Oracle Storage Connect plug-in Development Guide

Storage Array

Revision 1.2.8-BETA

SPECIFICATION, DEVELOPMENT AND DISTRIBUTION LICENSE AGREEMENT

"We," "us," and "our" refers to Oracle USA, Inc., for and on behalf of itself and its subsidiaries and affiliates under common control. "You" and "your" refers to the individual or entity that wishes to use materials from Oracle. "Programs" refers to the software product you wish to download and use and program documentation. “Specification” refers to the specification associated with the Programs that you wish to download and use. “Redistributables” refers to the materials associated with the program that are also identified in the Program documentation as redistributable. “Licensed Materials” refers to the Programs, Specification, and/or Redistributables. "License" refers to your rights to use the Licensed Materials under the terms of this agreement. This agreement is governed by the substantive and procedural laws of California. “Sample Code” refers to modules identified as sample code and which are licensed under the open source license included in their header files. You and Oracle agree to submit to the exclusive jurisdiction of, and venue in, the courts of San Francisco or Santa Clara counties in California in any dispute arising out of or relating to this agreement.

We are willing to license the Licensed Materials to you only upon the condition that you accept all of the terms contained in this agreement. Read the terms carefully and select the "Accept" button at the bottom of the page to confirm your acceptance. If you are not willing to be bound by these terms, select the "Do Not Accept" button and the registration process will not continue.

License Rights

We grant you a nonexclusive, nontransferable limited license to use the Programs and Specifications in unmodified form for purposes of developing your plug-in applications that interface with the Programs.

We grant you a nonexclusive, nontransferable right to copy and distribute the Redistributables in unmodified form with your plug-in application solely for the purpose of allowing your plug-in application to interface with the Program. Prior to distributing the Redistributables you shall require your end users to execute an agreement binding them to terms consistent with those contained in this section and the sections of this agreement entitled "License Rights," "Ownership and Restrictions," "Export," "Disclaimer of Warranties and Exclusive Remedies," "No Technical Support," "End of Agreement," and "Relationship Between the Parties." You must also include a provision specifying us as a third party beneficiary of the agreement. You are responsible for obtaining these agreements with your end users. We may audit your use of the Licensed Materials. Documentation is either shipped with the Programs, or documentation may be accessed online at http://otn.oracle.com/docs.

If you want to use the Licensed Materials for any purpose other than as expressly permitted under this agreement you must contact us to obtain the appropriate license.

You may terminate this agreement by destroying all copies of the programs. We have the right to terminate your right to use the Licensed Materials if you fail to comply with any of the terms of this agreement, in which case you shall destroy all copies of the programs.

Relationship Between the Parties

The relationship between you and us is that of licensee/licensor. Neither party will represent that it has any authority to assume or create any obligation, express or implied, on behalf of the other party, nor to represent the other party as agent, employee, franchisee, or in any other capacity. Nothing in this agreement shall be construed to limit either party's right to independently develop or distribute software that is functionally similar to the other party's products, so long as proprietary information of the other party is not included in such software.

Entire Agreement

You agree that this agreement is the complete agreement for the Licensed Materials, and this agreement supersedes all prior or contemporaneous agreements or representations. If any term of this agreement is found to be invalid or unenforceable, the remaining provisions will remain effective.

Structure of the documents

The development guide for the API is broken up into three different documents, namely the *General*, *Storage Array* (this document) and the *File System* documents. Please **first** see the ***General*** document as it gives information regarding both types of plug-ins and this document constantly refers back to the *General* document.

Implementing a Storage Array (IStorageArrayPlugin) plug-in

Class definition

Implementing a Storage Array (block device based) plug-in consists of creating a class that inherits from the `IStorageArrayPlugin` class in the `OSCPlugin` module. The name of the classes that implements the plug-ins need to be set in the `__all__` class variable, for example:

```
import OSCPlugin
from   OSCPlugin import *

class OraiSCSIPlugin(IStorageArrayPlugin):
    """Oracle iSCSI Storage Array Plugin"""
…
```

Class variables

The plug-in must set the following class variables that will be queried by the Oracle Storage Connect Plug-in Manager when discovering the plug-in: `plugin_name`, `vendor_name`, `plugin_version`, `plugin_desc`, `ss_extra_info_help`, `se_extra_info_help`, `storage_types` and `plugin_ability`. For example:

```
plugin_name        = "Oracle iSCSI Storage Server"
vendor_name        = "Oracle "
plugin_version     = "1.0.1-1"
plugin_desc        = "Oracle iSCSI reference implementation"
ss_extra_info_help = "To enable SSL to the file server use: SSL=yes"
se_extra_info_help = "To enable space reservation use: FullReserve=yes"
storage_types      = [IStorageArrayPlugin.iSCSIStorage]
plugin_ability     = {"snapshot":               ABILITY_TYPES.INVALID,
                      "custom_snap_name":       ABILITY_TYPES.INVALID,
                      "snap_is_sync":           ABILITY_TYPES.INVALID,
                      "clone":                  ABILITY_TYPES.INVALID,
                      "custom_clone_name":      ABILITY_TYPES.INVALID,
                      "clone_is_sync":          ABILITY_TYPES.INVALID,
                      "resize":                 ABILITY_TYPES.INVALID,
                      "resize_is_sync":         ABILITY_TYPES.INVALID,
                      "splitclone":             ABILITY_TYPES.INVALID,
                      "splitclone_is_sync":     ABILITY_TYPES.INVALID,
                      "splitclone_while_open":  ABILITY_TYPES.INVALID,
                      "snapclone":              ABILITY_TYPES.INVALID,
                      "snapclone_is_sync":      ABILITY_TYPES.INVALID,
                      "require_storage_name":   ABILITY_TYPES.INVALID,
                      "access_control":         ABILITY_TYPES.INVALID,
                      "max_access_entries":     0}
```

The Oracle VM Manager will display the `plugin_name` attribute to the user to be able to identify the specific plug-in. The `vendor_name` attribute is self-explanatory. The `plugin_version` attribute is for use by the plug-in provider to version the plug-in; it is used in conjunction with the fully qualified plug-in class to uniquely identify any given plug-in. The plug-in version (`plugin_version` class variable) MUST exactly match the RPM version set in the `.spec` file and MUST be in the following format: `Major.Minor.Patch-Release`, for example "`1.0.1-1`". The `plugin_desc` variable is an open format description string for the plug-in; this string will displayed in the Oracle VM Manager at the time of Storage Array configuration. For `IStorageArrayPlugin` based plug-ins, the plug-in can opt to use `SANStorage`, `iSCSIStorage` or both. The `storage_types` class variable need to be set to the list of types the plug-in will support for example:

```
storage_types = [IStorageArrayPlugin.SANStorage]
```

OR

```
storage_types = [IStorageArrayPlugin.iSCSIStorage]
```

OR

```
storage_types = [IStorageArrayPlugin.iSCSIStorage,
                 IStorageArrayPlugin.SANStorage]
```

If the plug-in support more than one type of storage it needs to check the `storage_type` key in the `ss_record` every time a method is invoked to determine which type of storage is being addressed. If the plug-in only support a single type it is only necessary to check the `storage_type` in the `validate` method.

If the plugin will be using the `extra_info` fields, it should set the `ss_extra_info_help`, `fs_extra_info_help` and / or `file_extra_info_help` class variables instructing the user what the format and use of the specific `extra_info` field would be.

NOTE: Only set the help text for fields that will be used, for example, if the plugin will only use the Storage Server Record's `extra_info` field, only set the `ss_extra_info_help` class variable and leave the other two unset.

Raw data from the Storage Array can be cached to speed up the plugin operations. The cache is essentially a cache of caches and is implemented using the `IPlugin.cache.set()`, `IPlugin.cache.get()`, `IPlugin.cache.extend()` and `IPlugin.cache.clear()` methods.

The `plugin_ability dict` is used to determine what advanced snap and cloning features the plug-in support. Note that the `plugin_ability dict` is statically defined to indicate what the *plug-in* is capable of performing while the `dict` (even though the `dict` definition is exactly the same) returned by the `getCapabilites` method represent what the specific Storage Array that is targeted by the specific `ss_record`. The keys for the `plugin_ability dict` are as follows:

`snapshot`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online snapshot is not available) if the plug-in can create snapshots.
`custom_snap_name`	If the plug-in allow a custom snapshot name to be supplied, set this to `ABILITY_TYPES.YES`.

`snap_is_sync`	If the plug-in will always create a snapshot synchronously (i.e. when the plug-in method returns the snapshot operation is complete), set this to `ABILITY_TYPES.YES`.
`clone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online cloning is not available) if the plug-in can create direct clones (Note, this implies that a clone can be created without any intermediate storage requirement like a snapshot).
`custom_clone_name`	If the plug-in allow a custom clone name to be supplied, set this to `ABILITY_TYPES.YES`.
`clone_is_sync`	If the plug-in will always create a clone synchronously (i.e. when the plug-in method returns the clone operation is complete), set this to `ABILITY_TYPES.YES`.
`resize`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online resizing is not available) if the plug-in can resize a Storage Elements.
`resize_is_sync`	If the plug-in will always resize a Storage Element synchronously (i.e. when the plug-in method returns the resize operation is complete), set this to `ABILITY_TYPES.YES`.
`splitclone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online split cloning is not available) if the plug-in can split clones.
`splitclone_is_sync`	If the plug-in will always split the clones synchronously (i.e. when the plug-in method returns the split clone operation is complete), set this to `ABILITY_TYPES.YES`.
`splitclone_while_open`	If the plug-in allows the clones to be split while they are open and actively being used, set this to `ABILITY_TYPES.YES`.
`snapclone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online snap cloning is not available) if the plug-in can create a clone from an existing snapshot.
`snapclone_is_sync`	If the plug-in will always create the clone synchronously (i.e. when the plug-in method returns the clone from snap operation is complete), set this to `ABILITY_TYPES.YES`.
`require_storage_name`	If the plug-in require a storage name to be set to be able to communicate to the correct storage server (as in the case if the plug-in communicate to the Storage Server via a concentrator or appliance) set this to `ABILITY_TYPES.YES`.
`access_control`	This should be set to `ABILITY_TYPES.YES` if the plug-in can support access control to the Storage Elements.
`max_access_entries`	This must be set to the maximum (or `-1` if unlimited) number of access control entries can be set in any one access control group.

Class methods

When any method experience an error the method must raised an exception, there are no error return codes, if no exception is raised it is assumed that the method succeeded.

validate() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record to validate.

Return value:

N/A

The `validate` method is used to validate the Storage Server record completed by the user. In general, this would encompass making sure that the all the required keys for the specific Storage Array are supplied, especially if the plug-in require anything to be set in the `extra_info` field. If possible, the plug-in should connect to the Storage Array to verify that the storage is ready and that the specific storage type requested is licensed etc. If the method finds any discrepancy or is unable to verify that the storage is ready etc., it should raise the appropriate exception. The exact exception class that should be raised will depend on the exact error. If none of the predefined exception classes matches exactly the situation, the plug-in should raise the `InvalidStorageArrayEx` with the message set to something explanatory of what went wrong. Note, this method should not be called internally by the plug-in to validate the Storage Server record in any other method call. The plug-in will never be given an un-validated Storage Server record to any other method in the plug-in.

getCapabilities() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server to obtain the capabilities from.

Return value:

New `plugin_ability dict` for the specific Storage Array.

`getCapabilities` is used to determine the capabilities that are supported by the specific Storage Array at this moment in time. Below are all the keys that are expected in the `dict`. NOTE: The plug-in should set all the keys in the `dict` listed below. If the Storage Array does not support, or the license expired for the specific feature, it should be set to `ABILITY_TYPES.UNSUPPORTED`:

`snapshot`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online snapshot is not available) if the Storage Array can create snapshots.
`custom_snap_name`	If the Storage Array allow for a custom snapshot name to be supplied, set this to `ABILITY_TYPES.YES`.

`snap_is_sync`	If the Storage Array will always create a snapshot synchronously (i.e. when the plug-in method returns the snapshot operation is complete), set this to `ABILITY_TYPES.YES`.
`clone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online cloning is not available) if the Storage Array can create direct clones (Note, this implies that a clone can be created without any intermediate storage requirement like a snapshot).
`custom_clone_name`	If the Storage Array allow for a custom clone name to be supplied, set this to `ABILITY_TYPES.YES`.
`clone_is_sync`	If the Storage Array will always create a clone synchronously (i.e. when the plug-in method returns the clone operation is complete), set this to `ABILITY_TYPES.YES`.
`resize`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online resizing is not available) if the Storage Array can resize a Storage Elements.
`resize_is_sync`	If the Storage Array will always resize a Storage Element synchronously (i.e. when the plug-in method returns the resize operation is complete), set this to `ABILITY_TYPES.YES`.
`splitclone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online split cloning is not available) if the Storage Array can split clones.
`splitclone_is_sync`	If the Storage Array will always split the clones synchronously (i.e. when the plug-in method returns the split clone operation is complete), set this to `ABILITY_TYPES.YES`.
`splitclone_while_open`	If the Storage Array allows clones to be split while they are open and actively being used, set this to `ABILITY_TYPES.YES`.
`snapclone`	Set this to `ABILITY_TYPES.ONLINE` (or `ABILITY_TYPES.OFFLINE` if online snap cloning is not available) if the Storage Array can create a clone from an existing snapshot.
`snapclone_is_sync`	If the Storage Array will always create the clone synchronously (i.e. when the plug-in method returns the clone from snap operation is complete), set this to `ABILITY_TYPES.YES`.
`require_storage_name`	If the plug-in require a storage name to be set to be able to communicate to the correct storage server (as in the case if the plug-in communicate to the Storage Server via a concentrator or appliance) set this to `ABILITY_TYPES.YES`.
`access_control`	This should be set to `ABILITY_TYPES.YES` if the Storage Array can support access control to the Storage Elements.

`max_access_entries`	This must be set to the maximum (or `-1` if unlimited) number of access control entries can be set in any one access control group.

getInfo() [REQUIRED (BOTH FORMS) - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to obtain information from / about.
`se_record`	Yes	Storage Element to obtain information about. (Default is `None`)

Return value:

Either returns an updated `ss_record` or `se_record`, depending if a `se_record` was supplied when called.

`getInfo` is used to query the Storage Array about either the Storage Array itself or a specific Storage Element, if a `se_record` is supplied (i.e. `if se_record != None`). When `getInfo` is called with only the `ss_record` supplied, the expected behavior is to obtain information from the Storage Array and update the `ss_record`. Of particular interest would be the `storage_desc` and status fields in the `ss_record`. Normally right after the initial validate call, the Oracle VM Manager will call the `getInfo` call on the Storage Array to get the `storage_desc` and status fields filled in to show this in the Oracle VM Manager. Interesting information to have in the `storage_desc` field would be for instance the type and model number of the storage etc. Partners are welcome to put whatever info they think would be of use to the user into this field about the physical Storage Array (firmware revisions etc. comes to mind, but this is entirely up to the plug-in provider of what to put in this string).

If a `se_record` is also passed to the method, the method should obtain information on the specific Storage Element from the Storage Array and return an updated `se_record`.

getStorageNames() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating Storage Server to get all the storage names from.

Return value:

Return the list of available storage names or if not supported or required, an empty list (`[]`).

`getStorageNames` is used to obtain a list of available storage names from which the user should select the specific Storage Array this Storage Server record will address. This is intended for plug-ins and Storage Arrays that uses a concentrator or management appliance that manages multiple Storage Arrays and the name is used to distinguish between them. If the plug-in or Storage Array does not support multiple Storage Arrays or does not require it, the plug-in should just return an empty list (`[]`).

getAccessGroups() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to query for the access groups.
`se_record`	Yes	Optional Storage Element to query.

Return value:

Return the list of access groups from the Storage Array (or if a `se_record` is supplied for the specific Storage Element).

`getAccessGroups` is used to query the Storage Array to obtain the access groups already defined, if a Storage Element record is specified, the method should only return the list of access groups the specific Storage Element is currently presented to.

createAccessGroups() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to create the access group on.
`access_grps`	No	List of access groups to create on the Storage Array.

Return value:

Updated `access_grps` with the newly create access groups added to the `access_grps` list.

`createAccessGroups` creates the list of `access_grps` passed in on the Storage Array as well as adding all the successfully created groups to the Storage Server record's list of access groups (`access_grps`) and returning it. Note the `access_grps` in the `ss_record` cannot be updated directly, it needs to get updated and returned by the call.

removeAccessGroups() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description

`ss_record`	No	Storage Server record indicating which Storage Array from which to remove (delete) the access group from.
`access_grps`	No	List of access groups to remove (delete) from the Storage Array.

Return value:

Updated `access_grps` with the deleted access groups removed from the `access_grps` list.

`removeAccessGroups` deletes the list of `access_grps` passed in from the Storage Array as well as removing all the successfully deleted groups from the Storage Server record's list of access groups (`access_grps`) and returning it. Note the `access_grps` in the `ss_record` cannot be updated directly, it needs to get updated and returned by the call.

renameAccessGroup() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array from which to remove (delete) the access group from.
`access_grp_name`	No	Name of access group to rename on the Storage Array.
`new_access_grp_name`	No	New name for the access group.

Return value:

Updated `access_grps` with the access group renamed.

`renameAccessGroup` rename an Access Group named by `access_grp_name` to the new name specified by `new_access_grp_name` on the Storage Array as well as renaming the successfully renamed group in the Storage Server record's list of access groups (`access_grps`) Note the `access_grps` in the `ss_record` cannot be updated directly, it needs to get updated and returned by the call.

addToAccessGroup() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array the access group entries should be added to.
`access_grp_name`	No	Access group the entries should be added to.

grp_entries	No	The list of entries to add to the access group. (For example SAN this would WWNs, for iSCSI it would be initiator names etc.)

Return value:

Updated access_grp with the new entries added to the list of grp_entries.

addToAccessGroup adds new access control entries to the access group on the Storage Array and return the updated access group.

removeFromAccessGroup() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server record indicating which Storage Array to remove the access group entries from.
access_grp_name	No	Access group the entries should be added to.
grp_entries	No	The list of entries to remove from the access group. (For example SAN this would WWNs, for iSCSI it would be initiator names etc.)

Return value:

Updated access_grp with the entries remove from the list of grp_entries.

removeFromAccessGroup deletes access control entries from the access group on the Storage Array and return the updated access group.

discover() [OPTIONAL - ISCSI ONLY]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server record indicating which iSCSI Storage Array to obtain targets from.

Return value:

Return an updated ss_record with the storage_id field updated with the iSCSI targets obtained from the Storage Array.

discover is used only when the plug-in is dealing with an iSCSI storage type, this method will never be called for a SAN Storage Array. This method is a direct mapping to the iSCSI 'sendtargets' call. If the plug-in do not implement the method, a generic version will be used. Note that only the reachable targets (via `netdevs` if it is set in the `ss_record`) are stored in the `storage_id` field.

start() [OPTIONAL - ISCSI ONLY]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which iSCSI Storage Array to start (login).

Return value:

N/A

This method is also only of interest for plug-ins dealing with an iSCSI storage type; it will not be called for a SAN Storage Array. From an iSCSI storage perspective this is used to login to the iSCSI targets, note that the plug-in must take into account if CHAP authentication should be used (flagged via the `chap` key in the `ss_record`) and which network device should be used for the connection. If the plug-in do not implement the method, a generic version will be used that will login to all targets specified by the `storage_id` list in the `ss_record`.

stop() [OPTIONAL - ISCSI ONLY]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which iSCSI Storage Array to stop (logout).

Return value:

N/A

This method is also only of interest for plug-ins dealing with an iSCSI storage type; this method will not be called for a SAN Storage Array. From an iSCSI storage perspective, this is used to logout of the iSCSI targets; note that the plug-in must take into account that any multipath device layers need to be torn down before the logout can occur. If the plug-in wish, it can use the `destroyMPDev` method from the `OSCPluginUtils` module to tear down the multipath device. If the plug-in do not implement the method, a generic version will be used that will logout of all the targets specified by the `storage_id` list in the `ss_record`.

refresh() [OPTIONAL - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array is

		being refreshed.

Return value:

N/A

When a new LUN is presented or an existing LUN is un-presented to the host the SCSI layer needs to be refreshed to notice the SCSI bus changes. This is true for both SAN and iSCSI based Storage Arrays. If the plug-in does not implement this method a generic version of the method will be used that will do a full SCSI bus rescan for the specific storage type. There is a very good case to be made for a plug-in to not rely on the generic implementation since the plug-in can determine what changed on the Storage Array and target just that specific device that changed instead of kicking of a full bus rescan.

list() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to obtain the list of Storage Elements from.
`se_type`	Yes	The type(s) of Storage Elements that should be returned. (Defaults to `[IStorageArrayPlugin.LUNType, IStorageArrayPlugin.SnapCloneType]`

Return value:

List of `se_records` containing all the Storage Elements of the specified type(s) specified in the `se_type`.

This method should return all of the Storage Elements available on the Storage Array of type `se_type`. Note that the `se_type` will never refer to a snapshot type, only LUN types. Note: the `list` method does not and should not try to determine if the Storage Element is mapped to the host it is executing on, however it is **required** that the page83_id is filled in for the returned list of se_records. The plugin can format the page 83 id to be the required format from a raw page 84 id using the makeMPPage83FromRawPage83() function in OSCPluginUtils.

updateSERecords() [OPTIONAL - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array that owns the Storage Elements.
`se_records`	No	List of Storage Element records to examine and update with all the required information, this includes the local device path, if it is available on the server.

`append_allowed`	Yes	By default, the method is not allowed to add new Storage Element records to the list passed in, if this flag is set the method is allowed to add Storage Element records to the list for newly discovered Storage Elements. (Defaults to `False`)

Return value:

List of updated `se_records`.

This method should complete as much as possible information in each of the Storage Element records including the device path(s) on the server (for example `/dev/mapper/mpath1`) for the specific Storage Element. Note: the plug-in must use the `IPlugin.dev_path_prefix` class variable to determine which device path prefix to use, as it can change depending on whether multipath is enabled or not.

getStatus() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage server record indicating which Storage Array should be queried.
`se_record`	Yes	Storage Element to retrieve the status for (Defaults to `None`).

Return value:

String containing the status from the Storage Array or the Storage Element.

The method is used to get the status of the Storage Array or a specific Storage Element. This is intended to be a lighter weight call to get the status than the `getInfo` call. If the Storage Array does not have a light weight call, the plug-in can internally call the `getInfo` to obtain the status.

online() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to online.
`se_record`	No	Storage Element to set online.

Return value:

N/A

Set the specific Storage Element online. Note that this should not change the presentation of the Storage Element.

offline() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to offline.
`se_record`	No	Storage Element to set offline.

Return value:

N/A

Set the specific Storage Element offline. Note that the intention is not to change the presentation of the Storage Element.

create() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating on which Storage Array the Storage Element will be created.
`se_record`	No	Storage Element to create.
`thin_provision`	Yes	Indicate if the Storage Element should be thin provisioned (if the Storage Array support thin provisioning) or fully provisioned. (Defaults to `True`)
`qos`	Yes	Desired QoS for the newly created Storage Element (if the Storage Array does not support a QoS when creating the new Storage Element, it can safely be ignored but the plug-in should still fill in the `qos` in the newly created `se_record` with the value appropriate for the Storage Array). (Defaults to `None`)

Return value:

`se_record` for the newly created Storage Element.

The create method should create a new Storage Element on the Storage Array. Note that the `se_record` will not be a fully completed Storage Element record, the only fields that the plug-in can depend on to be filled in are the: `se_type`, `ss_uuid`, `size` and `vol_group_name`, and `name`. The plug-in should update and return the `se_record` with all the appropriate info it will require in the future to locate and operate on the Storage Element, this include the `qos` if supported by the Storage Array. If the `thin_provision` flag is set the plug-in should direct the Storage Array to use thin provisioning for the Storage Element if the Storage Array support thin provisioning. Note: it is **required** that the page83_id for the new Storage Element to be filled in for the returned se_record. The plugin can format the page 83 id to be the required format from a raw page 84 id using the makeMPPage83FromRawPage83() function in OSCPluginUtils.

startPresent() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to present.
`se_record`	No	The Storage Element to present.
`access_grp_names`	No	List of access group names the Storage Element should be presented to.

Return value:

`se_record` with all the currently active access group names updated for the Storage Element.

`startPresent` is called to start presenting the Storage Element specified in the `se_record` to the `access_groups` specified by the `access_grp_names` parameter and return the `se_record` with the `access_grp_names` updated.

stopPresent() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to stop presenting.
`se_record`	No	Storage Element to stop presenting.
`access_grp_names`	No	List of access group names the Storage Element should not be presented to anymore.

Return value:

`se_record` with all the currently active access group names updated for the Storage Element.

On a successful completion, the `se_record` will not be presented to any of the `access_groups` and the `se_record`'s `access_grp_names` updated.

resize() **[REQUIRED - iSCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to be resized.
`se_record`	No	Storage Element to resize.
`new_size`	No	New size for the Storage Element.

Return value:

Updated `se_record` with the new size.

The `resize` method is used to resize the Storage Element (larger or smaller); upon successful completion of the operation, an updated `se_record` with the actual size of the Storage Element will be returned.

remove() **[REQUIRED - iSCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to be removed.
`se_record`	No	Storage Element to remove.

Return value:

N/A

This method should remove the Storage Element from the Storage Array. Note: since this method is very destructive, it should make no automatic decisions on behalf of the caller, for example if the Storage Element is still presented it should just raise an exception and NOT automatically stop presenting the Storage Element to be able to remove it.

getCloneLimits() **[REQUIRED - iSCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to obtain the limit from.

se_record	Yes	Storage Element to obtain limit for (if specified). (Defaults to None)

Return value:

Maximum number of possible shallow clones (-1 for unlimited) supported by the Storage Array.

The method should obtain the maximum number of thin (shallow) clones that can be created for a Storage Element. If a se_record is supplied (i.e. se_record != None) the limits specific to this Storage Element should be returned (if different from the global limit), otherwise any global limit (Storage Array level) should be returned.

isCloneable() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server record indicating which Storage Array owns the Storage Element to check.
se_record	No	Storage Element to check if it can be shallow cloned.

Return value:

True if the Storage Element can be shallow (thin) cloned otherwise it should return False.

Check if the Storage Array can create a shallow (thinly provisioned) clone the Storage Element at this specific moment in time. This would take into account for example any limits that may be imposed on a specific Storage Element as well as any global Storage Array level limits.

clone() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server record indicating which Storage Array owns the Storage Element to be cloned.
se_record	No	Storage Element to create a clone from.
dest_se_record	Yes	Destination for the new Storage Element. This will not be supplied if the plug-in indicates that it is unable to accept user naming for clones (see plugin_ability dict).
qos	Yes	Desired QoS for the newly created clone (if the Storage Array does not support a QoS on the clone, it can safely be ignored but the plug-in should fill in the qos in the newly created se_record with the value appropriate for the Storage Array.

Return value:

Newly created `se_record` for the shallow (thin provisioned) clone.

If the Storage Array always needs an intermediate snapshot before a clone can be created, this method should create the snapshot and create the clone. If possible it should, remove the snapshot after the clone is created. In essence, the method will create a new shallow clone for the original LUN specified by the `se_record`. The `dest_se_record` is used to name the new clone (if supported by the Storage Array). Note that the `dest_se_record` will not be a fully completed Storage Element record, the only fields that the plug-in can depend on to be filled in are the: `se_type`, `ss_uuid` and `name`. Note: it is **required** that the page83_id for the new Storage Element to be filled in for the returned se_record. The plugin can format the page 83 id to be the required format from a raw page 84 id using the makeMPPage83FromRawPage83() function in OSCPluginUtils.

isSplittable() **[REQUIRED]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which storage server owns the Storage Element to check.
`se_record`	No	Storage Element to check if it can be split from its parent or peers.

Return value:

`True` if Storage Server can split the Storage Element from its parent (or peers), otherwise it should return `False`.

Check if the Storage Server can split (after the operation none of the Storage Element's blocks should be shared with any other Storage Element). If the operation would not be permanent, for example, the Storage Server supports automatic DEDUP, the method should return `False`.

splitClone() **[REQUIRED – ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Elements to be split.
`se_record`	No	Storage Element to split from its parent.

Return value:

N/A

The `splitClone` method is called to split two dependent clones (i.e. create a deep copy clone from the shallow clone). This can be thought of the breaking the parent child relationship between two Storage Elements on the Storage Array so that they do not share the same storage anymore. If the Storage Array does not support deep copy clones the method should just raise the `NoSuchOperationEx` exception.

cloneFromSnap() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to be cloned.
`se_record`	No	Original Storage Element to be cloned.
`snap_se_record`	No	The Snapshot that will be used for the clone operation.
`dest_se_record`	Yes	Destination for the new Storage Element. This will not be supplied if the plug-in indicates that it is unable to accept user names for clones (see `plugin_ability dict`).
`qos`	Yes	QoS value for the newly created clone (if the Storage Array does not support a QoS on the clone, it can safely be ignore but the plug-in should fill in the `qos` in the newly created `se_record` with the value appropriate for the Storage Array.

Return value:

Newly created `se_record` for the shallow (thin provisioned) clone from the specific snapshot.

In essence, the method will create a new shallow clone for the original LUN specified by the `se_record` as it was frozen at the time of the snapshot `snap_se_record` . The `dest_se_record` is used to name the new clone (if supported by the Storage Array). Note that the `dest_se_record` will not be a fully completed Storage Element record, the only fields that the plug-in can depend on to be filled in are the: `se_type`, `ss_uuid` and `name`. Note: it is **required** that the page83_id for the new Storage Element to be filled in for the returned se_record. The plugin can format the page 83 id to be the required format from a raw page 84 id using the makeMPPage83FromRawPage83() function in OSCPluginUtils.

getCurrentClones() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to be queried.
`se_record`	Yes	Specific Storage Element (this can be any of the defined `se_types`) to query if supplied.

Return value:

A list of `se_records` for the current known clones on the Storage Array (global list) or if the `se_record` is supplied for this particular Storage Element.

The intent is to be able to determine either all the clones that currently exist on the Storage Array or the clones that exist for the specific Storage Element. If it is not possible to obtain this information from the Storage Array, the plug-in should raise the `NoSuchOperationEx` exception.

getSnapLimits() **[REQUIRED - ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to get the limit from.
`se_record`	Yes	Storage Element to get limit for (if specified). (Defaults to `None`)

Return value:

Maximum number of possible snapshots supported by the Storage Array (`-1` for unlimited).

The method should obtain the maximum number of snapshots that can be created for a Storage Element. If a `se_record` is supplied (i.e. `se_record != None`) the limit specific to this Storage Element should be returned (if it is different from the global), otherwise the global limit (Storage Array level) should be returned.

isSnapable() **[REQUIRED - ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to check.
`se_record`	Yes	Storage Element to check if it can be snapshot.

Return value:

`True` if snapshots can be created on the Storage Array or, if specified, for the specific Storage Element, otherwise it should return `False`.

Check if the Storage Array can create a snapshot for the Storage Element at this specific moment in time. This would take into account for example any limits that may be imposed on a specific Storage Element as well as any global Storage Array level limits. Note, it is not required that the snapshot be at the same level as the Storage Element, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

If the Storage Element is not specified, the plug-in should check if the Storage Array will allow creation of any snapshots at this time, for example if the feature is not licensed on the Storage Array etc. it would return `False`.

createSnap() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to be cloned.
`se_record`	No	Storage Element to snapshot.
`snap_se_record`	Yes	Optionally specify the name for the snapshot. (Defaults to `None`)

Return value:

A `se_record` for the newly created snapshot.

The `createSnap` method is used to create a snapshot for the Storage Element specified by the `se_record`. If the plug-in supports supplying a name for snapshots (communicated via the `plugin_ability` class variable) `snap_se_record` would contain the desired name for the snapshot. Note, the `snap_se_record` will not be a fully completed Storage Element record, the only fields that the plug-in can depend on to be filled in are the: `se_type`, `ss_uuid` and `name`. It is not required that the snapshot be at the same level as the Storage Element, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

createMultiSnap() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element to snapshot.
`se_records`	No	List of Storage Elements to snapshot.
`snap_se_record`	Yes	Optionally specify the name for the snapshot. (Defaults to `None`)

Return value:

A `se_record` for the newly created snapshot.

The `createMultiSnap` method is similar to the `createSnap` method with the exception that instead of taking a snapshot of a single Storage Element on the Storage Array, it will take a single snapshot of multiple Storage Elements on the array at the same time. If the Storage Array does not support taking a snapshot of multiple Storage Elements in a single operation, the plug-in should raise the `NoSuchOperationEx` exception. If the plug-in supports supplying a name for snapshots (communicated via the `plugin_ability` class variable) `snap_se_record` would contain the desired name for the snapshot. Note, the `snap_se_record` will not be a fully completed Storage Element record, the only fields that the plug-in can depend on to be filled in are the: `se_type`, `ss_uuid` and `name`. It is not required that the snapshot be at the same level as the Storage Element, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

isRestorable() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the snapshot.
`snap_se_record`	No	Snapshot to check if it can roll back.
`se_record`	Yes	Storage Element that would be rolled back if supplied.

Return value:

`True` if the Storage Array can safely roll back to the snapshot, otherwise it should return `False`.

`isRestorable` should determine if the Storage Array would be able to roll back to the snapshot identified by the `snap_se_record`. If the Storage Element is supplied, the check should make sure that this specific Storage Element can be rolled back using the snapshot AND that no other entity on the Storage Array would be affected by the restore operation.

snapRestore() [REQUIRED - iSCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the snapshot.
`snap_se_record`	No	Snapshot to roll back to.
`se_record`	Yes	Storage Element that should be rolled back if supplied.

Return value:

N/A

`snapRestore` is intended to roll back the Storage Array or if a Storage Element is supplied, just the Storage Element to the snapshot given in `snap_se_record`.

NOTE: Extremely important, if a Storage Element is specified and the Storage Array cannot roll back just this Storage Element to this particular snapshot then the method should just raise the `SnapRestoreNotSafeEx` exception. Under **NO** circumstance is the plug-in allowed to roll back a snapshot that would affect other Storage Elements on the Storage Array.

snapRemove() **[REQUIRED - ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the snapshot.
`snap_se_record`	No	Snapshot to remove from the Storage Array.

Return value:

N/A

`snapRemove` should remove / delete (if possible) the snapshot given in `snap_se_record`. If the snapshot cannot be removed because it is busy, the plug-in should raise the `StorageElementBusyEx` exception.

getCurrentSnaps () **[REQUIRED - ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the snapshot.
`se_record`	Yes	Specific Storage Element to query if supplied.

Return value:

A list of `se_records` for the current known snapshots on the Storage Array (global list) or if the `se_record` is supplied for this particular Storage Element.

The intent is to be able to determine either all the snapshots that currently exist on the Storage Array or the snapshots that exist for the specific Storage Element. If it is not possible to obtain this information from the Storage Array, the plug-in should raise the `NoSuchOperationEx` exception.

getQoSList() **[REQUIRED - ISCSI & SAN]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array to get QoS list from.

Return value:

List of `qos_vals` `dicts` with all the known Quality-of-Service values for the Storage Array.

`getQoSList` should create and return a list `qos_vals` `dicts` in subsequent calls the `value` attribute of the `dict` will be passed to methods that accept the QoS parameter.

setQoS() [REQUIRED - ISCSI & SAN]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which Storage Array owns the Storage Element.
`se_record`	No	Storage Element which QoS setting should be updated.
`qos`	No	New QoS value for the Storage Element.

Return value:

N/A

Update the Quality-of-Service value for the specified Storage Element. If the Storage Array does not allow changing the QoS value while the Storage Element is online, the plug-in should raise the `StorageElementBusyEx` exception. If the Storage Array does not support updating the QoS of the Storage Element, the plug-in should raise the `NoSuchOperationEx` exception.

getAsyncProgress() [REQUIRED IF ASYNC IS SUPPORTED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server.
`some_record`	No	Record previously returned from the call that started the asynchronous operation with the `async_progress` and if required the `async_handle` fields added (and not set to `None`).

Return value:

Fully completed record. This need to be the exact same record that would have been returned if the initial call completed synchronously except it have the `async_progress` field.

`getAsyncProgress` is a special call that is only called if and only if the original call (`clone()`, `resize()` etc.) is asynchronous. In the case when the operation is asynchronous on the Storage Server, the original call would add `async_progress` field and set the value to either `-1` (indicating the Storage Server is unable to give a percent complete for the operation) or a number between `0` and `100` indicating the percent complete for the operation. When the operation is completed, the value for the key must be set to `None` to indicate the operation is done. At this point the record will be returned to the application for processing. Note, the `async_handle` is not required or used by the caller. It is solely for use by the plugin, for example storing all the info required by the plugin to be able to locate and get status on the specific operation. Once the operation is completed, both these fields will be dropped out from the record returned to the Manager.