresses
    - lastIP: (string) The last IP address in the range of client-facing IP addresses

  - If advanced networking IS enabled, the 'range' parameter must include:
    - firstIP: (string) The first address in the range of client-facing IP addresses
    - netmask: (string) Optional. The netmask for the client VLAN
    - name: (string) The name of the IP range, obtainable from the vserver.get method
    - lastIP: (string) The last address in the range of client-facing IP addresses
    - vlan: (string) Optional. The name of the client VLAN
    - group: (string) Optional. The name of the port group for this range.

RETURNS

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.modifyClientIPs('vserver1', {'firstIP':
  '10.1.20.136', 'lastIP': '10.1.20.138', 'name': 'dataIP0'})
ccb2052e-ab8a-11e3-8e3b-000c291f288f
```

NAME

vserver.modifyJunction

SYNOPSIS

vserver.modifyJunction(vserverName, path, filerName, nfsExport, [advancedParameters]) => status

DESCRIPTION

Modifies an existing junction between filesystems on a GNS-enabled vserver.

PARAMETERS

- vserverName: (string) The name of the GNS-enabled vserver on which the junction is located
- path: (string) The junction path
- filerName: (string) The name of the core filer that is the source of the junction
- nfsExport: (string) The name of the NFS export that is the source of the junction on the core filer

- [advancedParameters]:
  An optional XML-RPC struct that contains the following name:value pairs:

  - subdir: (string) The subdirectory that is the source of the junction within the specified NFS export specified by the 'export' parameter

  - createSubdirs: (string) Optional. Determines whether any missing subdirectories specified in the 'subdir' parameter will be created ('yes') or not ('no' which is the default).

  - access: (string) The access-control mechanism for the share, one of the following:
    - 'posix' for POSIX mode bits (the default)
    - 'nfsv4' for NFSv4 ACLs
    - 'cifs' for CIFS ACLs. If this option is used, a 'sharename' must be specified.

  - sharename: (string) The (optional) core filer CIFS share name that provides access to the junction. This parameter is required when 'access' is set to 'cifs'

  - sharesubdir: (string) The (optional) core filer subdirectory within the CIFS share that provides access to the junction. This parameter is required when 'access' is set to 'cifs'.

  - [permissions]: (string) Optional. Permission change for the share, one of the following:

    - 'preserve': no changes are made to the current POSIX mode bits or ACL present (the default)
    - 'modebits': force the NFS owner to be 0, the NFS group owner to be 0, and the mode bits to be 0777. This permission can only be used with cloud filers with CIFS enabled.
    - 'cifsacl' to force the CIFS owner to be an "Administrator", and to use a default access control of "Everyone Full Control", which is the default for CIFS ACL junctions on a cloud core filer. This permission can only be used with cloud filers that have CIFS enabled.

  - [inheritPolicy]: (string) Optional. Determines the method used to assign an NFS export policy to the junction.
    - 'yes' to use the NFS export policy associated with the containing NFS export

- 'no' to specify the NFS export policy in the 'policy' parameter

- [policy]: (string) Optional. The NFS export policy to apply to the junction when 'inheritPolicy' is set to 'no'.

RETURNS
- status: (string) Either 'success' or a reason for failure.

EXAMPLE
```
print clientHandle.vserver.modifyJunction('vserver1', '/users/my_directory',
  'grape', '/vol/users', {'access': 'cifs'})
success
```

NAME

vserver.removeClientIPs

SYNOPSIS

vserver.removeClientIPs(vserverName, rangeName) => status

DESCRIPTION

Removes a range of client-facing IP addresses from an advanced networking VLAN configuration.

PARAMETERS

- vserverName: (string) The name of the vserver that has the VLAN configuration
- rangeName: (string) The name of the IP address range, which is returned by the vserver.get method

RETURNS:

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.removeClientIPs('vserver1', 'dataIP0')
94283a41-ab8f-11e3-8e3c-000c291f288f
```

NAME

vserver.removeJunction

SYNOPSIS

vserver.removeJunction(vserverName, path) => status

DESCRIPTION

Removes a junction from a GNS-enabled vserver. The method does not delete any data.

PARAMETERS

- vserverName: (string) The name of the GNS-enabled vserver
- path: (string) The junction path

RETURNS:

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.removeJunction('gns', '/workplace')
success
```

NAME

vserver.rename

SYNOPSIS

vserver.rename(vserverName, newname) => status

DESCRIPTION

Renames a vserver with the specified new name.

PARAMETERS

- vserverName: (string) The original name of the vserver
- newname: (string) The new name for the vserver

RETURNS:

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE

```
print clientHandle.vserver.rename('vserver1', 'starTurn')
success
```

NAME
vserver.suspend

SYNOPSIS
vserver.suspend(vserverName) => status

DESCRIPTION
Suspends the specified vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS:
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.vserver.suspend('starTurn')
success
```

NAME

vserver.unhomeAllClientIPs

SYNOPSIS

vserver.unhomeAllClientIPs(vserverName) => status

DESCRIPTION

Unassigns all client-facing IP addresses for the specified vserver from their home nodes.

PARAMETERS

- vserverName: (string) The name of the vserver

RETURNS:

- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

NAME
vserver.unsuspend

SYNOPSIS
vserver.unsuspend(vserverName) => status

DESCRIPTION
Unsuspends a previously suspended vserver.

PARAMETERS
- vserverName: (string) The name of the vserver

RETURNS:
- status: (string) If the activity is complete, either 'success' or a reason for failure. If the activity is not complete, the activity UUID, which can be used as input for the cluster.getActivity and cluster.abortActivity methods.

EXAMPLE
```
print clientHandle.vserver.unsuspend('starTurn')
success
```

Deprecated Commands

Deprecated command	Replacement
cifs.deleteShare	cifs.removeShare
cifs.newShare	cifs.addShare
cluster.createSchedule	cluster.addSchedule
cluster.deleteSchedule	cluster.removeSchedule
cluster.getDataParameters	cluster.getHADataParameters
cluster.moveDataParameters	cluster.moveHADataParameters
cluster.setDataParameters	cluster.setHADataParameters
corefiler.enableLocalDirectories	cachePolicy.modify
corefiler.modifyCachePolicy	cachePolicy.modify
corefiler.modifyCacheQuota	cachePolicy.modify
corefiler.modifyWriteMode	cachePolicy.modify
corefiler.setNetwork	cluster.addNetwork
corefiler.unsetNetwork	cluster.removeNetwork
dirServices.directoryPoll	dirServices.loginPoll
dirServices.getAttributes	dirServices.get
dirServices.modifyAttributes	dirServices.modify
maint.startStatsMonitoring	
maint.statsMonitoringOn	
maint.stopStatsMonitoring	
mcd.changePassword	
mcd.destroyCluster	
monitoring.enableSyslogServer	
monitoring.syslogServerEnabled	
nfs.createPolicy	nfs.addPolicy
stats.deleteHistoryConfig	
stats.newHistoryConfig	
stats.pauseRecording	
stats.resumeRecording	
support.supportMode	
vserver.listJunctionsByMass	vserver.listJunctionsByCoreFiler
vserver.listMasses	vserver.listCoreFilers
analytics.createCacheReport	
analytics.genCacheReportPlot	
analytics.getCacheReport	
analytics.getCacheReportHistogram	
analytics.listCacheReports	

The LZ4HC option for cloud object compression is no longer supported in Avere OS and the configuration option is no longer valid in the API.

See the 4.7 Release Notes on the Avere Library site for more information about updated XML-RPC calls.

Sample XML-RPC Client Application

This section lists the source code for sample XML-RPC client modules.

Python Client

The following client is written and run in Python. It works with both Python 2.6 and 2.7.

`xmlrpcClt.py`, an XML-RPC client module in Python

```
#!/usr/bin/env python
#
# xmlrpcClt.py
# Sample XML-RPC client module in Python
# Copyright 2010-2011 Avere Systems, Inc.

import os
import base64
import xmlrpclib
import urllib2
import cookielib

class CookieAuthXMLRPCTransport(xmlrpclib.Transport):
      """ xmlrpclib.Transport that sends HTTP cookie"""
      def __init__(self, use_datetime=0):
            self._use_datetime = use_datetime
            self._hdrs = []

      def send_cookie_auth(self, connection):
            """Include Cookie Authentication data in a header"""
            cookiestr = None
            for hdr in self._hdrs:
                  if cookiestr:
                        cookiestr = "%s; %s" %(cookiestr, hdr.rstrip(','))
                  else:
                        cookiestr = hdr.rstrip(',')
            if cookiestr:
                  connection.putheader("Cookie", cookiestr)
## override the send_host hook to also send authentication info

      def send_host(self, connection, host):
            xmlrpclib.Transport.send_host(self, connection, host)
            self.send_cookie_auth(connection)
      def request(self, host, handler, request_body, verbose=0):
            # issue XML-RPC request
            h = self.make_connection(host)
            if verbose:
                  h.set_debuglevel(1)
            self.send_request(h, handler, request_body)
            self.send_host(h, host)
            self.send_user_agent(h)
            self.send_content(h, request_body)
            errcode, errmsg, headers = h.getreply()
```

```
            if errcode != 200:
                  raise xmlrpclib.ProtocolError(host + handler, errcode, errmsg,
            headers)
            self._hdrs = headers.getheaders('Set-Cookie')
            self.verbose = verbose
            try:
                  sock = h._conn.sock
            except AttributeError:
                  sock = None
            return self._parse_response(h.getfile(), sock)

      def getXmlrpcClient(server_uri):
            """ this will return an xmlrpc client which supports """
            """ authentication through cookies """
            trans = CookieAuthXMLRPCTransport()
            client = xmlrpclib.Server(server_uri, transport=trans, verbose=False)
            return client
```

Assuming that the `xmlrpcClt.py` file is in your current directory, you can run it on Python as follows:

```
$ python
Python 2.6 (r26:66714, Jun 8 2009, 16:07:29)
[GCC 4.4.0 20090506 (Red Hat 4.4.0-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import xmlrpcClt
>>> import base64
>>> s=xmlrpcClt.getXmlrpcClient("http://<cluster_management_IP_addr>/cgi-
bin/rpc2.py")
>>> u=base64.encodestring("admin")
>>> p=base64.encodestring("password")
>>> res=s.system.login(u, p)
>>> print res
success
>>> res = s.system.listMethods()
>>> print res
['system.listMethods', 'system.methodHelp', 'system.methodSignature',
'system.multicall', 'system.login', 'system.logout', 'system.listModules',
'system.setInt64Representation', 'node.rename', 'node.allowToJoin', 'node.
getHardwareInfo', 'node.suspend', 'node.restartService', 'node.modifyIPMI'
, 'node.uptime', 'node.reboot', 'node.online', 'node.performance', 'node.u
nsuspend', 'node.get', 'node.powerdown', 'node.listUnconfiguredNodes', 'no
de.offline', 'node.list', 'node.cpu', 'dirServices.adLookup', 'dirServices
.netgroupPoll', 'dirServices.get', 'dirServices.modify', 'dirServices.setL
dapPassword', 'dirServices.directoryPoll(deprecated)', 'dirServices.modify
Attributes(deprecated)', 'dirServices.downloadCert''dirServices.getAttribu
tes(deprecated)', 'dirServices.usernamePoll', 'dirServices.usernameMapPoll
', 'stats.newHistoryConfig(deprecated)', 'stats.enableHotCollection', 'sta
ts.pauseRecording(deprecated)', 'stats.get', 'stats.disableHotCollection',
'stats.list', 'stats.hotClients', 'stats.isHotCollectionEnabled', 'stats.r
esumeRecording(deprecated)', 'stats.deleteHistoryConfig(deprecated)', 'sta
ts.getHistoryConfig', 'stats.hotFiles', 'stats.activeClients', 'stats.list
```

```
HistoryConfigs', 'stats.listConnections', 'stats.history', 'cifs.getAdStat
us', 'cifs.enable', 'cifs.configure', 'cifs.addShare', 'cifs.listShares',
'cifs.disable', 'cifs.removeShare', 'cifs.deleteShare(deprecated)', 'cifs.
isEnabled', 'cifs.modifyShare', 'cifs.getShare', 'cifs.getConfig', 'cifs.g
etJoinStatus', 'cifs.newShare(deprecated)', 'admin.changePermission', 'adm
in.addUser', 'admin.permission', 'admin.changePwd', 'admin.listUsers', 'ad
min.removeUser', 'support.taskIsDone', 'support.uploadCores', 'support.get',
'support.executeAdvancedMode', 'support.stopAdvancedMode', 'support.modify
', 'support.listAdvancedModes', 'support.updatePackage', 'support.testUplo
ad', 'support.executeNormalMode', 'support.listCores', 'support.removeCore
s', 'support.listNormalModes', 'support.taskComplete', 'maint.statsMonitor
ingOn', 'maint.isAccessSuspended', 'maint.unsuspendAccess', 'maint.stopSta
tsMonitoring', 'maint.startStatsMonitoring', 'alert.get', 'alert.dismiss',
'alert.conditions', 'alert.events', 'alert.history', 'vserver.rename', 'vs
erver.suspend', 'vserver.addClientIPs', 'vserver.lastNetgroupUpdate', 'vse
rver.removeClientIPs', 'vserver.modifyClientIPHomes', 'vserver.listMasses',
'vserver.create', 'vserver.listClientIPHomes', 'vserver.unsuspend', 'vserv
er.get', 'vserver.modifyClientIPs', 'vserver.listJunctions', 'vserver.make
CurrentHome', 'vserver.modifyJunction', 'vserver.unhomeAllClientIPs', 'vse
rver.removeJunction', 'vserver.list', 'vserver.addJunction', 'cluster.setD
ataParameters', 'cluster.addClusterIPs', 'cluster.listActivities', 'cluste
r.activateAltImage', 'cluster.modifyIPMI', 'cluster.disableVMwareOptimizat
ion', 'cluster.upgrade', 'cluster.getDataParameters', 'cluster.modifyClust
erIPNumPerNode', 'cluster.addSchedule', 'cluster.listSchedules', 'cluster.
activities', 'cluster.modifyNodeMgmtIPs', 'cluster.removeNodeMgmtIPs', 'cl
uster.get', 'cluster.powerdown', 'cluster.addVLAN', 'cluster.modifyCluster
IPs', 'cluster.getVLAN', 'cluster.cancelUpgrade', 'cluster.enableAdvancedN
etworking', 'cluster.rename', 'cluster.deleteSchedule(deprecated)', 'clust
er.restartService', 'cluster.moveDataParameters', 'cluster.getStaticRoutes',
'cluster.modifyAvereDataParameters(deprecated)', 'cluster.reboot', 'cluste
r.modifyVLAN', 'cluster.removeClusterIPs', 'cluster.enableVMwareOptimizati
on', 'cluster.modify', 'cluster.createSchedule(deprecated)', 'cluster.upgr
adeStatus', 'cluster.disableHA', 'cluster.removeSchedule', 'cluster.enabl
eHA', 'cluster.listVLANs', 'cluster.modifyStaticRoutes', 'cluster.addNodeM
gmtIPs', 'cluster.getActivity', 'cluster.modifySchedule', 'cluster.abortAc
tivity', 'cluster.removeVLAN', 'nfs.listExports', 'nfs.addRule', 'nfs.list
Policies', 'nfs.createPolicy(deprecated)', 'nfs.modifyPolicy', 'nfs.addPol
icy', 'nfs.removeRule', 'nfs.modifyRule', 'nfs.getExportPolicy', 'nfs.list
Rules', 'nfs.removePolicy', 'monitoring.testSyslog', 'monitoring.syslogSer
ver', 'monitoring.modifySnmpSettings', 'monitoring.modifyEmailSettings',
'monitoring.setSyslogServer', 'monitoring.emailSettings', 'monitoring.snmp
Settings', 'monitoring.testEmail', 'mass.rename', 'mass.unsuspend', 'mass.
get', 'mass.modify', 'mass.create', 'mass.list', 'mass.modifyWriteMode',
'mass.getCacheQuota', 'mass.modifySnapshotName', 'mass.modifiedFilesInfo',
'mass.getBandwidthThrottle', 'mass.modifyCacheQuota', 'mass.modifyBandwidt
hThrottle']
>>> print s.vserver.list()
['vserver5', 'defiant', 'enterprise', 'centos53', 'nx01']
>>>
```

Perl Client

The following client is written and run in Perl. The client requires that the Perl XML-RPC libraries are installed on the machine. On Fedora-based Linux systems, you can use the following command to install the required libraries:

```
% yum install perl-RPC-XML
```

`xmlrpcClt.pl`, an XML-RPC client module in Perl

```
#!/usr/bin/env perl
#
# xmlrpcClt.pl
# Sample XML-RPC client module in Perl; reads and reports statistics
# Copyright 2011 Avere Systems, Inc.

require RPC::XML::Client;
require MIME::Base64;

if ($#ARGV < 2) {
      print "Usage: cluster_mgmt_ip username password\n";
      exit(1);
}

$cli = RPC::XML::Client->new("http://$ARGV[0]/cgi-bin/rpc2.py");
$cli->useragent()->cookie_jar({});

$resp = $cli->send_request('system.login',
MIME::Base64::encode_base64($ARGV[1]),
MIME::Base64::encode_base64($ARGV[2]));
if ($resp->value() ne 'success') {
      die 'Could not log in'
}

while (1) {
      $resp3 = $cli->send_request('stats.get', 'summary');
      print $resp3->{'nfs_front_call'}->value(), "\n";
      sleep 2;
}
```

Assuming that the xmlrpcClt.pl file is in your current directory, you can run it as follows:

```
% ./xmlrpcClt.pl 10.0.14.20 admin admin
209030857.0
209033072.0
^C
```