Oracle Storage Connect plug-in Development Guide

File System

Revision 1.2.10-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.

Each Sample Code module is licensed under the terms and conditions of the open source license included with that module.

Ownership and Restrictions
We retain all ownership and intellectual property rights in the Licensed Materials. You may make a sufficient number of copies of the Licensed Materials for the licensed use and one copy of the Program for backup purposes.

You may not:
- use the Licensed Materials for any purpose other than as provided above;
- distribute the Redistributables unless accompanied with your plug-in applications;
- remove or modify any markings or any notice of our proprietary rights that may appear in any of the Licensed Materials;

- use the Licensed Materials to provide third party training on the content and/or functionality of the Programs, except for training your licensed users;
- assign this agreement or give the Licensed Materials, Program access or an interest in the Licensed Materials to any individual or entity except as provided under this agreement;
- cause or permit reverse engineering (unless required by law for interoperability), disassembly or decompilation of the Programs;
- disclose results of any Program benchmark tests without our prior consent; or,
- use any Oracle name, trademark or logo.

Indemnification
You agree to: (a) defend and indemnify us against all claims and damages caused by your distribution of the programs in breach of this agreements and/or failure to include the required contractual provisions in your end user agreement as stated above; (b) keep executed end user agreements and records of end user information including name, address, date of distribution and identity of programs distributed; (c) allow us to inspect your end user agreements and records upon request; and, (d) enforce the terms of your end user agreements so as to effect a timely cure of any end user breach, and to notify us of any breach of the terms.

Export
You agree that U.S. export control laws and other applicable export and import laws govern your use of the programs, including technical data; additional information can be found on Oracle's Global Trade Compliance web site located at http://www.oracle.com/products/export/index.html?content.html. You agree that neither the programs nor any direct product thereof will be exported, directly, or indirectly, in violation of these laws, or will be used for any purpose prohibited by these laws including, without limitation, nuclear, chemical, or biological weapons proliferation.

Disclaimer of Warranty and Exclusive Remedies

THE LICENSED MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. WE FURTHER DISCLAIM ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT.

IN NO EVENT SHALL WE BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, DATA OR DATA USE, INCURRED BY YOU OR ANY THIRD PARTY, WHETHER IN AN ACTION IN CONTRACT OR TORT, EVEN IF WE HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. OUR ENTIRE LIABILITY FOR DAMAGES HEREUNDER SHALL IN NO EVENT EXCEED ONE THOUSAND DOLLARS (U.S. $1,000).

No Technical Support
Our technical support organization will not provide technical support, phone support, or updates to you for the programs licensed under this agreement.

Restricted Rights
If you distribute a license to the United States government, the Licensed Materials, including documentation, shall be considered commercial computer software and you will place a legend, in addition to applicable copyright notices, on the documentation, and on the media label, substantially similar to the following:

NOTICE OF RESTRICTED RIGHTS
"Programs delivered subject to the DOD FAR Supplement are 'commercial computer software' and use, duplication, and disclosure of the programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are 'restricted computer software' and use, duplication, and disclosure

of the programs, including documentation, shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software-Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065."

End of Agreement
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.

NOTE: *As of rev 1.2.10-BETA the File System part is obsoleted and is only included as a reference, no vendor supplied File System plugin will be accepted / supported.*

Structure of the documents

The development guide for the API is broken up into three different documents, namely the *General*, *Storage Array* and the *File System* (this document) 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 File System (IFileSystemPlugin) plug-in

Class definition

Implementing a file system plug-in consists of creating a class that inherits from the `IFileSystemPlugin` class in the `OSCPlugin` module. The name of the class that implements the plug-in need to be set in the `__all__` class variable, for example:

```
import OSCPlugin
from   OSCPlugin import *

__all__ = ["OraNFSPlugin"]

class OraNFSPlugin(IFileSystemPlugin):
    """Oracle NFS File System 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`, `filesys_type`, `filesys_name`, `ss_extra_info_help`, `fs_extra_info_help`, `file_extra_info_help` and `plugin_ability`. For example:

```
plugin_name          = "Oracle NFS File System"
vendor_name          = "Oracle"
plugin_version       = "1.0.1-1"
plugin_desc          = "Oracle NFS reference implementation"
filesys_type         = IFileSystemPlugin.NetworkFileSystem
filesys_name         = "NFS"
ss_extra_info_help   = "To enable SSL to the file server use: SSL=yes"
fs_extra_info_help   = "To enable duplication elimination use: NoDup=yes"
file_extra_info_help = "To enable full space allocation use: AllocateAll=yes"
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,
                        "backing_device_type":  BACKING_DEVICE_TYPES.INVALID,
                        "access_control":       ABILITY_TYPES.INVALID,
                        "max_access_entries":   0}
```

The Oracle VM Manager display the `plugin_name` attribute so the user can identify the 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` format, for example "`1.0.1-1`". The `plugin_desc` is an open format description string for the plug-in; this string will displayed in the Oracle VM Manager at the time of Storage server configuration. 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 Server 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 by the server to determine what advanced snap and cloning features the plug-in support. 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 files can be actively used (i.e. open) when splitting them, 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`.
`require_storage_name`	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`.
`backing_device_type`	This is used to specify what type of device the File System requires. See `BACKING_DEVICE_TYPES` enumeration for all valid values the plugin can set.
`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]**

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record for the File Server to be validated.

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 file system is supplied especially if the plug-in require anything to be specified in the `extra_info` field. If possible, the plug-in should connect to the storage server to verify that the file system is ready and that the specific file system type requested is licensed etc. There is no return value for the method, if the method finds any discrepancy or is unable to verify that the storage is ready etc. it should raise an exception. The exact exception class that should be raised would 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 that this method should not be called internally by the plug-in to validate the Storage Server record in any other method calls. The plug-in will never be given an un-validated Storage Server record to any other method in the plug-in.

getCapabilities() [REQUIRED]

Parameters:

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

Return value:

New `plugin_ability dict` for the specific Storage Server.

`getCapabilities` is used to determine the capabilities that are supported by the specific Storage Server 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 Server 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 Server can create snapshots.
custom_snap_name	If the Storage Server allow for a custom snapshot name to be supplied, set this to `ABILITY_TYPES.YES`.
snap_is_sync	If the Storage Server 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 Server 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 Server allow for a custom clone name to be supplied, set this to `ABILITY_TYPES.YES`.
clone_is_sync	If the Storage Server 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 Server can resize a File Systems.
resize_is_sync	If the Storage Server will always resize a File System 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 Server can split clones.
splitclone_is_sync	If the Storage Server 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 files can be actively used (i.e. open) when splitting them, 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 Server can create a clone from an existing snapshot.

`snapclone_is_sync`	If the Storage Server 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`.
`backing_device_type`	This is used to specify what type of device the File System requires. See `BACKING_DEVICE_TYPES` enumeration for all valid values the plugin can set.
`access_control`	This should be set to `ABILITY_TYPES.YES` if the Storage Server can support access control to the File Systems.
`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.

getStorageServerInfo() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	File Server to obtain information from/about.

Return value:

Returns an updated `ss_record` with all the applicable information for the Storage Server.

`getStorageServerInfo` is used to obtain information about the Storage Server. Of particular interest would be the `storage_desc` and all the status fields in the `ss_record`. Normally right after the initial validate call, the Oracle VM Manager will call the `getStorageServerInfo` call on the Storage Server 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 server etc. Vendors are welcome to put whatever info they think would be of use to the user into this field about the physical Storage Server (firmware revisions etc. comes to mind, but this is entirely up to the plug-in provider of what to put in this string).

getFileSystemInfo() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	File Server to obtain information from/about.
`fs_record`	No	File System to obtain information about. (Default is `None`)

Return value:

Returns an updated `fs_record` with all the applicable information for the File System.

`getFileSystemInfo` is used to query the Storage Server about a specific File System. The plugin should update the `fs_record` and return it.

getFileInfo() [REQUIRED (BOTH FORMS)]

Parameters:

Name	Optional	Description

ss_record	No	File Server to obtain information from/about.
fs_record	No	File System to obtain information about.
file_record	No	File to obtain information about.
file_comp_record	Yes	File to use for data block sharing comparison. (Default is None)

Return value:

Returns an updated file_record.

getInfo is used to query a specific file, and if the file_comp_record is supplied (i.e. if file_comp_record != None), the amount of bytes shared between the two files (if any) should be set in the shared_sz field as well.

getStorageNames() [REQUIRED]

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 Server this Storage Server record will address. This is intended for plug-ins and Storage Servers that uses a concentrator or management appliance that manages multiple Storage Servers and the name is used to distinguish between them. If the plug-in or Storage Server does not support multiple Storage Servers or does not require it, the plug-in should just return an empty list ([]).

getAccessGroups() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to query for the access groups.
fs_record	Yes	Optional File System to query.

Return value:

Return the list of access groups from the Storage Server (or if an fs_record is supplied for the specific File Systems).

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

createAccessGroups() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to create the access group on.
access_grps	No	List of access groups to create on the Storage Server.

Return value:

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

createAccessGroups creates the list of access_grps passed in on the Storage Server as well as adding all the successfully created groups to the Storage Server record's list of access groups (access_grps).

removeAccessGroups() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server from which to remove (delete) the access group from.
access_grps	No	List of access groups to remove (delete) from the Storage Server.

Return value:

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

removeAccessGroups deletes the list of access_grps passed in from the Storage Server as well as removing all the successfully deleted groups from the Storage Server record's list of access groups (access_grps).

renameAccessGroup() [REQUIRED]

Parameters:

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

Return value:

Updated ss_record 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 Server as well as renaming the successfully renamed group in the Storage Server record's list of access groups (access_grps).

addToAccessGroup() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to query for the access groups.
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 NFS this would be the list of hosts the file system is exported to)

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 Server and return the updated access group.

removeFromAccessGroup() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server to query for the access groups.
`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 NFS this would be the list of hosts the file system is exported to)

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 Server and return the updated access group.

listFileSystems() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Record indicating which File Server to obtain the list from.
`include_unpresented`	Yes	If set (i.e. set to `True`) all the file systems on the File Server should be returned not just the presented ones. (Defaults to `False`)

Return value:

List of `fs_records` for all the exported file systems on the Storage Server.

This method should return all of the file systems available on the Storage Server. If the `include_unpresented` flag is set to `True` (it defaults to `False`), the method should return all the file system that exist on the Storage Server not just the ones that are exported (presented) or available for mounting by the calling server.

listMountPoints() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Record indicating which File Server to obtain the list from.
`fs_record`	Yes	File system containing the files to list.

Return value:

List of `mount_records` for all mounted file systems for this Storage Server.

This method should return the list of mount points all the file systems on the Storage Server are currently mounted on this server. If the `fs_record` is supplied only the mount points for that specific File System should returned.

list() [REQUIRED]

Parameters:

Name	Optional	Description

ss_record	No	Storage Server record indicating which File Server to obtain the list from.
fs_record	No	File system containing the files to list.
file_record	No	File record specifying the file (or directory) to list.
recursive	Yes	If set (i.e. set to True) the list will traverse all directories below the directory specified in the file_record.

Return value:

List of file_records for file(s).

This method should return the list of file(s) specified in the file_record. The file_record, at a minimum, will have the fr_type, fs_uuid, and file_path fields filled in. It is possible that the file_path is specified relative to the file system, the plug-in should handle both cases. Optionally name_pattern would contain the regular expression for filtering the file names from the list.

getStatus() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Record indicating the File Server to be queried.
fs_record	Yes	File system to retrieve the status for (Defaults to None).

Return value:

Return a string containing the status for the File Server or the File system (if an fs_record is also supplied).

The method is used to get the status of the File Server or a specific File system. This is intended to be a lightweight call to get the status than the getFileSystemInfo call. If the File server does not have a lightweight call, the plug-in can internally call the getFileSystemInfo to obtain the status.

mount() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Record indicating which File Server owns the file system to mount.
fs_record	No	File system to mount.
mount_point	No	Directory (mount point) where File System should be mounted.
share_path	Yes	Extra path on the file Storage Server relative to the export path to use for the mount (Defaults to None). This allows the use of directories in the file system that is exported to be used as mount points. For example if the export path is nfsserver:/export/vmrepos and share_path is set to poola then the full path on the mount would be nfsserver:/export/vmrepos/poola
create_mount_point	Yes	Create the mount point recursively if it does not exist (Defaults to False)
mount_options	Yes	Mount options that should be used when, mounting the File System (Defaults to None)

Return value:

Returns the `fs_record` updated with the mount fields filled in for the file system mount.

Mount the specified file system, taking in account the optionally specified mount options passed to the call.

unmount() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Record indicating which Storage Server owns the file system to un-mount.
`fs_record`	No	File system to un-mount.
`mount_point`	No	Directory (mount point) of the File System that should be un-mounted.
`destroy_mount_point`	Yes	Destroy (remove) the mount point after the file system has been un-mounted (Defaults to `False`).

Return value:

N/A

Un-mount the file system specified by the `mount_point` parameter. Note if the file system is busy, the plug-in should just raise the `FileSystemBusyEx` exception.

createFileSystem() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating on which File Server the file system will be created.
`name`	No	User friendly name for the new File System.
`backing_device`	No	File System backing device. This can be either a String with the backing device if only a single device is supported (as indicated in the plugin ability dict) or a List of backing devices.
`size`	Yes	The desired size for the new file system. A size of zero (`0`) indicates the file system should be created as large as possible. (Defaults to `0`)
`access_grp_names`	Yes	Access group names to which the file system should be exported to, if any. (Defaults to `None`)
`qos`	Yes	Desired QoS for the newly created File System (if the Storage Server does not support a QoS when creating the new File System, it can safely be ignored but the plug-in should still fill in the `qos` in the newly created `fs_record` with the value appropriate for the Storage Server.
`force`	Yes	Flag to indicate that any existing File System should be overwritten (default `False`)

Return value:

Newly created `fs_record` for the File System.

The create method should create a new file system on the File Server. The plug-in should create and return a new `fs_record` with all the appropriate info it will require in the future to locate and operate on the file system, this include the `qos` if supported by the File Server.

create()

[REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating on which File Server the file system will be created.
`fs_record`	No	File system to create the file on.
`file_name`	No	Name of the file to create.
`file_type`	No	Type of file to create, can be either `IFileSystemPlugin.FileType` or `IFileSystemPlugin.DirType`.
`size`	No	Size of the file to create.
`sparse`	Yes	Flag to indicate that a sparse (not fully allocated) file should be created (default `True`)
`force`	Yes	Flag to indicate that any existing file should be overwritten (default `False`)

Return value:

A `file_record` for the newly created snapshot.

The create method should create a new file on the File System. The plug-in should create and return the `file_record` with all the appropriate info it will require in the future to locate and operate on the file system as well as the size fields.

startPresent()

[REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Record indicating which File Server owns the file system to present.
`fs_record`	No	File system to present (export).
`access_grp_names`	No	List of access group names the File System should be presented to.

Return value:

`fs_record` with all the currently active access group names updated for the File System.

`startPresent` is called to start presenting (exporting or make visible) the file system specified in the `fs_record` to the hosts listed in the access groups as identified by the `access_grp_names` and return an updated `fs_record` with the `access_grp_names` updated.

stopPresent()

[REQUIRED]

Parameters:

Name	Optional	Description

ss_record	No	Record indicating which File Server owns the file system to stop presenting.
fs_record	No	File system to stop presenting (un-export).
access_grp_names	No	List of access group names the File System should not be presented to anymore.

Return value:

fs_record with all the currently active access group names updated for the File System.

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

resizeFileSystem() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	File Server owns the file system to resize.
fs_record	No	File system to resize.
new_size	No	New size for the file system.

Return value:

Returns the fs_record updated with the new size.

The resizeFileSystem method is used to resize the file system (larger or smaller); upon successful completion of the operation, an updated fs_record with the actual size of the file system will be returned.

resize() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	File Server owns the file system to resize.
fs_record	No	File system containing the file that is to be resized.
file_record	No	File to resize.
new_size	No	New size for the file.
sparse	Yes	Flag to indicate that the file should be extended using sparse allocation (not fully allocated) (default True)

Return value:

Returns the file_record updated with the new size.

The resize method is used to resize the file (larger or smaller); upon successful completion of the operation, an updated file_record with the actual size of the file will be returned. Note: It is the responsibility of the plug-in to verify that all conditions are met for the resize operation. For example if online resize is not supported, the plug-in must make sure the file is not currently open and/or locked and raise an Exception if it is.

destroyFileSystem() [REQUIRED]

Parameters:

Name	Optional	Description

ss_record	No	Storage Server record indicating which storage server owns the File System to be removed.
fs_record	No	File System to remove.

Return value:

N/A

This method should remove the File System from the Storage Server. Note since this method is very destructive, it should make no automatic decisions on behalf of the caller, for example if the File System is still mounted, it should just raise and exception and NOT automatically un-mount the File System to be able to remove it.

destroy() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server record indicating which storage server owns the File to be removed.
fs_record	No	File System containing the File to remove.
file_record	No	File to remove.

Return value:

N/A

This method should remove the File from the File System. Note since this method is very destructive, it should make no automatic decisions on behalf of the caller, for example if the File is still open or locked, it should just raise an exception and NOT remove it.

getFileSystemCloneLimits() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to get the limit from.
fs_record	Yes	File System to get limits for (if specified).

Return value:

Maximum number of possible shallow clones (`-1` for unlimited) supported by the File Server or File System.

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

getCloneLimits() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to get the limit from.
fs_record	No	File System containing the File.
file_record	No	File to get limits for.

Return value:

Maximum number of possible shallow clones (`-1` for unlimited) supported on the specific File.

The method should obtain the maximum number of thin (shallow) clones that can be created for a file.

isCloneable() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which storage server owns the File to check.
`fs_record`	No	File System to check.
`file_record`	No	File to check if clone-able.

Return value:

`True` if File Server and File System support shallow (thin) cloned files, otherwise it should return `False`.

Check if the File Server can create a shallow (thinly provisioned) clone of the file (identified by the `file_record`) at this specific moment in time. The method need to take into account any limits the File Systems or File Server limits.

clone() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Record indicating which Storage Server owns the File System on which the file is that should be cloned.
`fs_record`	No	File System that contains the file to be cloned.
`file_record`	No	File to clone.
`dest_file_name`	Yes	Destination for the new (clone) file. This will not be supplied if the plug-in indicates that it is unable to accept user names for clones (see `plugin_ability_dict`).

Return value:

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

This method should be implemented by the plug-in only if the Storage Server and File System support direct creation of shallow (thin provisioned) clones. If the Storage Server or File System always needs an intermediate snapshot before a clone of a file can be created, the plug-in should not implement the method. In essence, the method will create a new shallow clone for the original file specified by the `file_record` on the File System specified by the `fs_record`. The `dest_file_name` is used to name the new clone (if supported by the Storage Server and File System).

isSplittable() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server record indicating which storage server owns the File to check.

fs_record	No	File System to check.
file_record	No	File to check if split-able.

Return value:

> `True` if File Server and File System can split the file from its parent (or peers), otherwise it should return `False`.

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

splitClone() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server that owns the File System on which the clone to split resides.
fs_record	No	File System containing the file clone to split.
file_record	No	The cloned file 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 files on the Storage Server so that they do not share the same underlying storage anymore. If the Storage Server does not support deep copy clones the method should just raise the `NoSuchOperationEx` exception.

cloneFromSnap() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Record indicating which Storage Server owns the File System on which the file is that should be cloned.
fs_record	No	File System that contains the file to be cloned.
snap_file_record	No	Snapshot to use for the clone operation.
dest_file_name	Yes	Destination for the new clone of the file. This will not be supplied if the plug-in indicates that it is unable to accept user names for clones (see plugin_ability_dict).

Return value:

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

In essence, the method will create a new shallow clone for the file specified by the `snap_file_record` as it was froze at the time of the snapshot. The `dest_file_name` is used to name the new clone (if supported by the Storage Server and the File System).

getCurrentClones() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	The Storage Server to query.
fs_record	No	File System to query.
file_record	Yes	Specific file to query if supplied.

Return value:

A list of file_records for the current known clones on the Storage Server (global list) or if the file_record is supplied for this particular File.

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

getFileSystemSnapLimits() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to query.
fs_record	Yes	File System to get limit for (if specified).

Return value:

Maximum number of possible snapshots supported by the Storage Server or File System if supplied (`-1` for unlimited).

The method should obtain the maximum number of snapshots that can be created for a file. If a fs_record is supplied the limit specific to this File System should be returned (if it is different than the global), otherwise the global limit (Storage Server level) should be returned.

getSnapLimits() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to query.
fs_record	No	File System that contains the File.
file_record	No	File to get limit for.

Return value:

Maximum number of possible snapshots supported for the file (`-1` for unlimited).

The method should obtain the maximum number of snapshots that can be created for the file.

isSnapable() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server to query.
fs_record	No	File System to query.
file_record	No	Specific file to check.

Return value:

`True` if a snapshot can be created for the file otherwise it should return `False`.

Check if the Storage Server can create a snapshot for the file at this specific moment in time. This would take into account for example any limits that may be imposed on a specific file, File System globally by the Storage Server. Note, it is not required that the snapshot be at file level, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

isRestorable() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server to query.
`fs_record`	No	File System to query.
`file_record`	No	Specific file to check.
`snap_file_record`	No	Snapshot file record for the file.

Return value:

`True` if a snapshot can be created for the file otherwise it should return `False`.

Check if the Storage Server can roll back a file indicated by file_record to the snapshot indicated by snap_file_record the for the file at this specific moment in time..

createSnap() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server to create the Snapshot on.
`fs_record`	No	File System on which the file is located.
`file_record`	No	File to create the snapshot for.
`snap_name`	Yes	Optionally specify the name of the Snapshot.

Return value:

A `file_record` for the newly created snapshot.

The `createSnap` method creates a snapshot for the file specified by the `file_record`. If the plug-in supports supplying a name for snapshots (communicated via the `plugin_ability` class variable) `snap_name` would contain the desired name for the snapshot. It is not required that the snapshot be at the same level as the file, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

createMultiSnap() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server to create the Snapshot on.
`fs_record`	No	File System on which the files are located.
`file_records`	No	List of all the files to Snapshot together.
`snap_name`	Yes	Optionally specify the name for the snapshot.

Return value:

A `file_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 file on the Storage Server, it will take a single snapshot of multiple files at the same time. If the Storage Server or File System does not support taking a snapshot of multiple files 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_name` would contain the desired name for the snapshot. It is not required that the snapshot be at the same level as the file, it is completely acceptable if the snapshot is at higher level, for example at the volume group level.

snapRestore() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server containing the Snapshot.
`fs_record`	No	File system containing the Snapshot.
`file_record`	No	File to rollback.
`snap_file_record`	No	Snapshot to restore.

Return value:

N/A

`snapRestore` roll back the file indicated by the `file_record` to the snapshot indicated by `snap_file_record`.
NOTE: Extremely important, if a Storage Server cannot roll back just the file 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 files on the Storage Server.

snapRemove() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server containing the Snapshot.
`fs_record`	No	File system containing the Snapshot.
`snap_file_record`	No	Snapshot to remove.

Return value:

N/A

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

getCurrentSnaps() [REQUIRED]

Parameters:

Name	Optional	Description
`ss_record`	No	Storage Server containing the Snapshot.
`fs_record`	No	File system containing the Snapshot.

file_record	Yes	File to get the list of Snapshots for.

Return value:

List of file_records for the current known snapshots on the Storage Server (global list) or if the file_record is supplied for this particular file.

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

getQoSList() [REQUIRED]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server.
fs_record	No	File System.

Return value:

List of qos_vals dicts with all the known Quality-of-Service values for the File System.

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]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server.
fs_record	No	File System which QoS setting should be updated.
qos	No	New QoS value for the File System.

Return value:

N/A

Update the Quality-of-Service value for the specified File System. If the Storage Server does not allow changing the QoS value while the File System is online the plug-in should raise the FileSystemBusyEx exception. If the Storage Server does not support changing the QoS of the File System the plug-in should raise the NoSuchOperationEx exception.

getFileSystemBackingDevices() [REQUIRED IF ABILITY IS SET]

Parameters:

Name	Optional	Description
ss_record	No	Storage Server.
fs_record	No	File System for which the backing devices should be returned.

Return value:

List of backing_device dicts with all the available (valid) backing devices for the File System.

`getFileSystemBackingDevices` is used only if the plugin ability `backing_device_type` is set to `BACKING_DEVICE_TYPES.PLUGIN_SINGLE` or `BACKING_DEVICE_TYPES.PLUGIN_MULTI`. The user would select one (or more in the case of `BACKING_DEVICE_TYPES.PLUGIN_MULTI`) of the `dicts`. The "`value`" key would be passed to the `createFileSystem()` call to allow the plugin to create the file system on the selected backing devices. The value is completely opaque to the Oracle VM Manager; it will never be displayed, only stored and passed back to the plugin.

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.