Storage Management Technical Specification, Part 4 File Systems

Version 1.3.0, Rev 6

SNIA Technical Position

21 April, 2009

Revision History

Revison 1

Date

4 January, 2007

SCRs Incorporated and other changes

Comments

Format and variables updated for new revision, editorial comments displayed.

Revison 2

Date

14 April 2007

SCRs Incorporated and other changes

Filesystem Performance Profile Subprofile

- Updated XML and frame text (SMIS-130-Draft-SCR00002) (4-0-0)

File Server Manipulation Profile Subprofile

- Updated the diagrams and changed the CIM Elements section (SMIS-130-Draft-SCR00009) (4-0-0)

Filesystem Copy (Abstract) subprofile

- Added this new profile (FSM TWG SCR 00055) (3-0-0)

Filesystem Copy Configuration (Abstract) subprofile

- Added this new profile (FSM TWG SCR 00055) (3-0-0)

Filesystem Checkpoint Configuration subprofile

- Added this new profile (FSM TWG SCR 00055) (3-0-0)

Filesystem Local Mirror Configuration subprofile

- Added this new profile (FSM TWG SCR 00055) (3-0-0)

NAS Head Profile (FSM-TWG-SCR00056) (4-0-0)

- Changed the version number of the profile to be 1.3.0
- Added Indication Events subclauses of the Implementation subclause (marked as DRAFT)
- Added Bellwether Indications subclauses of the Implementation subclause (marked as DRAFT)
- Added a Standard Messages subclause in the Health & Fault Management subclause
- Changed profile version numbers to 1.3.0 in the “Supported Subprofiles and Packages” subclause
- Made changes for Comments on 1.2.0 NAS Head

File Export Manipulation profile

- Based on SMIS-120-Errata-SCR00039)

Filesystem Manipulation profile

- Based on SMIS-120-Errata-SCR00038)

Filesystem profile

- Based on SMIS-120-Errata-SCR00040)

Comments

Only minor editorial work for this revision.

Revison 3

Date

19 June 2007

SCRs Incorporated and other changes

File Export Profile

- Incorporate errata from 1.2.0 (SMIS-120-Errata-SCR00059) (8-1-0)

File Storage Profile

- Incorporate errata from 1.2.0 (SMIS-120-Errata-SCR00058) (9-0-0)

Filesystem Performance Profile Subprofile

- Updated and **Promote to Experimental** (SMIS-130-Draft-SCR00002) (5-0-0)

Filesystem Quotas Profile

- Incorporate errata from 1.2.0 (SMIS-120-Errata-SCR00063) (7-0-2)

File Server Manipulation Profile Subprofile

- Updated and **Promote to Experimental** (SMIS-130-Draft-SCR00009) (2-0-1)

Host Filesystem Profile

- **Deleted Host Filesystem profile** (FSM-TWG-SCR00050) (3-0-0)

NAS Head Profile (FSM-TWG-SCR00056) (4-0-0)

- Changed SNIA_LogicalFile in the Instance Diagram to just LogicalFile

- Fixed the Instance diagram for the optional Initiator Ports profile

- Fixed the File Storage diagram to delete the reference to the Initiator Ports profile

- Deleted the Recipe Conventions bullets from the beginning of the Client Considerations and Recipes section

- Deleted the Recipe Subroutines and replace with "Not defined in this version of the specification"

Self-Contained NAS Profile (FSM-TWG-SCR00057) (4-0-0)

- Changed the version number of the profile to be 1.3.0

- Added Indication Events subclauses of the Implementation subclause (marked as DRAFT)

- Added Bellwether Indications subclauses of the Implementation subclause (marked as DRAFT)

- Added a Standard Messages subclause in the Health & Fault Management subclause

- Changed profile version numbers to 1.3.0 in the “Supported Subprofiles and Packages” subclause

- Changed SNIA_LogicalFile in the Instance Diagram to just LogicalFile

- Deleted the Recipe Conventions bullets from the beginning of the Client Considerations and Recipes section

- Deleted the Recipe Subroutines and replace with "Not defined in this version of the specification"

- Added Conditions for TCPProtocolEndpoint, IPProtocolEndpoint & LANEndpoint and applied them to the appropriate associations

- Made BindsToLANEndpoint conditional on implementation of LANEndpoint

- Made BindsTo conditional on implementation of TCPProtocolEndpoint or IPProtocolEndpoint

- Made DeviceSAPImplementation for the LANEndpoint conditional on implementation of LANEndpoint

- Changed the Dependent ProtocolEndpoint in a BindsToLANEndpoint from TCPProtocolEndpoint to IPProtocolEndpoint

- Replaced the HostedAccessPoint for TCP, IP and LAN endpoints with three HostedAccessPoints and made them conditional

- Separated SystemDevice for LogicalDisks from LogicalDisks to "tighten" the spec to show two LogicalDevices supported by the Self-Contained NAS

- Global changed SC NAS to Self-contained NAS

Volumes on Files Profile

- **Deleted Volumes on Files profile** (FSM-TWG-SCR00049) (2-0-2)

Pools on Filesystems Profile

- **Deleted Pools on Filesystems profile** (FSM-TWG-SCR00049) (2-0-2)

Comments

Editorial notes displayed.

Responses to INCITS editor queries re SMI-S 1.1.0 incorporated as applicable.

Typographical Conventions revised in all books: Revised explanation of Experimental text (per SMIS-120-Errata-SCR00061 - Typographical Conventions), added explanations of Draft and Editorial text.

Revision 4

Date

20 July 2007

SCRs Incorporated and other changes

Filesystem Quotas Profile (FSM-TWG-SCR00058) (5-0-0)

- Change references to 1.2.0 to 1.3.0.

File Server Manipulation Profile Subprofile (FSM-TWG-SCR00046) (5-0-0)

- Changed “Draft” to “Experimental” in frame file.

- Removed duplicate class definitions

- Removed classes that are defined in other profiles

- Added Conditions “CIFSSettings”, “NFSSettings”, “DNSSettings”, “NISSettings”, and “VLANSettings”. Modified conditional associations to use these.

- Various other changes

- Modified the two Visio diagrams by removing unnecessary classes that are defined in other profiles

NAS Head Profile (FSM-TWG-SCR00056) (5-0-0)

- Deleted the DRAFT sections on Indications and Standard Messages

- Deleted the references to those sections from the CIM Elements table

Self-Contained NAS Profile (FSM-TWG-SCR00057) (5-0-0)

- Deleted the DRAFT sections on Indications and Standard Messages

Comments

Editorial notes displayed, but the DRAFT material is not.

Revision 5

Date

14 November 2007

SCRs Incorporated and other changes

Clause 10: Filesystem Performance Profile (FSM-TWG-SCR00059)

- Made SNIA_FileSystemStatisticsManifestCollection conditional (on Capabilities property)

Clause 6: File Server Manipulation Subprofile (FSM-TWG-SCR00060)

- Updated a couple of figures and removed one of the ElementCapabilities tables

Comments

Editorial notes and DRAFT material are not displayed.

Revision 6

Date

14 January 2009

SCRs Incorporated and other changes

References to *Storage Management Technical Specification, Part 7 Information Lifecycle Management*, deleted.

Removed test CARDINALITY (SMIS-130-Errata-SCR00001)

Filesystem Performance Profile changes (SMIS-130-Errata-SCR00007)

Invalid version numbers in supported profiles tables replaced with valid numbers (SMIS-130-Errata-SCR00017)

Updated NAS Head Profile (SMIS-130-Errata-SCR0042)

Updated SC NAS Profile (SMIS-130-Errata-SCR0043)

Updated Filesystem Profile (SMIS-130-Errata-SCR0044)

Updated File Export Profile (SMIS-130-Errata-SCR0045)

Updated File Export Manipulation Profile (SMIS-130-Errata-SCR00046)

Updated Filesystem Manipulation Profile (SMIS-130-Errata-SCR00047)

Comments

Editorial notes and DRAFT material are not displayed.

Suggestion for changes or modifications to this document should be sent to the SNIA Storage Management Initiative Technical Steering Group (SMI-TSG) at http://www.snia.org/feedback/.

INTENDED AUDIENCE

This document is intended for use by individuals and companies engaged in developing, deploying, and promoting interoperable multi-vendor SANs through the SNIA organization.

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Portions of the CIM Schema are used in this document with the permission of the Distributed Management Task Force (DMTF). The CIM classes that are documented have been developed and reviewed by both the Storage Networking Industry Association (SNIA) and DMTF Technical Working Groups. However, the schema is still in development and review in the DMTF Working Groups and Technical Committee, and subject to change.

CHANGES TO THE SPECIFICATION

Each publication of this specification is uniquely identified by a three-level identifier, comprised of a version number, a release number and an update number. The current identifier for this specification is version 1.2.0. Future publications of this specification are subject to specific constraints on the scope of change that is permissible from one publication to the next and the degree of interoperability and backward compatibility that should be assumed between products designed to different publications of this standard. The SNIA has defined three levels of change to a specification:

- Major Revision: A major revision of the specification represents a substantial change to the underlying scope or architecture of the SMI-S API. A major revision results in an increase in the version number of the version identifier (e.g., from version 1.x.x to version 2.x x). There is no assurance of interoperability or backward compatibility between releases with different version numbers.
- Minor Revision: A minor revision of the specification represents a technical change to existing content or an adjustment to the scope of the SMI-S API. A minor revision results in an increase in the release number of the specification's identifier (e.g., from x.1.x to x.2.x). Minor revisions with the same version number preserve interoperability and backward compatibility.
- Update: An update to the specification is limited to minor corrections or clarifications of existing specification content. An update will result in an increase in the third component of the release identifier (e.g., from x.x.1 to x.x.2). Updates with the same version and minor release levels preserve interoperability and backward compatibility.

TYPOGRAPHICAL CONVENTIONS

This specification has been structured to convey both the formal requirements and assumptions of the SMI-S API and its emerging implementation and deployment lifecycle. Over time, the intent is that all content in the specification will represent a mature and stable design, be verified by extensive implementation experience, assure consistent support for backward compatibility, and rely solely on content material that has reached a similar level of maturity. Unless explicitly labeled with one of the subordinate maturity levels defined for this specification, content is assumed to satisfy these requirements and is referred to as "Finalized". Since much of the evolving specification

content in any given release will not have matured to that level, this specification defines three subordinate levels of implementation maturity that identify important aspects of the content's increasing maturity and stability. Each subordinate maturity level is defined by its level of implementation experience, its stability and its reliance on other

emerging standards. Each subordinate maturity level is identified by a unique typographical tagging convention that clearly distinguishes content at one maturity model from content at another level.

Experimental Maturity Level

No material is included in this specification unless its initial architecture has been completed and reviewed. Some content included in this specification has complete and reviewed design, but lacks implementation experience and the maturity gained through implementation experience. This content is included in order to gain wider review and to gain implementation experience. This material is referred to as “Experimental”. It is presented here as an aid to implementers who are interested in likely future developments within the SMI specification. The contents of an Experimental profile may change as implementation experience is gained. There is a high likelihood that the changed content will be included in an upcoming revision of the specification. Experimental material can advance to a higher maturity level as soon as implementations are available. Figure 1 is a sample of the typographical convention for Experimental content.

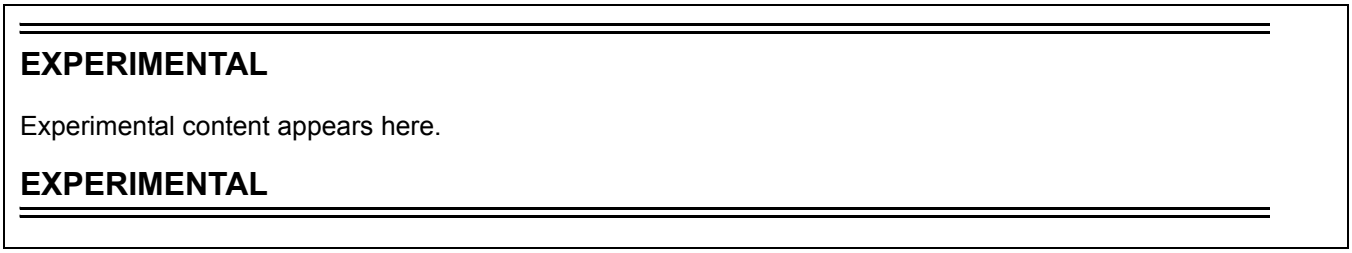

Figure 1 - Experimental Maturity Level Tag

Implemented Maturity Level

Profiles for which initial implementations have been completed are classified as “Implemented”. This indicates that at least two different vendors have implemented the profile, including at least one provider implementation. At this maturity level, the underlying architecture and modeling are stable, and changes in future revisions will be limited to the correction of deficiencies identified through additional implementation experience. Should the material become obsolete in the future, it must be deprecated in a minor revision of the specification prior to its removal from subsequent releases. Figure 2 is a sample of the typographical convention for Implemented content.

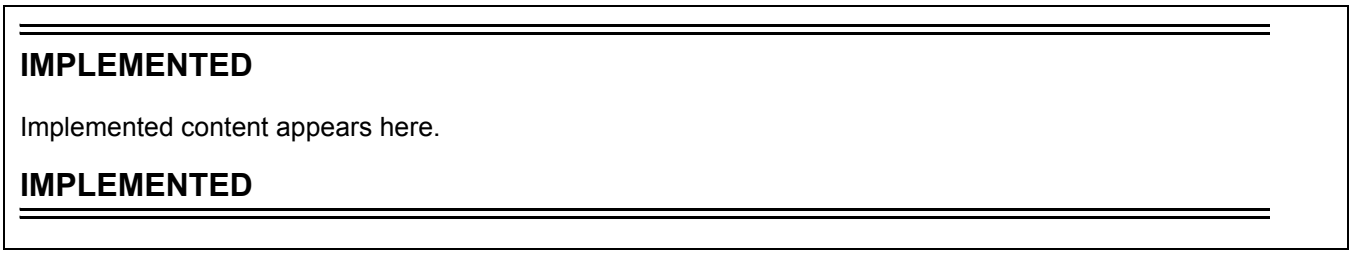

Figure 2 - Implemented Maturity Level Tag

Stable Maturity Level

Once content at the Implemented maturity level has garnered additional implementation experience, it can be tagged at the Stable maturity level. Material at this maturity level has been implemented by three different vendors, including both a provider and a client. Should material that has reached this maturity level become obsolete, it may only be deprecated as part of a minor revision to the specification. Material at this maturity level that has been deprecated may only be removed from the specification as part of a major revision. A profile that has reached this maturity level is guaranteed to preserve backward compatibility from one minor specification revision to the next. As a result, Profiles at or above the Stable maturity level shall not rely on any content that is Experimental. Figure 3 is a sample of the typographical convention for Implemented content.

STABLE

Stable content appears here.

STABLE

Figure 3 - Stable Maturity Level Tag

Finalized Maturity Level

Content that has reached the highest maturity level is referred to as “Finalized.” In addition to satisfying the requirements for the Stable maturity level, content at the Finalized maturity level must solely depend upon or refine material that has also reached the Finalized level. If specification content depends upon material that is not under the control of the SNIA, and therefore not subject to its maturity level definitions, then the external content is evaluated by the SNIA to assure that it has achieved a comparable level of completion, stability, and implementation experience. Should material that has reached this maturity level become obsolete, it may only be deprecated as part of a major revision to the specification. A profile that has reached this maturity level is guaranteed to preserve backward compatibility from one minor specification revision to the next. Over time, it is hoped that all specification content will attain this maturity level. Accordingly, there is no special typographical convention, as there is with the other, subordinate maturity levels. Unless content in the specification is marked with one of the typographical conventions defined for the subordinate maturity levels, it should be assumed to have reached the Finalized maturity level.

Deprecated Material

Non-Experimental material can be deprecated in a subsequent revision of the specification. Sections identified as “Deprecated” contain material that is obsolete and not recommended for use in new development efforts. Existing and new implementations may still use this material, but shall move to the newer approach as soon as possible. The maturity level of the material being deprecated determines how long it will continue to appear in the specification. Implemented content shall be retained at least until the next revision of the specialization, while Stable and Finalized material shall be retained until the next major revision of the specification. Providers shall implement the deprecated elements as long as it appears in the specification in order to achieve backward compatibility. Clients may rely on deprecated elements, but are encouraged to use non-deprecated alternatives when possible.

Deprecated sections are documented with a reference to the last published version to include the deprecated section as normative material and to the section in the current specification with the replacement. Figure 4 contains a sample of the typographical convention for deprecated content.

DEPRECATED

Content that has been deprecated appears here.

DEPRECATED

Figure 4 - Deprecated Tag

USAGE

Contents

List of Tables

List of Figures

Foreword

The Filesystems Part of the Storage Management Technical Specifications contains Profiles and other clauses for management of devices and programs that support filesystems. A filesystem is a specific formatting of storage for storing and accessing files on external storage. This part describes how filesystems are created, modified and deleted, as well as how they can be found and reported. This part also describe modeling for how filesystems are exported for access from remote systems. The filesystem profiles use information from other parts of the Storage Management Technical Specifications. Specifically, they reference profiles in the Common Profiles and the Block Devices parts of the specification. This part describes how these profiles are used in filesystem profiles.

Parts of this Standard

This standard is subdivided in the following parts:

- *Storage Management Technical Specification, Overview, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 1 Common Architecture, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 4 File Systems, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 5 Fabric, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 6 Host Elements, 1.3.0 Rev 6*
- *Storage Management Technical Specification, Part 7 Media Libraries, 1.3.0 Rev 6*

SNIA Web Site

Current SNIA practice is to make updates and other information available through their web site at http://www.snia.org

SNIA Address

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent via the SNIA Feedback Portal at http://www.snia.org/feedback/ or by mail to the Storage Networking Industry Association, 500 Sansome Street, Suite #504, San Francisco, CA 94111, U.S.A.

Acknowledgments

The SNIA SMI Technical Steering Group, which developed and reviewed this standard, would like to recognize the significant contributions made by the following members:

Organization Represented	*Name of Representative*
Brocade	John Crandall
Dell	Vance Corn
EMC	Mike Thompson
Hewlett Packard	Alex Lenart
	Steve Peters
Hitachi Data Systems	Steve Quinn
Individual member	Tom West
IBM	Krishna Harathi
	Mike Walker
	Martine Wedlake
Olocity	Scott Baker
Pillar	Gary Steffens
Symantec	Steve Hand
	Paul von Behren

Clause 1: Scope

The Filesystems Part of the Storage Management Technical Specification defines management profiles for Autonomous (top level) profiles for programs and devices whose central function is providing support and access to file data. In addition, it provides documentation of component profiles (or subprofiles) that deal with filesystems and management interface functions that may be used by other autonomous profiles not included in this part of the specification.

There is an informative annex that describes how storage is mapped from block storage to file shares exported by the file system and the mechanisms involved in that establishing those mappings. This annex is recommended for getting an overview of how the filesystem models work.

This version of the Filesystems part of the Storage Management Technical Specification includes two autonomous profiles:

- The NAS Head Profile

  This profile defines the model and functions of a NAS device that exports file shares to remote users and gets its storage from a SAN (array devices attached to the NAS Head device).

- The Self-Contained NAS Profile

  This profile defines the model and functions of a NAS device that exports file shares to remote users, but gets its storage from disk drives that are internal to the NAS device (instead of externally attached arrays).

In addition to these autonomous profiles, this part of the specification defines a number of component profiles, which are used by the autonomous NAS profiles and might also be used by other autonomous profiles that feature filesystem elements and services. The component profiles (subprofiles) defined in this version of the specification include:

- The File Export (component) Profile

  This component profile defines the elements used to model the exporting of filesystems or directories for any autonomous profile that exports file data to remote systems.

- The File Export Manipulation (component) Profile

  This component profile defines the elements used to model the services for creating, modifying and deleting the file shares (the representation of exported filesystems or directories) for any autonomous profile that provides manipulation of exported filesystems or directories.

- The File Storage (component) Profile

  This component profile defines the elements used to model the storage of filesystems on logical disks. This profile does not have services for maintaining the mapping of filesystems to logical disks. These services are addressed in the Filesystem Manipulation Profile.

- The Filesystem (component) Profile

  This component profile defines the elements used to model filesystems and its related elements, such as logical files, directories and information on how the filesystem is addressed when mounted to a specific system. The services for defining and maintaining the information in the Filesystem Profile are contained in the Filesystem Manipulation Profile.

- The Filesystem Manipulation (component) Profile

  This component profile defines the elements used to model the services for creating, modifying and deleting filesystems and their related elements.

- The Filesystem Quotas (component) Profile

  This component profile defines the elements used to model the elements associated with creating, maintaining and reporting on quotas on various filesystem elements.

Clause 2: Normative References

2.1 General

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.2 Approved references

ISO/IEC 14776-452, SCSI Primary Commands - 2 (SPC-2) [ANSI INCITS.351-2001]

2.3 References under development

Storage Management Technical Specification, Part 1 Common Architecture, 1.3.0 Rev 6

Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6

Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6

ISO/IEC 14776-452, SCSI Primary Commands - 3 (SPC-3) [ANSI INCITS.351-2005]

2.4 Other references

DMTF DSP0214:2004 CIM Operations over HTTP

Clause 3: Terms and definitions

3.1 General

For the purposes of this document, the terms and definitions given in *Storage Management Technical Specification, Part 1 Common Architecture, 1.3.0 Rev 6* and the following apply.

3.2 Definitions

3.2.1 CIFS

Common Internet File System.

3.2.2 Directory

A subtree within a filesystem. A directory may contain files or other directories.

3.2.3 File

A logical file in a filesystem.

3.2.4 File Server

A system configuration which supports the exporting of files and files systems. A file server may be a virtual system element.

3.2.5 File Share

Sharing protocols applied to a directory. A directory is exported to remote users through a file share.

3.2.6 Filesystem

A filesystem is the way in which files are named and where they are placed logically for storage and retrieval.

3.2.7 FS Quota

A quota (hard or soft limit) placed on filesystem resource usage.

3.2.8 Logical Disk

This refers to block storage on which filesystems are built. A logical disk would be formatted for a particular filesystem.

3.2.9 NAS

Network Attached Storage. In the context of this specification this refers to devices that serve files to a network.

3.2.10 NAS Head

A NAS device that gets its physical storage from one or more arrays that are externally attached to the NAS device.

3.2.11 NFS

Network File System.

3.2.12 Self-Contained NAS

A NAS device that has its own internal (to the NAS device) storage.

3.2.13 Quota

A hard or soft limit defined for users, user groups or resource collections on the amount of resources that may be consumed.

STABLE

Clause 4: File Export Profile

4.1 Description

4.1.1 Synopsis

Profile Name: File Export

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: SNIA_FileShare

Scoping Class: CIM_ComputerSystem

4.1.2 Overview

The File Export Profile is a subprofile for autonomous profiles that support exporting of filesystems. Specifically, in this release of SMI-S, this includes the NAS Head and Self-Contained NAS Profiles. In some of these autonomous profiles the File Export is required. In others it may not be. See the parent profile to see if this profile is required or not.

EXPERIMENTAL

NOTE: The ExportedFileShareSetting is defined as SNIA_ classes. While this is defined to hold new properties, the CIM version of this class will be supported for backward compatibility.

EXPERIMENTAL

4.1.3 Implementation

Figure 5 illustrates the classes mandatory for modeling the export of File Shares for the filesystem profiles. This profile is supported by the Self-contained NAS and the NAS Head Profiles. Figure 5 shows the ComputerSystem that hosts the LocalFileSystem (“Filesystem Host”) as different from the ComputerSystem hosting the FileShare (“File server”). While they may be different ComputerSystems, they may also be the same ComputerSystem instance.

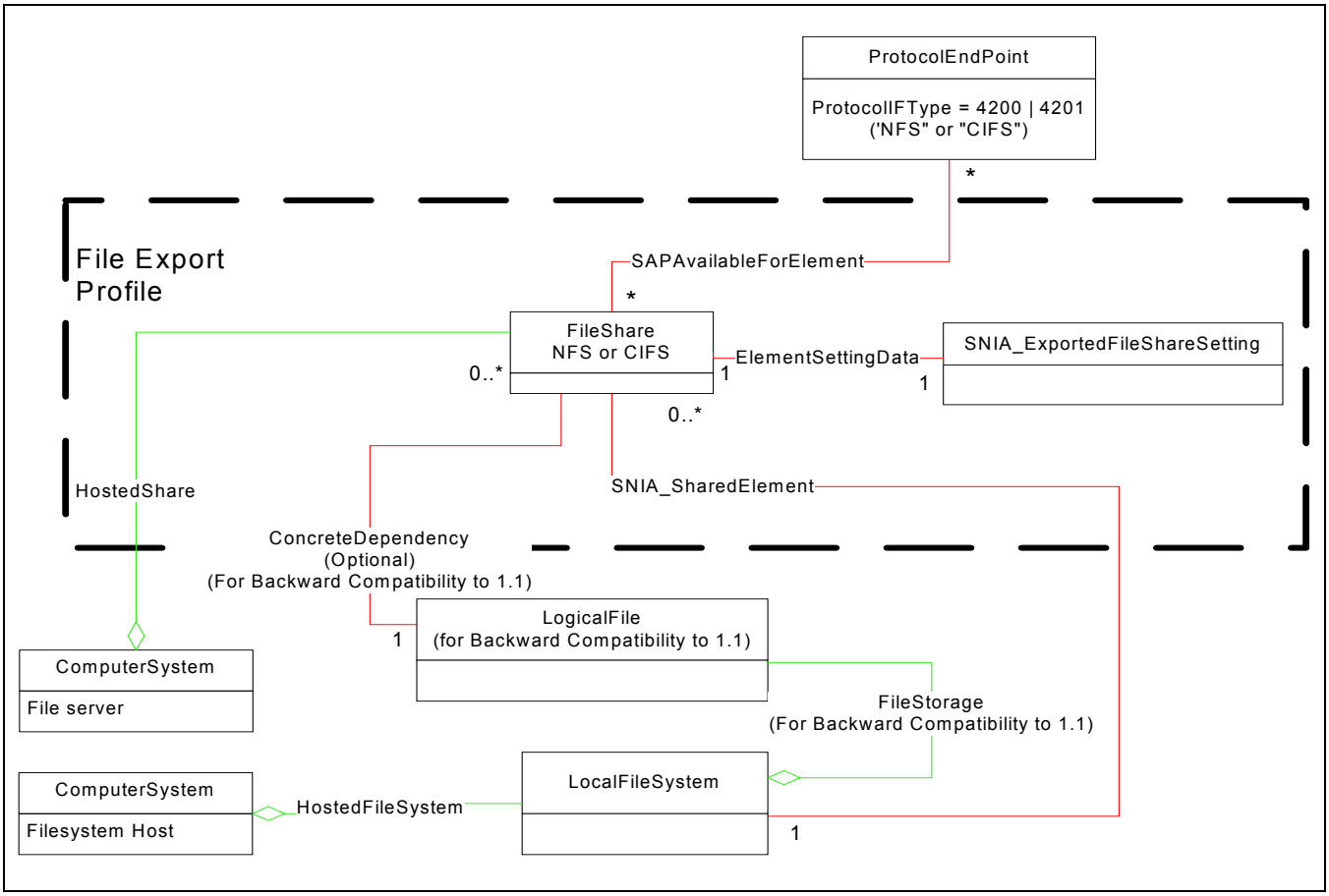

Figure 5 - File Export Instance

The referencing profile shall model any File Shares that have been exported to the network. A File Share shall be represented as a FileShare instance with associations to the ComputerSystem that hosts the share (via HostedShare), to the ExportedFileShareSetting (via ElementSettingData) and to the ProtocolEndpoint (via SAPAvailableForElement) through which the Share can be accessed.

EXPERIMENTAL

The FileShare also has a SharedElement association to the LocalFileSystem on which the share is based.

EXPERIMENTAL

In addition, there may also be an association between the FileShare and the LogicalFile that the share represents (via ConcreteDependency). This is provided for backward compatibility with SMI-S 1.1.0.

4.1.3.1 Associations to FileShare

The SAPAvailableForElement is a many to many association. That is, multiple FileShares may be exported through the same ProtocolEndpoint and multiple ProtocolEndpoints may support the same FileShare.

The SharedElement association between the FileShare and a LocalFileSystem is many to one association. Zero or more FileShares may be associated to one LocalFileSystem. But each FileShare shall only reference one LocalFileSystem.

The ConcreteDependency association between the FileShare and the LogicalFile is a many to one association. Zero or more FileShares may be associated to one LogicalFile. But each FileShare shall only reference one LogicalFile.

The ElementSettingData association between the FileShare and the ExportedFileShareSetting is a one to one association. That is, a FileShare shall have an ExportedFileShareSetting and that ExportedFileShareSetting shall be associated to exactly one FileShare.

4.2 Health and Fault Management Consideration

The File Export Profile supports state information (e.g., OperationalStatus) on the following element of the model:

- FileShares that are exported (See section 4.2.1)

4.2.1 OperationalStatus for FileShares

Table 1 - FileShare OperationalStatus

OperationalStatus	Description
OK	FileShare is online
Error	FileShare has a failure. This could be due to a Filesystem failure.
Stopped	FileShare is disabled
Unknown	

4.3 Cascading Considerations

None

4.4 Supported Profiles, Subprofiles, and Packages

Table 3 describes the supported profiles for File Export.

Table 2 - Supported Profiles for File Export

Registered Profile Names	Mandatory	Version
Indication	Yes	1.3.0

4.5 Methods of the Profile

4.5.1 Extrinsic Methods of the Profile

None

4.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

4.6 Client Considerations and Recipes

4.6.1 List Existing FileShares on the system

A client shall be able to find FileShares attached to a system (e.g., a file server) by doing an association traversal from the ComputerSystem that represents the system using the HostedShare association.

4.7 Registered Name and Version

File Export version 1.3.0

4.8 CIM Elements

Table 3 describes the CIM elements for File Export.

Table 3 - CIM Elements for File Export

Element Name	Requirement	Description
4.8.1 CIM_ConcreteDependency	Optional	Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based. This is provided for backward compatibility.
4.8.2 CIM_ElementSettingData (FileShare)	Mandatory	Associates a FileShare and ExportedFileShareSetting elements.
4.8.3 CIM_ExportedFileShareSetting (Setting)	Mandatory	The configuration settings for an Exported FileShare that is a setting for a FileShare available for exporting.
4.8.4 CIM_FileShare (Exported File Share)	Mandatory	Represents the sharing characteristics of a particular file element.
4.8.5 CIM_HostedShare	Mandatory	Represents that a shared element is hosted by a Computer System.

Table 3 - CIM Elements for File Export

Element Name	Requirement	Description
4.8.6 CIM_SAPAvailableForElement	Mandatory	Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
4.8.7 CIM_SharedElement	Mandatory	Associates a FileShare to the LocalFileSystem on which it is based.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.CIM_FileShare::OperationalStatus <> PreviousInstance.CIM_FileShare::OperationalStatus	Optional	CQL -Change of Status of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

4.8.1 CIM_ConcreteDependency

Created By: External
Modified By: Static
Deleted By: External
Requirement: Optional

Table 4 describes class CIM_ConcreteDependency.

Table 4 - SMI Referenced Properties/Methods for CIM_ConcreteDependency

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The LogicalFile that is being shared.
Dependent		Mandatory	The Share that represents the LogicalFile being shared.

4.8.2 CIM_ElementSettingData (FileShare)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 5 describes class CIM_ElementSettingData (FileShare).

Table 5 - SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare)

Properties	Flags	Requirement	Description & Notes
IsDefault	N	Optional	Not Specified in this version of the Profile
IsCurrent	N	Optional	Not Specified in this version of the Profile
IsNext	N	Optional	Not Specified in this version of the Profile
IsMinimum	N	Optional	Not Specified in this version of the Profile
IsMaximum	N	Optional	Not Specified in this version of the Profile
ManagedElement		Mandatory	The FileShare.
SettingData		Mandatory	The settings define on creation of the FileShare.

4.8.3 CIM_ExportedFileShareSetting (Setting)

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 6 describes class CIM_ExportedFileShareSetting (Setting).

Table 6 - SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Setting)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	A unique ID for the setting.
ElementName		Mandatory	A user-friendly name for the Setting.
FileSharingProtocol		Mandatory	The file sharing protocol supported by this share. NFS (2) and CIFS (3) are the supported values.
ProtocolVersions		Mandatory	An array of the versions of the supported file sharing protocol. A share may support multiple versions of the same protocol.
InitialEnabledState	N	Optional	Valid values are '1|2|3|7|8|9' for ('Other' | 'Enabled' | 'Disabled' | 'In Test' | 'Deferred' | 'Quiesce')
OtherEnabledState	N	Optional	This should be filled in if the InitialEnabledState is '1'.
DefaultUserIdSupported	N	Optional	Valid values are '2|3|4' for ('No Default User Id' | 'System-Specified Default User Id' | 'Share-Specified Default User Id').
RootAccess	N	Optional	Valid values are '2|3' for ('No Root Access' | 'Allow Root Access').

Table 6 - SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Setting)

Properties	Flags	Requirement	Description & Notes
AccessPoints	N	Optional	Valid values are '2|3|4|5' for ('None' | 'Service Default' | 'All' | 'Named Points').
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
DefaultReadWrite	N	Optional	Not Specified in this version of the Profile
DefaultExecute	N	Optional	Not Specified in this version of the Profile
ExecuteSupport	N	Optional	Not Specified in this version of the Profile
WritePolicy	N	Optional	Not Specified in this version of the Profile

4.8.4 CIM_FileShare (Exported File Share)

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 7 describes class CIM_FileShare (Exported File Share).

Table 7 - SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	A unique id for the FileShare element.
ElementName		Mandatory	This shall be a user friendly name for the FileShare.
Name		Mandatory	This shall be an opaque string that uniquely identifies the path to the directory or file.
SharingDirectory		Mandatory	Indicates if the shared element is a file or a directory. This is useful when importing but less so when exporting.
OperationalStatus		Mandatory	The OperationalStatus of the FileShare as defined in section 4.2.1
Description	N	Optional	This a comment describing the file share.
Caption	N	Optional	Not Specified in this version of the Profile
InstallDate	N	Optional	Not Specified in this version of the Profile
StatusDescriptions	N	Optional	Not Specified in this version of the Profile
HealthState	N	Optional	Not Specified in this version of the Profile
EnabledState	N	Optional	Not Specified in this version of the Profile
OtherEnabledState	N	Optional	Not Specified in this version of the Profile

Table 7 - SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share)

Properties	Flags	Requirement	Description & Notes
RequestedState	N	Optional	Not Specified in this version of the Profile
EnabledDefault	N	Optional	Not Specified in this version of the Profile
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile
RequestStateChange()		Optional	Not Specified in this version of the Profile

4.8.5 CIM_HostedShare

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 8 describes class CIM_HostedShare.

Table 8 - SMI Referenced Properties/Methods for CIM_HostedShare

Properties	Flags	Requirement	Description & Notes
RemoteShareWWN	N	Optional	Not Specified in this version of the Profile
Dependent		Mandatory	The Share that is hosted by a Computer System
Antecedent		Mandatory	The Computer System that hosts the FileShare.

4.8.6 CIM_SAPAvailableForElement

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 9 describes class CIM_SAPAvailableForElement.

Table 9 - SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The element that is made available through a SAP. In the File Export subprofile, these are FileShares configured for either export.
AvailableSAP		Mandatory	The Service Access Point that is available to this FileShare.

4.8.7 CIM_SharedElement

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 10 describes class CIM_SharedElement.

Table 10 - SMI Referenced Properties/Methods for CIM_SharedElement

Properties	Flags	Requirement	Description & Notes
SystemElement		Mandatory	The LocalFileSystem that is exporting some contained file or directory as a FileShare.
SameElement		Mandatory	The FileShare that exposes a contained file or directory of the LocalFileSystem as an exported object.

STABLE

EXPERIMENTAL

Clause 5: File Export Manipulation Subprofile

5.1 Description

5.1.1 Synopsis

Profile Name: File Export Manipulation

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: FileExportService

Scoping Class: File server ComputerSystem element (with Dedicated property containing ”16”)

5.1.2 Overview

The File Export Manipulation Subprofile is a subprofile of autonomous profiles that support filesystems. It makes use of elements of the Filesystem subprofiles and supports creation, modification and deletion of FileShares that are exported by the File Export subprofile. A number of other profiles and subprofiles also make use of elements of the Filesystem subprofile and will be referred to in this specification as “filesystem related profiles” -- these include but are not limited to the Filesystem subprofile, the Filesystem Manipulation subprofile, the File Export subprofile, the NAS Head profile, the Self-Contained NAS profile, and so on.

In this release of SMI-S, the autonomous profiles that use the File Export Manipulation Subprofile are the NAS Head and Self-Contained NAS profiles.

Annex A:, "(Informative) State Transitions from Storage to File Shares" describes the states that a storage element, initially an unused LogicalDisk, goes through before it can be exported as a CIFS or NFS file share. The Filesystem Manipulation subprofile provides the methods to create the filesystem as a LocalFileSystem and make it locally accessible at a file server ComputerSystem (associated to the file server ComputerSystem via the LocalAccessAvailable association). This profile (the File Export Manipulation Profile) provides the methods to "Export a file share" from the file server that allows the file server to share its contents with remote operational users. Sharing the contents of a LocalFileSystem can be from the root directory or some contained internal directory, or some contained internal file. When a directory (root or otherwise) is shared, all files and sub-directories of that directory are also automatically shared (recursively). The semantics of sharing are ultimately controlled by the Authorization profiles and by the filesystem implementation, so sharing cannot violate the access rules specified internally to the filesystem. In addition to specifying the object (file or directory) to be shared, the filesystem implementation shall specify the protocol to use (CIFS, NFS, or other protocol) for sharing.

SMI-S uses a FileShare element to represent the externally accessible file share. A SharedElement association will exist between the FileShare and the LocalFileSystem. The FileShare.Name property indicates the shared object (it is the filesystem-specific path to the contained file or directory that is being shared). The format of Name is specific to the filesystem type indicated by the associated FileSystemSetting.ActualFileSystemType property; the LocalFileSystem.PathnameSeparatorString property indicates the "separator string" that may be used to split the PathName into the components of a hierarchical path name from the root of the associated file system (indicated by the LocalFileSystem).

Note: Some incompatibilities with SMI-S 1.1 (in which this profile was also "EXPERIMENTAL") have been introduced in the parameters to some of the extrinsic methods.

5.1.3 Instance Diagrams

5.1.3.1 File Export Creation classes and associations

Figure 6 illustrates the constructs involved with creating a FileShare for a File Export subprofile. This summarizes the mandatory classes and associations for this subprofile. Specific areas are discussed in later sections.

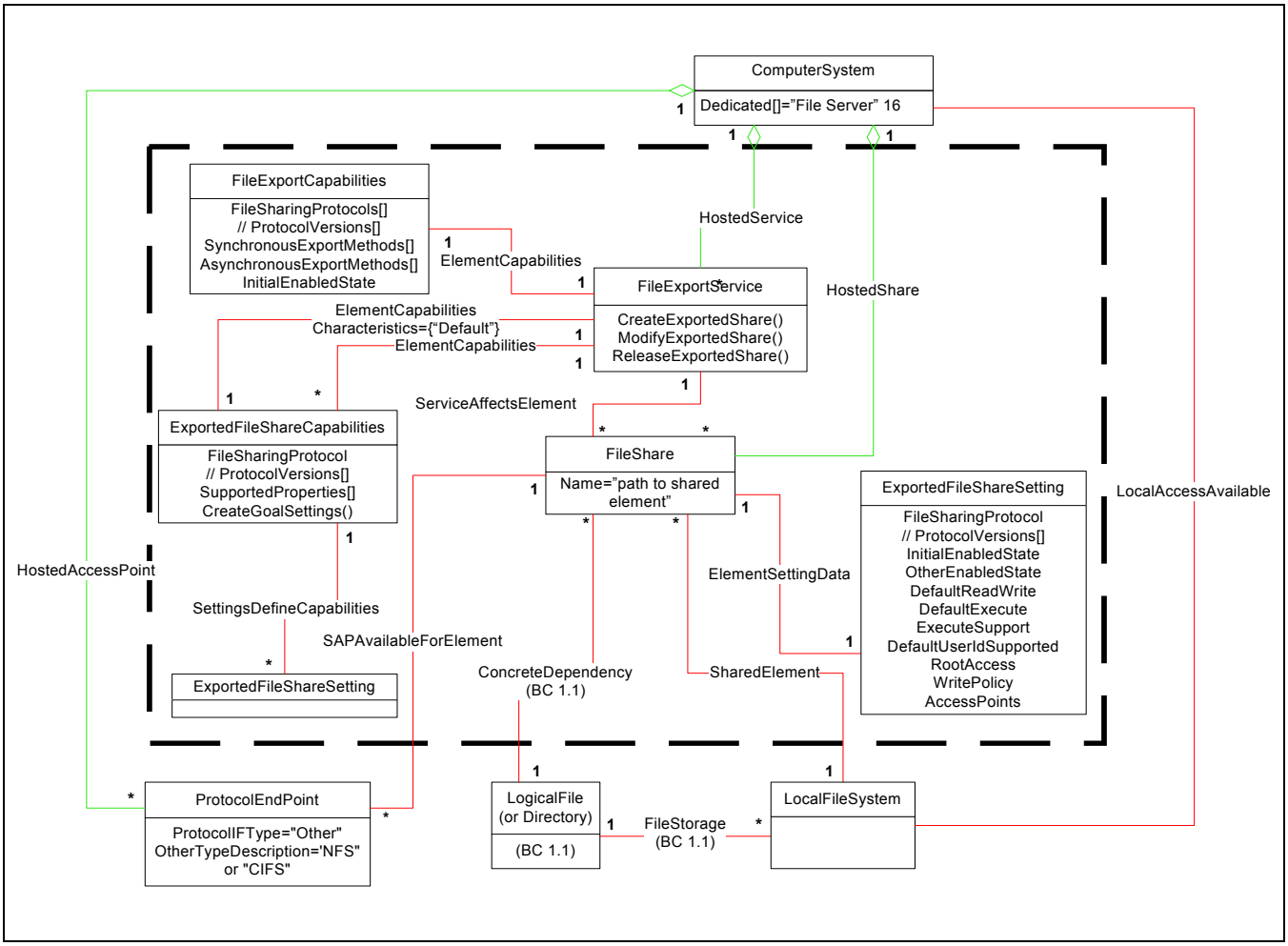

Figure 6 - File Export Manipulation Subprofile Instance

The FileExportService provides configuration support for exporting elements ('files' and 'directories') of a LocalFileSystem as FileShare elements. A FileExportService is hosted by the file server ComputerSystem that exports the directories/files (these would be the file server ComputerSystems in the Filesystem subprofile that were given local access to the filesystem). FileShares are accessed through ServiceAccessPoint(s) hosted by the file server ComputerSystem. FileShares are associated with the FileExportService via ServiceAffectsElement and with the ServiceAccessPoint(s) via SAPAvailableToElement.

If a filesystem related profile supports the File Export Manipulation Subprofile, it shall have at least one FileExportService element. This FileExportService shall be hosted on the top level ComputerSystem of the File Export subprofile (which shall be a file server ComputerSystem element in the filesystem related profiles). The methods offered are CreateFileShare, ModifyFileShare, and ReleaseFileShare.

Associated to the FileExportService (via ElementCapabilities) shall be one FileExportCapabilities element that describes the capabilities of the service. It identifies the methods supported, whether the methods support Job Control or not, the protocols that the created file share can support, and whether or not the file share shall be made available after creation.

For each file sharing protocol that is supported, there shall be one ExportedFileShareCapabilities element that defines the range of capabilities supported for that particular file sharing protocol. The ExportedFileShareCapabilities elements shall be associated via ElementCapabilities to the FileExportService. One of the ExportedFileShareCapabilities may be identified as a default (by setting the property ElementCapabilities.IsDefault). The default ExportedFileShareCapabilities element also indicates the default file sharing protocol to be supported. These defaults apply if any of the extrinsic methods of the FileExportService are invoked with a NULL value for the Capabilities parameter.

Each ExportedFileShareCapabilities element is defined by a set of ExportedFileShareSettings that are associated to it by the SettingsDefineCapabilities association. These ExportedFileShareSettings may be structured to indicate a range of supported and unsupported property values and shall have the same value for the FileSharingProtocol property as the ExportedFileShareCapabilities element.

For the convenience of clients the File Export Manipulation subprofile may populate a set of “pre-defined” ExportedFileShareSettings for each of the ExportedFileShareCapabilities. These shall be associated to the ExportedFileShareCapabilities via the SettingsDefineCapabilities association -- the association shall have its SettingsDefineCapabilities.PropertyPolicy property set to "Correlated" and theSettingsDefineCapabilities.ValueRole property set to "Supported".

Note: That they are pre-defined and therefore exist at all times does not imply that these ExportedFileShareSettings must be made persistent by the implementation.

The ExportedFileShareCapabilities instance supports the CreateGoalSettings method, described in detail in 5.5.1, "Extrinsic Methods of the Profile". This method supports establishing one client-defined ExportedFileShareSettings (as a goal).

CreateGoalSettings takes an array of embedded SettingData elements as the input TemplateGoalSettings and SupportedGoalSettings parameters and may generates an array of embedded SettingData elements as the output SupportedGoalSettings parameter. However, this profile only uses a single embedded ExportedFileShareSettings element in the input parameters (both TemplateGoalSettings and SupportedGoalSettings) and generate a single valid embedded ExportedFileShareSettings element as output (SupportedGoalSettings). If a client supplies a NULL ExportedFileShareSettings (i.e., the empty string) as input to this method, the returned ExportedFileShareSettings structure shall be a default setting for the parent ExportedFileShareCapabilities. If the input (the embedded ExportedFileShareSettings) is not NULL, the method may return a “best fit” to the requested setting. The client may iterate on the CreateGoalSettings method until it acquires a setting that suits its needs. This embedded settings structure may then be used when the CreateFileShare or ModifyFileShare methods are invoked. The details of how iterative negotiation can work are discussed in 5.5.1.1, "ExportedFileShareCapabilities.CreateGoalSettings". Note that the file sharing protocol indicated by the FileSharingProtocol property is invariant in all of these interactions. It is an error if the client changes the FileSharingProtocol property for a Setting and submits it as a goal to the Capabilities element that provided the original Setting.

Note: It is not possible to guarantee that negotiation will terminate with an agreed upon setting and a fall-back mechanism is needed. This profile does not require negotiation -- an implementation may support only a set of pre-defined correlated point settings that a client can preload and use without modification. The implementation could also support only settings whose properties are selectable from an arithmetic progression or from a fixed enumeration, so that the client may construct a goal setting that is guaranteed to be supported without negotiation.

Note: That a client has obtained a goal setting supported by the implementation does not guarantee that a create or modify request will succeed. Such a setting only specifies a supported static configuration not that the current dynamic environment has the resources to implement a specific request.

Armed with the goal (the embedded ExportedFileShareSettings element), a reference to a LocalFileSystem, and a path to a file or directory contained within that LocalFileSystem, the client can now use the CreateFileShare method to create the file share for export. The CreateFileShare method creates a FileShare element, and a new ExportedFileShareSettings instance as well as several necessary associations. These associations are:

- HostedFileShare association between the FileShare and the file server ComputerSystem that hosts it.
- SharedElement association between the FileShare and the LocalFileSystem. In addition, the FileShare element specifies the pathname for the shared element (file or directory) relative to the root of the LocalFileSystem (using the Name property).
- ElementSettingData to associate the FileShare to the ExportedFileShareSetting defined for it
- For backward compatibility with the SMI-S 1.1 File Export subprofile:
  - The file or directory of the LocalFileSystem that is being shared is represented as a LogicalFile
  - A FileStorage association is created between the LogicalFile and the LocalFileSystem
  - A ConcreteDependency association is created between the FileShare and the LogicalFile.
- In addition, optional parameters to the method can cause other classes to be created:
  - DefaultUserId could create a Privilege (see Clause 5: File Export Manipulation Subprofile of *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*) associated to the FileShare as AuthorizationTarget and to a UserIdentity as AuthorizationSource
  - RootAccessHosts array parameter could create root access Privileges from a number of remote Host ComputerSystems to the FileShare (also using the Security Authorization subprofile)
  - AccessPointPorts array parameter could create SAPAvailableForElement associations to a number of ProtocolEndPoints representing TCP/IP ports through which access is provided to this FileShare.

The ReleaseFileShare method is straightforward -- it deletes the FileShare and the ExportedFileShareSetting**,** and the associations to those elements (HostedFileShare, the ElementSettingData element, SharedElement, all the SAPAvailableForElement associations and all Privileges that reference this FileShare as an AuthorizationTarget). Any ComputerSystem elements created to represent remote hosts with root access to this FileShare that have no further references may also be removed. A LogicalFile element associated to the LocalFileSystem via FileStorage will not necessarily be deleted (the implementation may keep track of the other users of this element and be able to delete it). For similar reasons, a ProtocolEndPoint created as a result of being specified in the AccessPointPorts parameter may not be deleted. In both these cases, if the element has no associations other than the scoping one (FileStorage to LocalFileSystem for LogicalFile and HostedAccessPoint to ComputerSystem for ProtocolEndPoint) the provider may stop surfacing it at any time.

The ModifyFileShare method modifies an existing FileShare -- this requires a new ExportedFileShareSetting element to be used as a goal. But not any ExportedFileShareSetting will do; the client shall use the ExportedFileShareCapabilities.CreateGoalSettings method which would have been used to create the file share, or an appropriate compatible ExportedFileShareCapabilities instance. The CreateGoalSettings method is used to establish a new ExportedFileShareSetting goal (as with the original file share creation, it may be necessary to iterate on the CreateGoalSettings method). Since only properties controlled by Settings can be changed by ModifyFileShare, elements surfaced as a side-effect of creating or modifying a file share (i.e., any ComputerSystems created to represent remote hosts with root access or an ProtocolEndPoints created to represent access points for the share, or any user id created as a default user id) cannot be deleted, though new ones can be created and/or added), the effect of ModifyFileShare is to change some properties of the FileShare or of the associated ExportedFileShareSetting.

5.1.3.2 Finding File Export Services, Capabilities and Pre-defined Settings

When creating a file share the first step is to determine what can be created. Figure 7 illustrates an instance diagram showing the elements that shall exist for supporting fileshare creation.

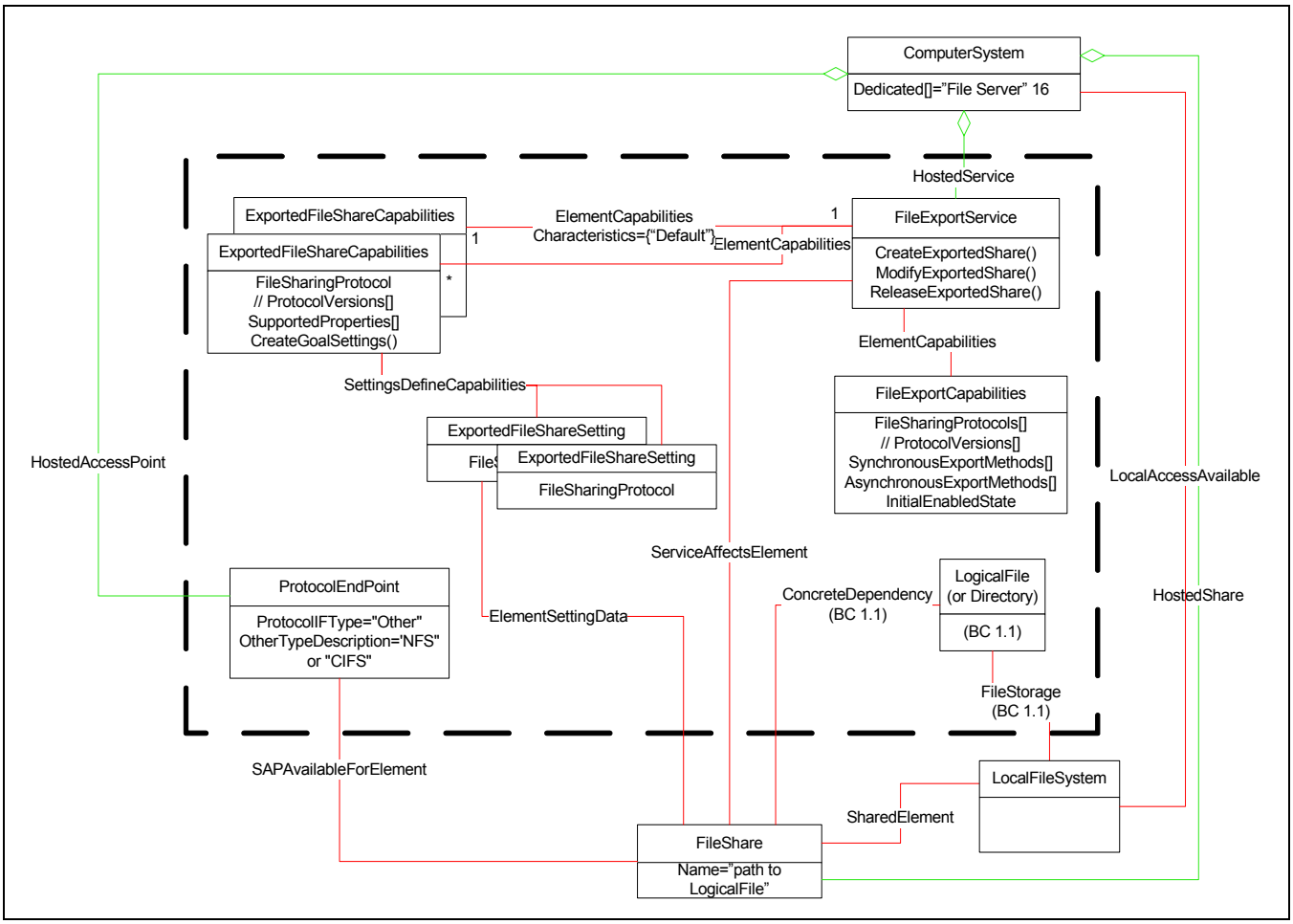

Figure 7 - Capabilities and Settings for Exported File Share Creation

At least one FileExportService shall exist if the File System Profile has implemented the File Export Manipulation Subprofile. The instance(s) of this service can be found by following the HostedService association and filtering on the target class of FileExportService.

Note: If no service is found from the Top Level file server ComputerSystem, the client should look for other component file server ComputerSystems that may be hosting the service. This is not recommended, but permitted for backward compatibility with SMI-S 1.1.

An instance of the FileExportCapabilities shall be associated to the FileExportService via the ElementCapabilities association. A client should follow this association (filtering on the result value of "CIM_FileExportCapabilities") to inspect the configuration capabilities that are supported. The client would choose between the file sharing protocols specified in the array property FileSharingProtocol.

For each entry in the FileSharingProtocol array, there shall be one instance of ExportedFileShareCapabilities with the same value for the FileSharingProtocol property that shall be associated to the FileExportService using the ElementCapabilities association (filtering on the result value of "CIM_ExportedFileShareCapabilities"). This

ExportedFileShareCapabilities element shall specify the supported capabilities for that FileSharingProtocol using a collection of ExportedFileShareSetting elements. These ExportedFileShareSetting shall be associated the ExportedFileShareCapabilities via SettingsDefineCapabilities.

An implementation may support a set of pre-defined ExportedFileShareSetting that clients could use directly if desired. The SettingsDefineCapabilities association from the ExportedFileShareCapabilities to the pre-defined ExportedFileShareSettings shall have the PropertyPolicy property be "Correlated", the ValueRole property be "Supported" and the ValueRange property be "Point". Other pre-defined combinations of property values may be specified by ExportedFileShareSetting whose SettingsDefineCapabilities association has the PropertyPolicy be "Independent", ValueRole property be "Supported" and the ValueRange array property contain "Minimums", "Maximums", or "Increment". These settings can be used by the client to compose ExportedFileShareSetting that are more likely to be directly usable.

5.2 Health and Fault Management Considerations

The key elements of this profile are the FileExportService and the file server ComputerSystem.

5.2.1 OperationalStatus for FileExportService

Table 11 - Operational Status for FileExport Service

Primary OperationalStatus	Description
2 “OK”	The service is running with good status
3 “Degraded”	The service is operating in a degraded mode. This could be due to the health state of the underlying file server, or of the storage being degraded or in error.
4 “Stressed”	The services resources are stressed
5 “Predictive Failure”	The service might fail because some resource or component is predicted to fail
6 “Error”	An error has occurred causing the service to become unavailable. Operator intervention through SMI-S to restore the service may be possible.
6 “Error”	An error has occurred causing the service to become unavailable. Automated recovery may be in progress.
7 “Non-recoverable Error”	The service is not functioning. Operator intervention through SMI-S will not fix the problem.
8 “Starting”	The service is in process of initialization and is not yet available operationally.
9 “Stopping”	The service is in process of stopping, and is not available operationally.
10 “Stopped”	The service cannot be accessed operationally because it is stopped -- if this did not happened because of operator intervention or happened in real-time, the OperationalStatus would have been “Lost Communication” rather than “Stopped”.
11 “In Service”	The service is offline in maintenance mode, and is not available operationally.

Table 11 - Operational Status for FileExport Service

Primary OperationalStatus	Description
13 “Lost Communications”	The service cannot be accessed operationally -- if this happened because of operator intervention it would have been “Stopped” rather than “Lost Communication”.
14 “Aborted”	The service is stopped but in a manner that may have left it in an inconsistent state.
15 “Dormant”	The service is offline; and the reason for not being accessible is unknown.
16 “Supporting Entity in Error”	The service is in an error state, or may be OK but not accessible, because a supporting entity is not accessible.

5.2.2 OperationalStatus for File Server ComputerSystem

Table 12 - Operational Status for File Server ComputerSystem

Primary OperationalStatus	Description
2 “OK”	The file server is running with good status
3 “Degraded”	The file server is operating in a degraded mode. This could be due to the health state of some component of the ComputerSystem, due to load by other applications, or due to the health state of backend or front-end network interfaces.
4 “Stressed”	The file server resources are stressed
5 “Predictive Failure”	The file server might fail because some resource or component is predicted to fail
6 “Error”	An error has occurred causing the ComputerSystem to become unavailable. Operator intervention through SMI-S to restore the service may be possible.
6 “Error”	An error has occurred causing the ComputerSystem to become unavailable. Automated recovery may be in progress.
7 “Non-recoverable Error”	The file server ComputerSystem is not functioning. Operator intervention through SMI-S will not fix the problem.
8 “Starting”	The ComputerSystem is in process of initialization and is not yet available operationally.
9 “Stopping”	The ComputerSystem is in process of stopping, and is not available operationally.
10 “Stopped”	The ComputerSystem cannot be accessed operationally because it is stopped -- if this did not happened because of operator intervention or happened in real-time, the OperationalStatus would have been “Lost Communication” rather than “Stopped”.

Table 12 - Operational Status for File Server ComputerSystem

Primary OperationalStatus	Description
11 “In Service”	The ComputerSystem is offline in maintenance mode, and is not available operationally.
13 “Lost Communications”	The ComputerSystem cannot be accessed operationally -- if this happened because of operator intervention it would have been “Stopped” rather than “Lost Communication”.
14 “Aborted”	The ComputerSystem is stopped but in a manner that may have left it in an inconsistent state.
15 “Dormant”	The ComputerSystem is offline; and the reason for not being accessible is unknown.
16 “Supporting Entity in Error”	The ComputerSystem is in an error state, or may be OK but not accessible, because a supporting entity is not accessible.

5.3 Cascading Considerations

Not Applicable.

5.4 Supported Subprofiles and Packages

Table 20 describes the supported profiles for File Export Manipulation.

Table 13 - Supported Profiles for File Export Manipulation

Registered Profile Names	Mandatory	Version
Job Control	No	1.3.0
File Export	Yes	1.3.0
Security	No	1.1.0
Indication	Yes	1.3.0

5.5 Methods of the Profile

5.5.1 Extrinsic Methods of the Profile

Table 14 - FileExportManipulation Methods

Method	Created Instances	Deleted Instances	Modified Instances
CreateExportedShare	FileShare (Export) ExportedFileShareSetting ElementSettingData HostedShare SharedElement SAPAvailableForElement ServiceAffectsElement LogicalFile (or Directory) *(for bc to 1.1)* ProtocolEndPoint	N/A	N/A
ModifyExportedShare			ExportedFileShareSetting FileShare (Export) ProtocolEndPoint
ReleaseExportedShare	N/A	FileShare (Export) ExportedFileShareSetting ElementSettingData HostedShare SharedElement ServiceAffectsElement ProtocolEndPoint LogicalFile	N/A
CreateGoalSettings	N/A	N/A	N/A

5.5.1.1 ExportedFileShareCapabilities.CreateGoalSettings

This extrinsic method of the ExportedFileShareCapabilities class validates support for a caller-proposed ExportedFileShareSetting passed as the TemplateGoalSettings parameter. This profile restricts the usage of this method to a single entry array for both TemplateGoalSettings and SupportedGoalSettings parameters.

If the input TemplateGoalSettings is NULL or the empty string, this method returns a single default ExportedFileShareSetting in the SupportedGoalSettings array. Both TemplateGoalSettings and

SupportedGoalSettings are string arrays containing embedded instances of type ExportedFileShareSetting. As such, these settings do not exist in the implementation but are the responsibility of the client.

If the TemplateGoalSettings specifies values that cannot be supported, this method shall return an appropriate error and should return a best match for a SupportedGoalSettings.

The client and the implementation can engage in a negotiation process that may require iterative calls to this method. To assist the implementation in tracking the progress of the negotiation, the client may pass previously returned values of SupportedGoalSettings as the new input value of SupportedGoalSettings. The implementation may determine that a step has not resulted in progress if the input and output values of SupportedGoalSettings are the same. A client may infer from the same result that the TemplateGoalSettings must be modified.

5.5.1.1.1 Client Considerations

It is important to understand that the client is acting as an agent for a human user -- either a "system" administrator, or other entity with administrative privileges with respect to the filesystem, the filesystem host, or the file server or the file share. During negotiation, the client will show the current state to the user -- the SupportedGoalSettings received to date (either the latest or some subset), the TemplateGoalSettings proposed (the most recent, but possibly more). But the administrator needs a representation of what is available, possibly the range or sets of values that the different setting properties can take. Some decisions are assumed to have been made already, such as the file-sharing protocol to be used or the filesystem element to be shared or the resources allocated for providing local access to the filesystem from the file server.

The SettingsDefineCapabilities association from the selected Capabilities element provides the information for the client to lay out these options. "Point" settings can be identified using ExportedFileShareSettings -- these points can be further qualified to indicate whether these are supported (or not), and even whether they represent some ideal point in the space -- a "minimum", or a "maximum", or an "optimal" point. Other settings can provide ranges for properties -- by specifying a minimum, a maximum, and an increment an arithmetic progression of values can be specified (a continuous range can be specified with a zero increment). Specifying a set of supported values for a property that do not follow some pattern is possible, if a bit tedious.

The set of settings associated via SettingsDefineCapabilities are expected to be quite stable -- real systems do not continually vary the functionality they can support. Such variations do occur -- for instance, if a new PCMCIA card is added to a running system -- and the best way for a client to be able to add these to the set of choices presented to a user is to subscribe to indications on new Capabilities elements and new instances of SettingsDefineCapabilities.

There is no guarantee that a negotiation will terminate successfully with the client and the implementation achieving agreement. The implementation may support some simpler mechanisms, short of fully-fledged negotiation, that would be used by a client to obtain an acceptable TemplateGoalSettings. The following two use cases are easily covered:

1) Client only considers only the canned settings specified exactly. The canned settings are specified by the ExportedFileShareSetting elements that are associated to the ExportedFileShareCapabilities via SettingDefinesCapabilities association with the following property values:

   - SettingDefinesCapabilities.PropertyPolicy = "Correlated"
   - SettingDefinesCapabilities.ValueRole = "Supported"
   - SettingDefinesCapabilities.ValueRange = "Point"

2) Client considers canned settings that range over values specified using minimum/increment/maximum for the Setting properties. For example, these could be specified by the ExportedFileShareSetting elements that are associated to the ExportedFileShareCapabilities via SettingDefinesCapabilities association with the following property values:

   - SettingDefinesCapabilities.ValueRange = "Minimums" or "Maximums" or "Increments"

- The PropertyPolicy and ValueRole properties of SettingDefinesCapabilities will be appropriately specified

5.5.1.1.2 Signature and Parameters of CreateGoalSettings

Table 15 - Parameters for Extrinsic Method ExportedFileShareCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	EmbeddedInstance ("*SNIA_ExportedFileShareSetting*") TemplateGoalSettings is a string array containing embedded instances of class ExportedFileShareSetting, or a derived class. This parameter specifies the client's requirements and is used to locate matching settings that the implementation can support.
SupportedGoalSettings[]	INOUT	string	EmbeddedInstance ("*SNIA_ExportedFileShareSetting*") SupportedGoalSettings is a string array containing embedded instances of class ExportedFileShareSetting, or a derived class. On input, it specifies a previously returned set of Settings that the implementation could support. On output, it specifies a new set of Settings that the implementation can support. If the output set is identical to the input set, both client and implementation may conclude that this is the best match for the TemplateGoalSettings that is available. If the output does not match the input and the non-NULL output does not match the non-NULL TemplateGoalSettings, then the method shall return "Alternative Proposed". If the output is NULL, the method shall return an "Failed".
Normal Return			
Status		uint32	"Success", "Failed", "Timeout", "Alternative Proposed"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

5.5.1.2 FileExportServices.CreateExportedShare

This extrinsic method creates a FileShare providing access to a contained sub-element of a LocalFileSystem (either a LogicalFile or its sub-class Directory). A reference to the created FileShare is returned as the output parameter TheShare. This FileShare element is hosted by the same file server ComputerSystem that hosts the FileExportService. The LocalFileSystem whose element is exported shall be locally accessible to the file server ComputerSystem (and need not be hosted by it), as represented by the LocalAccessAvailable association from the file server ComputerSystem to the LocalFileSystem.

The LocalFileSystem whose sub-element is being exported is specified by the input parameter Root. The input string parameter SharedElementPath specifies a pathname from the root directory of the Root to the sub-element to be exported. If SharedElementPath is NULL or the empty string, it specifies the root directory of Root. The format of SharedElementPath is implementation-specific -- the most common format is as a sequence of directory names separated by a character or short string indicated by the FileSystemSetting.PathNameSeparatorString property.

Note: The root directory of Root is the base from which a path to the LogicalFile being shared is specified. In the simplest and possibly the most common case, the LogicalFile element is the root directory of Root and the path is NULL or the empty string.

The desired settings for the FileShare are specified by the Goal parameter (a string-valued EmbeddedInstance object of class ExportedFileShareSetting). An ExportedFileShareSetting element shall be created that represents the settings of the created FileShare and will be associated via ElementSettingData to the FileShare. (This ExportedFileShareSetting may be identical to the Goal or may be its equivalent). The created element shall be returned as the output Goal parameter.

The input Goal parameter can be NULL, in which case the ExportedFileShareSetting associated with the default ExportedFileShareCapabilities of the FileExportService is used as the goal. In that case, the following references to Goal are to the output value of the Goal parameter.

If Goal.DefaultUserIdSupported="Share-Specified Default User Id" and the input parameter DefaultUserId is not NULL, the FileShare will support the specified user id as the default user when the share is accessed. This access privilege will be represented by creating instances of the Privilege class as described in the Security Authorization subprofile. The Security Authorization subprofile shall be used for fine-grained access to, or modification of, the default user.

Note: If the Security Authorization subprofile is not supported, this parameter may be set at creation but cannot be accessed later. It can only be replaced with a new DefaultUserId using the ModifyExportedShare method.

Note: The format of the user id is not specified by this sub-profile. If a Security Principal sub-profile or a Filesystem Quota subprofile is defined, the user id format defined therein may be used.

If Goal.RootAccess="Allow Root Access" then the input parameter RootAccessHosts will be an array of URIs of ComputerSystems from which root access will be permitted. This access privilege will be represented by creating instances of the Privilege class as described in the Security Authorization subprofile. The Security Authorization subprofile shall be used for fine-grained access to, or modification of, the set of hosts with root access.

Note: If the Security Authorization subprofile is not supported, this parameter may be set at creation but cannot be accessed later. It can only be replaced by specifying a new RootAccessHosts array using the ModifyExportedShare method.

Note: The computer systems may not be managed by this implementation, so they may not be represented by ComputerSystem references.

If Goal.AccessPoints="Named Points", then the input parameter AccessPointPorts will be an array of references to ProtocolEndpoints that provide access to this FileShare. This will be represented by creating instances of the SAPAvailableForElement association between the FileShare and the specified ProtocolEndpoint. Fine-grained

access to this set of ProtocolEndpoints or modification this set can be performed using the ModifyExportedShare method.

Note: This changes the type of the AccessPointPorts parameter from a string array in the previous version to an array of references to ProtocolEndpoints (or more generally to ServiceAccessPoints).

5.5.1.2.1 Signature and Parameters of CreateExportedShare

Table 16 - Parameters for Extrinsic Method FileExportServices.CreateExportedShare

Parameter Name	Qualifier	Type	Description & Notes
ElementName	IN	string	An end user relevant name for the FileShare being created. If NULL, then a system-supplied default name can be used. The value shall be stored in the 'ElementName' property for the created element.
Comment	IN	string	An end user relevant comment for the FileShare being created. If NULL, then a system-supplied default comment can be used. The value shall be stored in the 'Description' property for the created element.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Root	IN, REF	SNIA_LocalFileSystem	A reference indicating a LocalFileSystem element whose sub-element is being exported. The LocalFileSystem shall be locally available (either explicitly or implicitly) to the file server ComputerSystem that hosts the FileExportService.
SharedElementPath	IN, OUT	string	An opaque string representing a path to the shared element from the root directory of the FileSystem indicated by the Root parameter. The format of this is as a sequence of directory names (from the \"root\") separated by the PathNameSeparatorString property. Multiple paths could lead to the same element but the access rights or other privileges could be specific to the path. The client needs to specify the path. If SharedElementPath is NULL or is the empty string, it indicates the \"root\" directory of the file system indicated by Root. The value shall be stored in the 'Name' property for the created element.

Table 16 - Parameters for Extrinsic Method FileExportServices.CreateExportedShare

Parameter Name	Qualifier	Type	Description & Notes
Goal	IN, OUT, EI	string	EmbeddedInstance ("SNIA_ExportedFileShareSetting") The client-specified requirements for how the specified FileShare element is to be shared or exported by the FileExportService. This is an element of the SNIA_ExportedFileShareSetting class, or a derived class, encoded as a string-valued embedded object parameter. If NULL or the empty string, the default configuration will be specified by the FileExportService.
TheShare	OUT, REF	CIM_FileShare	If successful, this returns a reference to the created file share.
DefaultUserId	IN, OUT, REF, NULL allowed,	CIM_identity	A reference to a concrete derived class of CIM_Identity that indicates the user id to use for default access to this share. A NULL value on input indicates that no user id is requested. A NULL value on output indicates that no user id has been assigned, even by default. The provider is expected to surface this access using the Authorization subprofile. A default user id per share is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.
RootAccessHosts[]	IN, OUT, URI, NULL allowed	string	An array of strings that specify the hosts that have root access to this Share, if the SNIA_ExportedFileShareSetting.RootAccess property is set to 'Allow Root Access'. Each entry specifies a host by a URI. All entries up to the first empty string are allowed root access; the entries after the first empty string are denied root access. If this parameter is NULL, root access will be denied to all hosts, effectively overriding the value of the property SNIA_ExportedFileShareSetting.RootAccess. If the first entry is the empty string, root access will be allowed from all hosts, and subsequent entries will be denied root access. The provider is expected to surface this access using the Authorization subprofile. This property needs to be an array of URIs because the remote host may not be known to the provider and therefore a reference to the host may not exist. Root Access is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.

Table 16 - Parameters for Extrinsic Method FileExportServices.CreateExportedShare

Parameter Name	Qualifier	Type	Description & Notes
AccessPointPorts[]	IN, OUT, REF, NULL Allowed	CIM_ServiceAccessPoints	An array of references to the ProtocolEndpoints that can connect to this Share, if the SNIA_ExportedFileShareSetting.AccessPoints property is set to 'Named Ports'. If the parameter is NULL, all access points will be denied access, effectively overriding the value of the property ExportedFileShareSetting.AccessPoints. If the array has multiple entries and the first entry in the array is NULL, all access points supported by the service will be supported, and subsequent entries will be denied access. The provider is expected to surface these access rights (whether granted or denied) using the Authorization subprofile. Any AccessPoints granted access via this parameter will also be associated to this share with SAPAvailableForElement. If the AccessPoint is not already enabled it will appear in a disabled state. The CIFS protocol does not support multiple ProtocolEndpoints, so this is ignored if the Goal specifies creating a CIFSShare.
Normal Return			
Status	OUT	uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

5.5.1.3 FileExportServices.ModifyExportedShare

This extrinsic method modifies a FileShare element that is exporting a contained sub-element of a LocalFileSystem (either a LogicalFile or its sub-class Directory). The FileShare is specified by the reference parameter TheShare. TheShare cannot be NULL and it shall be hosted by the same file server ComputerSystem that hosts the FileExportService. The input parameters Root and SharedElementPath shall be NULL or shall be the same as the corresponding parameters when the FileShare was created (i.e., these cannot be changed using this method).

The precise constraint is that the sub-element shall not be changed even if the Root and SharedElementPath are different. For instance, this would allow a different path that leads to the same sub-element. However, this subprofile does not allow this flexibility.

The new desired settings for the FileShare are specified by the input parameter Goal (a string-valued EmbeddedInstance object of class ExportedFileShareSetting). An ExportedFileShareSetting element that represents the settings of the created FileShare (either identical to the Goal or its equivalent) will be associated via ElementSettingData to the FileShare. The implementation shall modify the existing ExportedFileShareSetting. The Setting that is actually established will be returned as the output parameter Goal.

The Goal parameter can be NULL on input, in which case, the settings for the FileShare are not changed. This can happen if this method is being called to provide new values for DefaultUserId, RootAccessHosts, or AccessPointPorts without changing any settings. In that case, the following references to Goal are to the output value or the parameter.

If Goal.DefaultUserIdSupported="Share-Specified Default User Id" and the input parameter DefaultUserId is not NULL, the FileShare will support the specified user id as the default user when the share is accessed. Any existing DefaultUserId specified will be overridden. This access privilege will be represented by creating instances of the Privilege class as described in the Security subprofile. The Security subprofile shall also be used to access or modify this privilege. If DefaultUserId is NULL, the existing specification will not be changed.

Note: If the Security subprofile is not supported, this parameter may be set but cannot be accessed later. It can only be replaced with a new DefaultUserId using the ModifyExportedShare method.

If Goal.RootAccess="Allow Root Access" then the non-NULL input parameter RootAccessHosts will be an array of URIs of ComputerSystems from which root access will be permitted. This access privilege will be represented by creating instances of the Privilege class as described in the Security subprofile. Any existing specification of root access by hosts will be overridden. If RootAccessHosts is NULL, the existing specification will not be changed.

Note: If the Security subprofile is not supported, this parameter may be set at creation but cannot be accessed later. It can only be replaced by specifying a new RootAccessHosts array using the ModifyExportedShare method.

If Goal.AccessPoints="Named Points", then the non-NULL input parameter AccessPointPorts will be an array of references to ProtocolEndpoints that provide access to this FileShare. This will be represented by creating instances of the SAPAvailableForElement association between the FileShare and the specified ProtocolEndpoint. Any existing specification of access points to the FileShare will be overridden. If AccessPointPorts is NULL, the existing specification will not be changed.

Note: This changes the type of the AccessPointPorts parameter from a string array to an array of references to ProtocolEndpoints (or more generally to ServiceAccessPoints).

The input parameters InUseOptions and WaitTime provide for delaying or aborting the execution of this method if the FileShare is in use and the method requires that no operations be in progress during that execution.

Note: The WaitTime and InUseOptions parameters are supported if the ExportedFileShareCapabilities.SupportedProperties includes the "RequireInUseOptions" option. This requires a change to the MOF that may not show up in this document as enumerations are not documented in the spec?.

5.5.2 Signature and Parameters of ModifyExportedShare

Table 17 - Parameters for Extrinsic Method FileExportServices.ModifyExportedShare

Parameter Name	Qualifier	Type	Description & Notes
ElementName	IN	string	A new end-user relevant name for the FileShare being modified. If NULL or the empty string, the existing name stored in the 'ElementName' property for the created element not be changed.
Comment	IN	string	A new end-user relevant comment for the FileShare being modified. If NULL or the empty string, the existing comment stored in the 'Description' property will not be changed.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Root	IN, OUT, REF	CIM_ManagedElement	A reference indicating a LocalFileSystem element whose sub-element is being exported. In the ModifyExportedShare method, this shall not indicate a different filesystem from the one indicated when the file share was created (even if the reference is to a different instance of LocalFileSystem). If Root is NULL on input it is ignored. As an OUT parameter, a reference to the LocalFileSystem is returned.
SharedElementPath	IN, OUT	string	A string representing a path to the shared element from the root directory of the LocalFileSystem indicated by Root. The ModifyExportedShare method cannot be used to change the object indicated by the path, but the path itself can be different as multiple paths could lead to the same element. Such a change may have side-effects, for instance, the access rights or other privileges could be specific to the path. If SharedElementPath is NULL, it indicates no change to the current path. If SharedElementPath consists of a single empty string, it indicates the root directory of the FileSystem indicated by Root. As an OUT parameter, the current path is returned. The value shall be stored in the 'Name' property for the created element.

Table 17 - Parameters for Extrinsic Method FileExportServices.ModifyExportedShare

Parameter Name	Qualifier	Type	Description & Notes
Goal	IN, OUT, EI	string	EmbeddedInstance ("SNIA_ExportedFileShareSetting") The client-specified requirements for how the specified FileShare element is to be shared or exported by the FileExportService. This is an element of the SNIA_ExportedFileShareSetting class, or a derived class, encoded as a string-valued embedded instance parameter. If NULL or the empty string, the current setting will be re-applied. As an OUT parameter, the current Setting is returned.
TheShare	IN, OUT, REF	CIM_FileShare	As an IN Parameter, it specifies the share that is to be modified or whose settings are being queried. As an OUT Parameter, this specifies the share if the request is successful.
DefaultUserId	IN, OUT, REF, NULL allowed,	CIM_identity	As an IN parameter, this is a reference to a concrete derived class of CIM_Identity that indicates the user id to use for default access to this share. A NULL value indicates no change to the existing user id, if one has been specified. The provider is expected to surface this access using Authorization subprofile. As an OUT Parameter, this returns a reference to the current DefaultUserId. A default user per share is not supported by the CIFS Protocol so this is ignored if the file share is a CIFSShare.

Table 17 - Parameters for Extrinsic Method FileExportServices.ModifyExportedShare

Parameter Name	Qualifier	Type	Description & Notes
RootAccessHosts[]	IN, OUT, URI, NULL allowed	string	An array of strings that specify the hosts that have root access to this Share, if the SNIA_ExportedFileShareSetting.RootAccess property is set to 'Allow Root Access'. Each entry specifies a host by a URI. The set of hosts specified is added to the existing set of hosts with root access. If this parameter is NULL, root access will be denied to all hosts, including the ones currently allowed root access, effectively overriding the value of the property SNIA_ExportedFileShareSetting.RootAccess. Each entry specifies a host by a URI. All entries up to the first empty string are allowed root access; the entries after the first empty string are denied root access. If the first entry is the empty string, root access will continue to be allowed from the existing hosts, and subsequent entries in the array will be denied root access. The provider is expected to surface this access using the Authorization subprofile. This property needs to be an array of URIs because the remote host may not be known to the provider and therefore a reference to the host may not exist. Root Access is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.

Table 17 - Parameters for Extrinsic Method FileExportServices.ModifyExportedShare

Parameter Name	Qualifier	Type	Description & Notes
AccessPointPorts[]	IN, OUT, REF, NULL Allowed	CIM_Service AccessPoints	An array of references to the ProtocolEndpoints that can connect to this Share, if the SNIA_ExportedFileShareSettings.AccessPoints property is set to 'Named Ports'. The set of access points specified in the array is added to the existing set of access points. If the parameter is NULL, all access points will be denied access, effectively overriding the value of the property SNIA_ExportedFileShareSetting.AccessPoints. If the first entry in the array is NULL, existing access points supported by the service will be supported, and subsequent entries in the array will be access points that are denied access. The provider is expected to surface these access rights (whether granted or denied) using the Authorization subprofile. Any AccessPoints granted access via this parameter will also be associated to this share with SAPAvailableForElement. If the AccessPoint is not already enabled it will appear in a disabled state. The CIFS protocol does not support multiple ProtocolEndpoints, so this is ignored if the Goal specifies creating a CIFSShare.
InUseOptions	IN	uint16	An enumerated integer that specifies the action that the provider should take if the FileShare is still in use when this request is made. The WaitTime parameter indicates the specified time used for this function. This option is only relevant if the FileShare needs to be made unavailable while the request is being executed.
WaitTime	IN	uint16	An integer that indicates the time (in seconds) that the provider needs to wait before executing this request if it cannot be done while the FileShare is in use. If WaitTime is not zero, the method will create a job, if supported by the provider, and return immediately. If the provider does not support asynchronous jobs, there is a possibility that the client could time-out before the job is completed. The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request' and will be performed asynchronously if possible.

Table 17 - Parameters for Extrinsic Method FileExportServices.ModifyExportedShare

Parameter Name	Qualifier	Type	Description & Notes
Normal Return			
Status	OUT	uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

5.5.2.1 FileExportServices.ReleaseExportedShare

This is an extrinsic method that will delete a FileShare specified by the parameter TheShare and delete any associated instances and associations that are no longer needed. The deleted instances will include the Directory (or LogicalFile) if it had been created only for the purpose of representing the shared sub-element.

Note: Deleting the Directory or LogicalFile deletes only the representation of the file or directory for management and does not delete the underlying operational element

The deleted associations include HostedShare, ElementSettingData, and any elements and associations created to support the DefaultUserId, RootAccessHosts, and AccessPointPorts parameters. In addition, the ExportedFileShareSetting will be deleted if appropriate.

The input parameters InUseOptions and WaitTime provide for delaying or aborting the execution of this method if the FileShare is in use and the method requires that no operations be in progress during that execution.

Note: The WaitTime and InUseOptions parameters are supported if the ExportedFileShareCapabilities.SupportedProperties includes the "RequireInUseOptions" option.

5.5.3 Signature and Parameters of ReleaseExportedShare

Table 18 - Parameters for Extrinsic Method FileExportServices.ReleaseExportedShare

Parameter Name	Qualifier	Type	Description & Notes
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
TheShare	IN, OUT, REF	CIM_FileShare	As an IN Parameter, it specifies the share that is to be modified or whose settings are being queried. As an OUT Parameter, this specifies the share if the request is successful.
InUseOptions	IN	uint16	An enumerated integer that specifies the action that the provider should take if the FileShare is still in use when this request is made. The WaitTime parameter indicates the specified time used for this function. This option is only relevant if the FileShare needs to be made unavailable while the request is being executed.
WaitTime	IN	uint16	An integer that indicates the time (in seconds) that the provider needs to wait before executing this request if it cannot be done while the FileShare is in use. If WaitTime is not zero, the method will create a job, if supported by the provider, and return immediately. If the provider does not support asynchronous jobs, there is a possibility that the client could time-out before the job is completed. The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request' and will be performed asynchronously if possible.
Normal Return			
Status	OUT	uint32	ValueMap{}, Values{} "Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

5.5.4 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

5.6 Client Considerations and Recipes

EXPERIMENTAL

Conventions used in all the filesystem related profiles for recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is used without further validation. Real code will need to be more robust.
- In SMI-S, Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.
- Error returns using the CIM_Error mechanism are not explicitly handled; the client shall implement handlers for these asynchronous returns.
- These recipes do not show the details of negotiating a setting acceptable to both client and provider.
- The recipes do not show the details of managing a Job if a method returns after setting one up.

All the recipes show very simple examples of the operations that should be supported. Some recipes have been simplified so that they would not even be minimally useful to a real client, but only show how more complete functionality would be implemented.

5.6.1 Creation of a FileShare for Export

```
// DESCRIPTION
//   GOAL: Create an NFS or CIFS FileShare that makes a specified
//         directory or file of a filesystem available to clients
//         and supports the properties specified in the array
//         parameter $propertynames[].
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The file server ComputerSystem host of the FileExportService
//      will also be the host of the FileShare that will be
//      made available to NFS or CIFS clients.
//
// FUNCTION CreateFileSystemShare
//   This function takes a filesystem and a file server host
//      ComputerSystem and creates a file share that will
```

```
//      support the specified properties.
// INPUT Parameters:
//   sharetype: The file sharing protocol (NFS or CIFS) that this
//      share should support.
//   fs: A reference to the LocalFileSystem whose element is
//      to be shared.
//   server: A reference to the file server ComputerSystem that
//      provides local access to the filesystem $fs.
//   fspath: A path to the sub-element that is to be shared.
//   name: A name for the created file share.
//   comment: A comment to be associated with the created file share.
//   propnames: An array of property names that the capabilities
//      element should support.
//   propvals: An array of property values corresponding to the
//      property names that specify values for those properties.
// OUTPUT Parameters:
//   fssh: A reference to the newly created FileShare element
//   job: A reference to a ConcreteJob that is executing a long-term job.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub CreateFileSystemShare(IN String $sharetype,        // CIFS, NFS, etc.
                          IN REF CIM_FileSystem $fs,   // the filesystem
                          IN REF CIM_ComputerSystem $server // the File Server
                         IN String $fspath,          // subpath in the filesystem,
                           or ""
                          IN String $name,
                          IN String $comment,
                         IN String[] $propnames,     // names of desired properties
                          IN String[] $propvals,       // values of desired
                           properties
                          OUT REF CIM_FileShare $fssh,
                          OUT REF CIM_Job $job)
{
    //
    // Get the service and capabilities
    //
////    &GetFileExportServiceAndCapabilities($server, $sharetype, $feservice,
                           $efscapability);
    //
    // Get a FileExportService via the HostedService association to
    // the file server ComputerSystem
    //
    $feservice = Associators($server,
                             "CIM_HostedService",
                             "SNIA_FileExportService",
```

```
                        "Antecedent",
                        "Dependent")->[0];

// Assumption: There is only one FileExportService per File Server
//
// Get an ExportedFileShareCapabilities from the FileExportService
// via the ElementCapabilities association to the ComputerSystem
// (it's indexed by NFS/CIFS/other sharing service and possibly
// other properties)
// Note: NFS and CIFS are two capabilities of the same service
// with different values of the FileSharingProtocol property
// In this example, we look for the
// ExportedFileShareCapabilities.IsDefault property to get a
// default sharetype.
//
$efscapabilities = Associators($feservice,
                               "CIM_ElementCapabilities"
                               "SNIA_ExportedFileShareCapabilities",
                               "ManagedElement",
                               "Capabilities");
if ($efscapabilities->[] == NULL || $efscapabilities-[].length == 0 ) {
#j = 0;
while ( ($efscapability = $efscapabilities->[#j]) != NULL) {
    if ( ($sharetype == "") && $efscapability.IsDefault ||
         ($efscapabilities->[#j].FileSharingProtocol == $sharetype) ) {
        $sharetype = $efscapability.FileSharingProtocol;
        // Should check here that the properties named in
        // $propnames-[] are supported by this capabilities
        // element.  If not, the method should fail as this profile
        // does not support multiple capabilities with the same
        // file sharing protocol that may have different.
        break;
    }
    #j++;
}

// Handle the error if any
if (#j == $efscapabilities-[].length) {
    <indicate error>
    return false;
}

//
// Call FileExportServiceCapabilities.CreateGoalSettings(Goal-N', Goal-N) to
                      get
// the next goal for EFSSetting -- iterate until satisfied or give up
// (beware of infinite loops)  Note: we don't iterate here, just give
// up if we don't get what we want.
```

```
//
// The function used is CreateGoal instead of CreateGoalSettings
// because the CreateGoalSettings method takes arrays
// as parameters and we only want to pass single-entry arrays
// The implementation details are left to the client.
&CreateGoal($efscapability, NULL, $goal);

//
// Inspect Goal and modify properties as desired.
//
#i = 0;
while ($propnames->[#i] != NULL) {
    $goal.$propnames->[#i] = $propvals->[#i];
    #i++;
}

// Iterate over the goal at least once
&CreateGoal($efscapability, $goal, $settings);

#i = 0;
while ($propnames->[#i] != NULL) {
    // funky syntax for propnames property of settings
    if ($settings.$propnames->[#i] != $propvals->[#i]) {
        //
        // give up
        //
        return false;
    }
    #i++;
}

// Verify that the FileSystem is locally accessible

// Does this fileserver have local access -- if not, there is no setting!
$laassocs->[] = ReferenceNames($server,
                               “SNIA_LocalAccessAvailable”,
                               “FileSystem”
                               $fs);
if ($laassocs->[] == NULL || $laassocs->[].length != 1) {
{
    // If the filesystem is not locally accessible from the server
    // there is no setting to be found
    return false;
}
$laassoc = $laassocs->[0];

//
```

```
// Get all the LocallyAccessibleFileSystemSettings
// associated with the CIM_FileSystem (via ElementSettingData)
//
$lasettings->[] = Associators($fs,
                        “CIM_ElementSettingData”,
                        “SNIA_LocallyAccessibleFileSystemSetting”,
                        “ManagedElement”,
                        “SettingData”);
if ($lasettings->[] == NULL || $lasettings->[].length == 0) {
    // This is an ERROR but for now we return with no results
    return NULL;
}

#i = 0;
$lasetting = NULL;
while ($lasettings->[#i] != NULL) {
    // Get the association that points to this setting
    $reference->[] = References($lasettings->[#i],
                                “CIM_ElementSettingData”,
                                “SettingData”);
    // There should be exactly one association to this SettingData
    if ($reference->[] == NULL || $reference->[].length != 1) {
        // This is an error -- should we continue?
        continue;
        // return NULL;
    }

    // The following test assumes that we only look at a setting
    // that is marked as IsCurrent.  There may be many such
    // settings but they will be scoped to other file servers.
    if ($reference->[0].IsCurrent == “Is Current”) {
        // Is this scoped to the fileserver?
        $servers = Associators($settings->[#i],
                               “CIM_ScopedSetting”,
                               “CIM_ComputerSystem”,
                               “Dependent”,
                               “Antecedent”);
        if ($servers->[] != NULL && $servers->[].length != 0 && $servers->[0]
                       == $fileserver) {
            $lasetting = GetInstance($lasettings->[#i]);
            break;
        }
    }
    #i++;
}
// if not found return NULL
if ($lasetting == NULL) {
    return false;
```

```
    }

    //
    // Note, this profile uses the FileSystem $fs as the Root
    // parameter to CreateExportedShare and does not support
    // other classes.
    // The fspath is a string that is FileSystemType-specific
    // If path is NULL or empty, it
    // identifies the root directory of the File System.
    //
//    $feservice.CreateExportedShare($name, $comment,
//                            $job, $fs, $fspath, $settings, $fssh);
    #result = $feservice.CreateExportedShare(
              $name,       // share name
              $comment,    // comment associated with share
              $job,        // OUTPUT parameter if needed
              $fs,         // file system of the shared element
              $fspath,     // relative path to shared element
              $settings,   // EmbeddedInstance of Goal
              $fssh,       // OUTPUT parameter -- reference to File Share
              NULL,        // $defaultUserId -- not being set in this example
              NULL,        // $RootAccessHosts[] -- not being set
              NULL         // $AccessPointPEs[] -- not being set
              )

    // Should handle failure and other errors here.

    return true;
}
```

5.6.2 Modification of an Exported FileShare

```
// DESCRIPTION
//   GOAL: Modify the creation-time settings of a NFS or
//         CIFS FileShare.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The file share already exists and has been surfaced through
//      SMI-S.
//   2. The file share has been defined with an associated
//      ExportedFileShareSettings element and hosted on an
//      surfaced file server ComputerSystem.
//   3. There is a FileExportService element hosted by
//      the same file server ComputerSystem that provides
//      this method.
//
// FUNCTION ModifyFileSystemShare
//   This function modifies the settings and some mutable
//      properties of an existing file share hosted by the
```

```
//      same ComputerSystem as the host of the service.
//      This routine cannot be used to change
//      the filesystem, the sharetype, or the file server.
//      It can be used to change the name, the comment, and
//      setting property values.
// INPUT Parameters:
//   name: A new name for the file share.
//   comment: A comment to be associated with the created file share.
//   fssh: A reference to the newly created FileShare element
//   propnames: An array of property names that the capabilities
//      element should support.
//   propvals: An array of property values corresponding to the
//      property names that specify values for those properties.
// OUTPUT Parameters:
//   job: A reference to a ConcreteJob that is executing a long-term job.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub ModifyFileSystemShare(IN String $name,
                          IN String $comment,
                          IN CIM_FileShare $fssh,
                          IN String $propnames[],
                          IN String $propvals[],
                          OUT CIM_Job $job)
{
    //
    // Get a client-side copy of the ExportedFileShareSetting
    // associated with the ExportedFileShare (via ElementSettingData
    // association) using GetInstance
    //
    $settings = Associators($fssh,
                            “CIM_ElementSettingData”,
                            “CIM_ExportedFileShareSetting”,
                            “ManagedElement”,
                            “SettingData”)->[0];
    #i = 0;
    while ($settings->[#i] != NULL) {
        if ($settings->[#i].IsCurrent) {
            $setting = GetInstance($settings->[#i].Name);
            break;
        }
    }

    //
    // Get the sharetype from the FileSystemShare
```

```
    // -- this cannot be changed by this method
    //
    $sharetype = $setting.FileSharingProtocol;

    //
    // Get the File Server
    //
//    &GetFileExportServer($fs, $server);
    //
    // Get the ComputerSystem for the filesystem (via HostedFileSystem association)
    //  There should be exactly one.
    $server = Associators($fssh,
                          "CIM_HostedFileShare",
                          "CIM_ComputerSystem",
                          "PartComponent",
                          "GroupComponent")->[0];

    //
    // Get the service and capabilities
    //
//    &GetFileExportServiceAndCapabilities($server, $sharetype, $feservice,
                           $efscapability);
    //
    // Get a FileExportService via the HostedService association to
    // the file server ComputerSystem
    //
    $feservice = Associators($server,
                             "CIM_HostedService",
                             "SNIA_FileExportService",
                             "Antecedent",
                             "Dependent")->[0];

    // Assumption: There is only one FileExportService per File Server
    //
    // Get an ExportedFileShareCapabilities from the FileExportService
    // via the ElementCapabilities association to the ComputerSystem
    // (it's indexed by NFS/CIFS/other sharing service and possibly
    // other properties)
    // Note: NFS and CIFS are two capabilities of the same service
    // with different values of the FileSharingProtocol property
    // The $sharetype must match the property
    // ExportedFileShareCapabilities.FileSharingProtocol.
    //
    $efscapabilities = Associators($feservice,
                                   "CIM_ElementCapabilities"
                                   "SNIA_ExportedFileShareCapabilities",
                                   "ManagedElement",
                                   "Capabilities");
```

```
if ($efscapabilities->[] == NULL || $efscapabilities-[].length == 0 ) {
#j = 0;
while ( ($efscapability = $efscapabilities->[#j]) != NULL) {
    if ( $efscapabilities->[#j].FileSharingProtocol == $sharetype ) {
        // Should check here that the properties named in
        // $propnames-[] are supported by this capabilities
        // element.  If not, the method should fail as this profile
        // does not support multiple capabilities with the same
        // file sharing protocol that may have different.
        break;
    }
    #j++;
}

// Handle the error if any
if (#j == $efscapabilities-[].length) {
    <indicate error>
    return false;
}

//
// Modify the copied ExportedFileShareSetting to the new
// desired properties
//
#i = 0;
while ($propnames->[#i] != NULL) {
    // Note funky syntax for accessing a named property of
    // the setting
    $setting.$propnames->[#i] = $propvals->[#i];
}

// Call FileExportServiceCapabilities.CreateGoalSettings(Goal-N', Goal-N) to
                        get
// the next goal for EFSSetting -- iterate until satisfied or give up
// (beware of infinite loops)  Note: we don't iterate here, just give
// up if we don't get what we want.
//
// The function used is CreateGoal instead of CreateGoalSettings
// because the CreateGoalSettings method takes arrays
// as parameters and we only want to pass single-entry arrays
// The implementation details are left to the client.
&CreateGoal($efscapability, $setting,  $newsetting);

// Did we get a goal back?
if ($newsetting==MULL)
#i = 0;
while ($propnames->[#i] != NULL) {
    if ($newsetting.$propnames->[#i] != $propvals->[#i]) {
```

```
                //
                // give up
                //
                return NULL;
            }
            #i++;
        }

        //
        #result = feservice.ModifyExportedShare(
                  $name,       // new name (no change if NULL)
                  $comment,    // new comment (no change if NULL)
                  $job,        // OUTPUT parameter if needed
                  NULL,        // $rootfilesystem - Cannot be changed
                  NULL,        // $Subelement -- cannot be changed
                  $newsetting, // EmbeddedInstance of Goal
                  $fssh,       // reference to File Share
                  NULL,        // $defaultUserId -- not being changed in this example
                  NULL,        // $RootAccessHosts[] -- not being changed
                  NULL,        // $AccessPointPEs[] -- not being changed
                  NULL,        // $InUseOptions -- take default
                  NULL         // $WaitTime -- take default
                  )

        // Should handle failure and other errors here.

        return TRUE;
    }
```

5.6.3 Removal of an Exported FileShare

```
// DESCRIPTION
//   GOAL: UnExport an exported NFS or CIFS FileShare.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The file share already exists and has been surfaced through
//      SMI-S.
//   2. The file share has been defined with an associated
//      ExportedFileShareSettings element and hosted on an
//      surfaced file server ComputerSystem.
//   3. There is a FileExportService element hosted by
//      the same file server ComputerSystem that provides
//      this method.
//
// FUNCTION UnExportFileSystemShare
//   This function removes an NFS or CIFS file share that is
//      hosted by the same ComputerSystem as the host of the
//      service.
// INPUT Parameters:
```

```
//   fssh: A reference to the newly created FileShare element
//   force: Whether the method should force all clients of the
//      file share to be disconnected.
//   waittime: The time in seconds to wait before implementing the
//      specified force option (default 300 seconds).
//   notification: A string used to notify clients that the file
//      share is going to be unavailable.  This is included in
//      the alert indication sent to clients that subscribe for
//      them (but... shouldn't this go to operational clients?)
// OUTPUT Parameters:
//   job: A reference to a ConcreteJob that is executing a long-term job.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub UnExportFileSystemShare(IN REF CIM_FileShare $fssh,
                            IN uint16 $force,
                            IN uint32 $waittime,
                            IN String $notification,
                            OUT REF CIM_Job $job);
{
    //
    // If waittime > 0, set force to 2 to distinguish between
    // a force with no wait and a force with wait
    // -- see the specification of ReleaseExportedShare.
    //
    if ($force > 0 && $waittime > 0) {
        $force = 2;
    }
    //
    // clients of the share may have registered for an indication
    // when a share is disconnected
    //
    <send indication -- see indications recipes>

    // Get the File Server
    //
//    &GetFileExportServer($fs, $server);
    //
    // Get the ComputerSystem for the filesystem (via HostedFileSystem association)
    //  There should be exactly one.
    $server = Associators($fssh,
                          "CIM_HostedFileShare",
                          "CIM_ComputerSystem",
                          "PartComponent",
                          "GroupComponent")->[0];
```

```
        //
        // Get a FileExportService via the HostedService association to
        // the file server ComputerSystem
        //
        $feservice = Associators($server,
                                 "CIM_HostedService",
                                 "SNIA_FileExportService",
                                 "Antecedent",
                                 "Dependent")->[0];

        //
        // Call ReleaseExportedShare() with the $force and $waittime values
        // which tell the share to wait for the specified time
        // if there are any clients still connected.
        //

        $feservice.ReleaseExportedShare($job, $fssh, $force, $waittime);

        // Should handle failure and other errors here.

        return TRUE;
    }
```

EXPERIMENTAL

5.6.4 File Export Manipulation Supported Capabilities Patterns

Table 19 lists the capabilities patterns that are formally recognized by SMI-S 1.2.0 for determining capabilities of various implementations:

Table 19 - SMI-S File Export Supported Capabilities Patterns

FileSharingProtocols	SynchronousMethods	AsynchronousMethods	InitialExportState
NFS, CIFS	Export Creation, Export Modification, Export Deletion	Null	*
NFS, CIFS	Null	Export Creation, Export Modification, Export Deletion	*
NFS, CIFS	Null	Null	Null

Note: Asterisk (*) means any state is valid.

5.7 Registered Name and Version

File Export Manipulation version 1.3.0

5.8 CIM Elements

Table 20 describes the CIM elements for File Export Manipulation.

Table 20 - CIM Elements for File Export Manipulation

Element Name	Requirement	Description
5.8.1 CIM_ConcreteDependency	Optional	Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based. This is provided for backward compatibility with previous releases of SMI-S.
5.8.2 CIM_ElementCapabilities (FES Configuration)	Mandatory	Associates the File Export Service to the FileExportCapabilities element that describes the service capabilities.
5.8.3 CIM_ElementSettingData (FileShare Setting)	Mandatory	Associates a FileShare and ExportedFileShareSetting elements.
5.8.4 CIM_FileStorage (Subelement)	Conditional	Conditional requirement: Required if parent profile is NAS Head. or Required if parent profile is a Self-contained NAS System.. Represents that a file or directory that is made available for export is contained by a LocalFileSystem specified as a dangling reference.
5.8.5 CIM_HostedService	Mandatory	Associates the File Export Service to the hosting File Server Computer System.
5.8.6 CIM_LogicalFile (Subelement)	Conditional	Conditional requirement: Required if parent profile is NAS Head. or Required if parent profile is a Self-contained NAS System.. A LogicalFile (or Directory subclass) that is a sub-element of a LocalFileSystem that is made available for export via a fileshare hosted on a ComputerSystem. This is included for backward compatibility with previous releases of SMI-S.
5.8.7 CIM_SAPAvailableForElement	Mandatory	Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
5.8.8 CIM_ServiceAffectsElement	Mandatory	Associates the File Export Service to the elements that the service manages (such as a FileShare configured for exporting a LogicalFile).

Table 20 - CIM Elements for File Export Manipulation

Element Name	Requirement	Description
5.8.9 SNIA_ElementCapabilities (FES Capabilities)	Mandatory	Associates the File Export Service to at least one ExportedFileShareCapabilities element that indicates that support is available for managing an exported FileShare for at least one of the file sharing protocols recognized by this profile. These include: "NFS"/2, "CIFS"/3, "DAFS"/4, "WebDAV"/5, "HTTP"/6, or "FTP"/7.
5.8.10 SNIA_ExportedFileShareCapabilities (FES Capabilities)	Mandatory	This element represents the Capabilities of the File Export Service for managing FileShares of a specific file sharing protocol (and version). The file sharing protocol is specified by the FileSharingProtocol and ProtocolVersions properties.
5.8.11 SNIA_ExportedFileShareSetting (FileShare Setting)	Mandatory	The configuration settings for an Exported FileShare; i.e., a setting for a FileShare available for exporting. This setting may have been created or modified by the extrinsic methods of this profile. Note that CIFS allows in-band creation, modification, or deletion of FileShares; also, some systems might define preexistent FileShares. All of these will be surfaced.
5.8.12 SNIA_ExportedFileShareSetting (Pre-defined)	Optional	This element represents a predefined configuration settings for exported FileShares that is used to define a Capabilities element associated with the FileExportService.
5.8.13 SNIA_FileExportCapabilities (FES Configuration)	Mandatory	This element represents the management capabilities of the File Export Service.
5.8.14 SNIA_FileExportService	Mandatory	The File Export Service provides the methods to create and export file elements as shares.
5.8.15 SNIA_FileShare (Exported File Share)	Mandatory	Represents the sharing characteristics of a particular file element.
5.8.16 SNIA_HostedShare	Mandatory	Represents that a shared element is hosted by a ComputerSystem.
5.8.17 SNIA_SettingsDefineCapabilities (Pre-defined)	Optional	Represents the association between a ExportedFileShareCapabilities and a predefined ExportedFileShareSetting element that specifies what the Capabilities can support.
5.8.18 SNIA_SharedElement	Mandatory	Associates a FileShare to the LocalFileSystem on which it is based.

Table 20 - CIM Elements for File Export Manipulation

Element Name	Requirement	Description
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA SNIA_FileShare	Mandatory	Creation of an exported file share. This indication returns the newly created FileShare.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA SNIA_FileShare	Mandatory	Deletion of an exported file share. This indication returns the model path to the deleted file share and its unique instance id. (Question: Should this return the pathname of the shared directory as well?) Note that a model path is like a CIM object path but not exactly the same.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_FileShare AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of state of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the subprofile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_FileShare AND SourceInstance.SNIA_FileShare::OperationalStatus <> PreviousInstance.SNIA_FileShare::OperationalStatus	Optional	CQL -Change of state of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the subprofile.

5.8.1 CIM_ConcreteDependency

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Optional

Table 21 describes class CIM_ConcreteDependency.

Table 21 - SMI Referenced Properties/Methods for CIM_ConcreteDependency

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The LogicalFile that is being shared.
Dependent		Mandatory	The Share that represents the LogicalFile being shared.

5.8.2 CIM_ElementCapabilities (FES Configuration)

Created By: Static

Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 22 describes class CIM_ElementCapabilities (FES Configuration).

Table 22 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (FES Configuration)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The FileExportCapabilities.
ManagedElement		Mandatory	The FileExportService.

5.8.3 CIM_ElementSettingData (FileShare Setting)

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 23 describes class CIM_ElementSettingData (FileShare Setting).

Table 23 - SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare Setting)

Properties	Flags	Requirement	Description & Notes
IsCurrent	N	Optional	Is always true in this version of the subprofile because we only support one setting per share. However support for the other flags, specifically, IsDefault and IsNext, could be added in future releases.
IsDefault	N	Optional	Not Specified in this version of the Profile
IsNext	N	Optional	Not Specified in this version of the Profile
IsMinimum	N	Optional	Not Specified in this version of the Profile
IsMaximum	N	Optional	Not Specified in this version of the Profile
ManagedElement		Mandatory	The FileShare used for exporting an element.
SettingData		Mandatory	A Setting that specifies possible configurations of the FileShare. In this version, we default this to isCurrent="true"

5.8.4 CIM_FileStorage (Subelement)

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Required if parent profile is NAS Head. or Required if parent profile is a Self-contained NAS System..

Table 24 describes class CIM_FileStorage (Subelement).

Table 24 - SMI Referenced Properties/Methods for CIM_FileStorage (Subelement)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	The file or directory that is made available for export.
GroupComponent		Mandatory	The Local File System that contains the exported File or Directory.

5.8.5 CIM_HostedService

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 25 describes class CIM_HostedService.

Table 25 - SMI Referenced Properties/Methods for CIM_HostedService

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting Computer System.
Dependent		Mandatory	The FileExportService

5.8.6 CIM_LogicalFile (Subelement)

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Required if parent profile is NAS Head. or Required if parent profile is a Self-contained NAS System..

Table 26 describes class CIM_LogicalFile (Subelement).

Table 26 - SMI Referenced Properties/Methods for CIM_LogicalFile (Subelement)

Properties	Flags	Requirement	Description & Notes
CSCreationClassName		Mandatory	CIM Class of the Computer System that hosts the Filesystem of this File.
CSName		Mandatory	Name of the Computer System that hosts the Filesystem of this File.
FSCreationClassName		Mandatory	CIM Class of the LocalFileSystem on the Computer System that contains this File.

Table 26 - SMI Referenced Properties/Methods for CIM_LogicalFile (Subelement)

Properties	Flags	Requirement	Description & Notes
FSName		Mandatory	Name of the LocalFileSystem on the Computer System that contains this File.
CreationClassName		Mandatory	CIM Class of this instance of LogicalFile.
Name		Mandatory	Name of this LogicalFile.

5.8.7 CIM_SAPAvailableForElement

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 27 describes class CIM_SAPAvailableForElement.

Table 27 - SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The element that is made available through a SAP. In the File Export subprofile, these are FileShares configured for either export.
AvailableSAP		Mandatory	The ProtocolEndpoint that is available to this FileShare. This shall be 4200 (NFS) or 4201 (CIFS).

5.8.8 CIM_ServiceAffectsElement

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 28 describes class CIM_ServiceAffectsElement.

Table 28 - SMI Referenced Properties/Methods for CIM_ServiceAffectsElement

Properties	Flags	Requirement	Description & Notes
ElementEffects		Mandatory	In this profile, the service provides management for the element. We allow Other to support vendor extensions. The standard values are 1 (Other) and 5 (Manages).
OtherElementEffects Descriptions		Mandatory	A description of other element effects that this association might be exposing.

Table 28 - SMI Referenced Properties/Methods for CIM_ServiceAffectsElement

Properties	Flags	Requirement	Description & Notes
AffectedElement		Mandatory	The FileShare.
AffectingElement		Mandatory	The FileExportService.

5.8.9 SNIA_ElementCapabilities (FES Capabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 29 describes class SNIA_ElementCapabilities (FES Capabilities).

Table 29 - SMI Referenced Properties/Methods for SNIA_ElementCapabilities (FES Capabilities)

Properties	Flags	Requirement	Description & Notes
Characteristics		Mandatory	If this array enum includes the value mapped to "Default", it indicates that the ExportedFileShareCapabilities element identified by this association is the default to be used for any extrinsic method of the associated FileExportService element.
Capabilities		Mandatory	The FileExportCapabilities. The FileSharingProtocol in these capabilities shall be 2 (NFS), 3 (CIFS), 4 (DAFS), 5 (WebDAV), 6 (HTTP) or 7 (FTP).
ManagedElement		Mandatory	The FileExportService.

5.8.10 SNIA_ExportedFileShareCapabilities (FES Capabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 30 describes class SNIA_ExportedFileShareCapabilities (FES Capabilities).

Table 30 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareCapabilities (FES Capabilities)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for a capability of a File Export Service
ElementName		Mandatory	A provider supplied user-Friendly Name for this Capabilities element.
FileSharingProtocol		Mandatory	This identifies the single file sharing protocol (e.g., NFS or CIFS) that this Capabilities represents.
ProtocolVersions		Optional	An array listing the versions of the protocol specified by the FileSharingProtocol property. A NULL value or entry indicates support for all versions of this protocol. At this point there is no standard mechanism for naming versions of CIFS or NFS, so this is being made optional. If the property is NULL, all versions of the protocol are supported.
SupportedProperties		Mandatory	This is the list of configuration properties (of ExportedFileShareSetting) that are supported for specification at creation time by this Capabilities element. Properties that can appear in this array are: "DefaultReadWrite" ("2"), "DefaultExecute" ("3"), "DefaultUserId" ("4"), "RootAccess" ("5"), "WritePolicy" ("6"), "AccessPoints" ("7"), and "InitialEnabledState" ("8").
CreateGoalSettings()		Mandatory	This extrinsic method supports the creation of a ExportedFileShareSetting that is a supported variant of a ExportedFileShareSetting passed in as an embedded IN parameter TemplateGoalSettings[0]. The method returns the supported ExportedFileShareSetting as an embedded OUT parameter SupportedGoalSettings[0].

5.8.11 SNIA_ExportedFileShareSetting (FileShare Setting)

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 31 describes class SNIA_ExportedFileShareSetting (FileShare Setting).

Table 31 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (FileShare Setting)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique ID for the Setting.
ElementName		Mandatory	A client-defined user-friendly name for the Setting.
FileSharingProtocol		Optional	The file sharing protocol supported by this share. NFS (2) and CIFS (3) are the supported values.
ProtocolVersions		Mandatory	An array of the versions of the supported file sharing protocol. A share may support multiple versions of the same protocol. A NULL value or a NULL entry indicates support for all versions. At this point there is no standard mechanism for naming versions of CIFS or NFS, so this is being made optional.
InitialEnabledState	N	Optional	This indicates the enabled/disabled states initially set for a created FileShare by this Setting element. Valid values are "1" ("Other"), "2" ("Enabled"), "3" ("Disabled"), "7" ("In Test"), "8" ("Deferred") or "9" ("Quiesce") Note: We need to rethink the usage of this property once the file share has been created. Maybe it should apply to when the file share is re-activated when the share or system is rebooted after a shutdown. With the current definition, neither this nor OtherEnabledState make sense.
OtherEnabledState	N	Optional	This should be filled in if the InitialEnabledState is "1" ("Other").
DefaultUserIdSupported	N	Optional	Indicates whether the associated FileShare will use a default user id to control access to the share if the id of the importing client is not provided. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Valid values are "2" ("No Default User Id"), "3" ("System-Specified Default User Id") or "4" ("Share-Specified Default User Id").
RootAccess	N	Optional	Indicates whether the associated FileShare will support default access privileges to administrative users from specified hosts. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Valid values are "2" ("No Root Access") or "3" ("Allow Root Access").

Table 31 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (FileShare Setting)

Properties	Flags	Requirement	Description & Notes
AccessPoints	N	Optional	An enumerated value that specifies the service access points that are available to this FileShare element by default (to be used by clients for connections). Any ServiceAccessPoint elements that actually connect to this FileShare element will be associated to it by a SAPAvailableForElement association. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. The default or built-in access points can always be overridden by the privileges explicitly defined through the Authorization subprofile. Valid values are "2" ("None"), "3" ("Service Default"), "4" ("All") or "5" ("Named Points").
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
DefaultReadWrite	N	Optional	Not Specified in this version of the Profile
DefaultExecute	N	Optional	Not Specified in this version of the Profile
ExecuteSupport	N	Optional	Not Specified in this version of the Profile
WritePolicy	N	Optional	Not Specified in this version of the Profile

5.8.12 SNIA_ExportedFileShareSetting (Pre-defined)

Created By: Static_or_External
Modified By: External
Deleted By: External
Requirement: Optional

Table 32 describes class SNIA_ExportedFileShareSetting (Pre-defined).

Table 32 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (Pre-defined)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for this Setting element.
ElementName		Mandatory	A provider supplied user-friendly name for this Setting element.
FileSharingProtocol		Mandatory	The file sharing protocol to which this Setting element applies. The entries in the ProtocolVersions property identify the specific versions of the protocol that are supported. This profile only supports "NFS" (2) and "CIFS" (3).

Table 32 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (Pre-defined)

Properties	Flags	Requirement	Description & Notes
ProtocolVersions		Optional	This array identifies the versions of the file sharing protocol (specified by FileSharingProtocol) to which this Setting element applies. If NULL, it indicates support for all versions. At this point there is no standard mechanism for naming versions of CIFS or NFS, so this is being made optional.
InitialEnabledState		Optional	This indicates the enabled/disabled states initially set for a created FileShare by this Setting element. Valid values are "1" ("Other"), "2" ("Enabled"), "3" ("Disabled"), "7" ("In Test"), "8" ("Deferred") or "9" ("Quiesce")
OtherEnabledState		Optional	A vendor-specific description of the initial enabled state of a created fileshare if InitialEnabledState=1("Other").
DefaultUserIdSupported		Optional	Indicates whether a FileShare created or modified by using this Setting element will use a default user id to control access to the share if the id of the importing client is not provided. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Valid values are "2" ("No Default User Id"), "3" ("System-Specified Default User Id") or "4" ("Share-Specified Default User Id").
RootAccess		Optional	Indicates whether a FileShare created or modified by using this Setting element will support default access privileges to administrative users from specific hosts specified at creation time. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Valid values are "2" ("No Root Access") or "3" ("Allow Root Access").
AccessPoints		Optional	An enumerated value that specifies the service access points that are available to a FileShare created or modified by using this Setting element by default (to be used by clients for connections). These default access points can always be overridden by the privileges explicitly defined by a supported authorization mechanism(s). Any ServiceAccessPoints that actually connect to this share will be associated to it by CIM_SAPAvailableForElement. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Valid values are "2" ("None"), "3" ("Service Default"), "4" ("All") or "5" ("Named Points").

Table 32 - SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (Pre-defined)

Properties	Flags	Requirement	Description & Notes
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
DefaultReadWrite		Optional	Indicates the default privileges that are supported for read and write authorization when creating or modifying a FileShare using this Setting element. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Not Specified in this version of the Profile
DefaultExecute		Optional	Indicates the default privileges that are supported for execute authorization when creating or modifying a FileShare using this Setting element. Note: The resulting access privileges shall be surfaced using the Authorization subprofile. Not Specified in this version of the Profile
ExecuteSupport		Optional	Indicates if the sharing mechanism provides specialized support for executing a shared element when creating or modifying a FileShare using this Setting element (for instance, does it provide paging support for text pages). Not Specified in this version of the Profile
WritePolicy		Optional	Indicates whether writes through a FileShare (created or modified by using this Setting element) to the shared element will be handled synchronously or asynchronously by default. This policy may be overridden or surfaced using the Policy subprofile. Not Specified in this version of the Profile

5.8.13 SNIA_FileExportCapabilities (FES Configuration)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 33 describes class SNIA_FileExportCapabilities (FES Configuration).

Table 33 - SMI Referenced Properties/Methods for SNIA_FileExportCapabilities (FES Configuration)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the capabilities of a File Export Service.
ElementName		Mandatory	A provider supplied user-friendly name for this Capabilities element.
FileSharingProtocol		Mandatory	An array listing all the protocols for file sharing supported by the FileExportService represented by this FileExportCapabilities element. Duplicate entries are permitted because the corresponding entry in the ProtocolVersions array property indicates the supported version of the protocol. Each entry must correspond to an ExportedFileShareCapabilities element associated via ElementCapabilities to the FileExportService -- the FileSharingProtocol and ProtocolVersion properties of that element must match the entry.
ProtocolVersions		Optional	An array listing all the versions of the file sharing protocol specified in the corresponding entry of the FileSharingProtocol array property. A NULL entry indicates support for all versions of the protocol. At this point there is no standard mechanism for naming versions of CIFS or NFS, so this property is optional in this subprofile.
SupportedSynchronousMethods	N	Mandatory	An array listing the extrinsic methods of the FileExportService that can be called synchronously. Note: Every supported method shall be listed either in this property or in the SupportedAsynchronousMethods array property.
SupportedAsynchronousMethods	N	Mandatory	An array listing the extrinsic methods of the FileExportService that can be called synchronously. Note: Every supported method shall be listed either in this property or in the SupportedSynchronousMethods array property.
InitialEnabledState		Optional	This represents the state of initialization of a FileShare on initial creation.

5.8.14 SNIA_FileExportService

Created By: Static
Modified By: Static
Deleted By: Static

Requirement: Mandatory

Table 34 describes class SNIA_FileExportService.

Table 34 - SMI Referenced Properties/Methods for SNIA_FileExportService

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A provider supplied user-friendly name for this Service.
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Service.
SystemName		Mandatory	The name of the Computer System hosting the Service.
CreationClassName		Mandatory	The CIM Class name of the Service.
Name		Mandatory	The unique name of the Service.
CreateExportedShare()		Mandatory	Create a FileShare element configured for exporting a file or directory as a share.
ModifyExportedShare()		Mandatory	Modify the configuration of a FileShare element setup to export a file or directory as a share.
ReleaseExportedShare()		Mandatory	Delete the FileShare element that is exporting a file or directory as a share, thus releasing that element.

5.8.15 SNIA_FileShare (Exported File Share)

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 35 describes class SNIA_FileShare (Exported File Share).

Table 35 - SMI Referenced Properties/Methods for SNIA_FileShare (Exported File Share)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	A unique id for the FileShare element.
ElementName		Mandatory	This shall be a user friendly name for the FileShare.
Name		Mandatory	This shall be an opaque string that uniquely identifies the path to the directory or file.
SharingDirectory		Mandatory	Indicates if the shared element is a file or a directory. This is useful when importing but less so when exporting.
OperationalStatus		Mandatory	The OperationalStatus of the FileShare as defined in the Health and Fault Management Clause.
Description	N	Optional	This a comment describing the file share.
Caption	N	Optional	Not Specified in this version of the Profile

Table 35 - SMI Referenced Properties/Methods for SNIA_FileShare (Exported File Share)

Properties	Flags	Requirement	Description & Notes
InstallDate	N	Optional	Not Specified in this version of the Profile
StatusDescriptions	N	Optional	Not Specified in this version of the Profile
HealthState	N	Optional	Not Specified in this version of the Profile
EnabledState	N	Optional	Not Specified in this version of the Profile
OtherEnabledState	N	Optional	Not Specified in this version of the Profile
RequestedState	N	Optional	Not Specified in this version of the Profile
EnabledDefault	N	Optional	Not Specified in this version of the Profile
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile
RequestStateChange()		Optional	Not Specified in this version of the Profile

5.8.16 SNIA_HostedShare

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 36 describes class SNIA_HostedShare.

Table 36 - SMI Referenced Properties/Methods for SNIA_HostedShare

Properties	Flags	Requirement	Description & Notes
RemoteShareWWN	N	Optional	Not Specified in this version of the Profile
Dependent		Mandatory	The Share that is hosted by a Computer System
Antecedent		Mandatory	The Computer System that hosts a FileShare. It may be any system, but the system shall have Dedicated=16 (File Server)

5.8.17 SNIA_SettingsDefineCapabilities (Pre-defined)

Created By: Static_or_External
Modified By: External
Deleted By: External
Requirement: Optional

Table 37 describes class SNIA_SettingsDefineCapabilities (Pre-defined).

Table 37 - SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Pre-defined)

Properties	Flags	Requirement	Description & Notes
PropertyPolicy		Mandatory	
ValueRole		Mandatory	
ValueRange		Mandatory	
GroupComponent		Mandatory	An Exported FileShare Capabilities element that is defined by a collection of ExportedFileShareSetting Settings.
PartComponent		Mandatory	A Exported FileShare Setting that provides a point or a partial definition for a Exported FileShare Capabilities element.

5.8.18 SNIA_SharedElement

Created By: Extrinsic: CreateExportedShare
Modified By: Extrinsic: ModifyExportedShare
Deleted By: Extrinsic: ReleaseExportedShare
Requirement: Mandatory

Table 38 describes class SNIA_SharedElement.

Table 38 - SMI Referenced Properties/Methods for SNIA_SharedElement

Properties	Flags	Requirement	Description & Notes
SystemElement		Mandatory	The LocalFileSystem that is sharing a directory or file through a SNIA_FileShare alter ego.
SameElement		Mandatory	The FileShare that is the alter ego for a directory or file in a LocalFileSystem.

EXPERIMENTAL

EXPERIMENTAL

Clause 6: File Server Manipulation Subprofile

6.1 Synopsis

Profile Name: File Server Manipulation

Version: 1.3.0

Organization: SNIA

CIM schema version: 2.15

Central Class: FileServerConfigurationService

Scoping Class: ComputerSystem

6.2 Description

6.2.1 Overview

The File Server Manipulation Subprofile is a subprofile of autonomous profiles that support filesystems. It makes use of elements of the Filesystem subprofiles and supports creation, modification and deletion of FileServers. A number of other profiles and subprofiles also make use of elements of the Filesystem subprofile and will be referred to in this specification as “filesystem related profiles” -- these include but are not limited to the Filesystem subprofile, the Filesystem Manipulation subprofile, the File Export subprofile, the NAS Head profile, the Self-Contained NAS profile, and so on.

In this release of SMI-S, the autonomous profiles that use the file server Manipulation Subprofile are the NAS Head and Self-Contained NAS profiles.

This profile models a file server as a virtualized system with virtual ports. The model doesn’t address the physical ports involved. Throughout this subprofile, the term file server will be synonymous for “ComputerSystem with Dedicated[]=”FileServer”.

The profile models a file server from a “read only” perspective and a “configuration” perspective. The read only perspective defines the objects and attributes that describe a file server instance. The configuration perspective defines the permitted actions on the file server for creating, deleting, and modifying instances. By providing these two perspectives, this profile takes the place of having two separate profiles.

6.2.2 Instance Diagrams

6.2.2.1 File Server classes and associations (read-only view)

Figure illustrates the constructs that are involved in defining a file server. This summarizes the “read only” view of the classes and associations for this subprofile.)

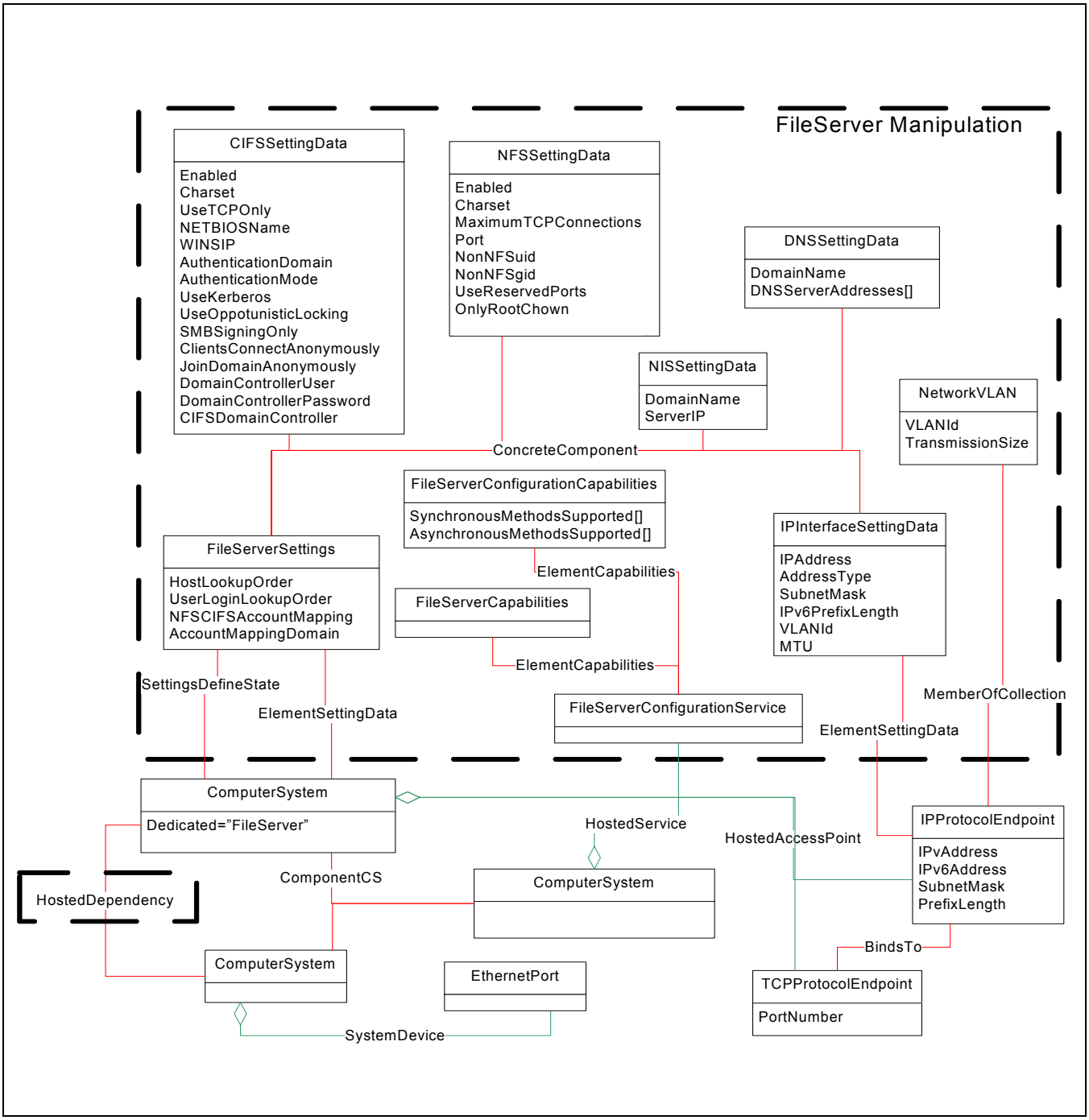

Figure 8 - File Server Classes and Associations (Read-only view)

The file server is modeled as a ComputerSystem whose Dedicated property is set to “FileServer”. A file server may be a ComponentCS of another computersystem such as a NAS Head or Self-Contained NAS for example. This

top level ComputerSystem has a HostedService association with FileServerConfigurationService, which provides the set of extrinsics for manipulating a file server.

A file server is hosted on a ComputerSystem. This may be a physical control unit or some other hardware system that has the EthernetPort through which the file server will serve files via CIFS and/or NFS. The HostedDependency association is used to relate the file server with the hosting ComputerSystem.

FileServerSettings captures the settings of the file server. It has ConcreteComponent associations with other setting data that capture the file server's settings for CIFS, NFS, NIS, DNS, and its IP Interface. A file server can be created with the minimum of IPInterfaceSettingData being specified. But it cannot serve FileShare instances in this state and will need to have either its CIFSSettingData or NFSSettingData specified in either the creation or modification extrinsic methods. If neither CIFSettingData nor NFSSettingData are specified at creation time, instances of both shall be created by the provider and their "Enabled" property will be set to "false".

The file server has two separate associations with FileServerSettings. SettingsDefineState is used to represent the current state of the file server's setting data while ElementSettingData is used to capture the setting data used to initially create or modify the file server.

When a file server is created or when it has additional IPInterfaces associated with it, an instance of NetworkVLAN may be created if VLAN tagging should be associated with the IPInterface. NetworkVLAN instances are associated with the specific IPProtocolEndpoint to capture the VLAN tag to be used when doing I/O on that IP interface. The properties VLANid and MTU in IPInterfaceSettingData specify the values to use when creating the NetworkVLAN instance.

The NISSettingData and DNSSettingData if present are used to resolve hosts and user names when authenticating hosts and users.

6.2.2.2 File Server Configuration classes and associations

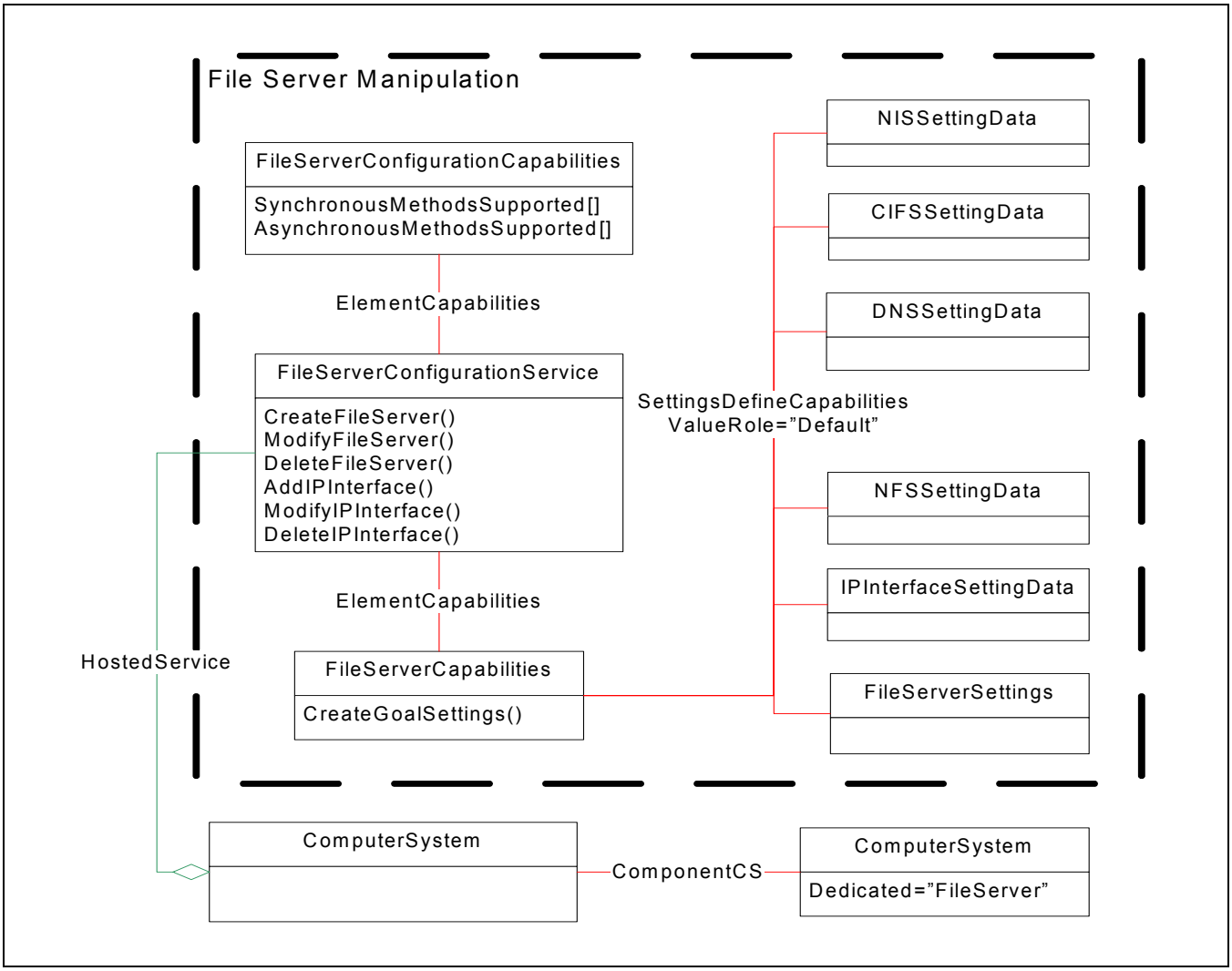

Figure 9 - File Server Configuration classes and association

Figure 9 illustrates the constructs that are involved in configuring a file server.

The top level ComputerSystem has a HostedService association with FileServerConfigurationService that defines the extrinsics that can be used to manage a file server. There are 3 methods for managing a file server and 3 methods for managing additional IPInterfaces for a given file server.

FileServerConfigurationCapabilities lists the extrinsics that can be called synchronously or asychronously. It is assocated with the FileServerConfigurationService via the ElementCapabilities assocation.

FileServerCapabilities provides one method CreateGoalSettings that can be used to arrive at a set of viable SettingData instances that can be used for creating or modifying a file server. It also is associated with FileServerConfigurationService via ElementCapabilities. It may have associations with SettingData instances that reflect the Default settings for the file server. The SettingsDefineCapabilities association (with ValueRole="Default") is used to capture these default SettingData instances.

6.2.3 Health and Fault Management Consideration

6.2.3.1 OperationalStatus for File Server ComputerSystem

A file server's operational status will be influenced by the operational status of the ComputerSystem that is hosting it via HostedDependency. For example, if the hosting ComputerSystem is "Stopped", then the status of the file server will be "Stopped". Providers must take this into account when formulating the status of the file server.

Table 39 - Operational Status for File Server ComputerSystem

Primary OperationalStatus	Description
2 "OK"	The File Server is running with good status
3 "Degraded"	The File Server is operating in a degraded mode. This could be due to the health state of some component of the ComputerSystem, due to load by other applications, or due to the health state of backend or front-end network interfaces.
4 "Stressed"	The File Server resources are stressed
5 "Predictive Failure"	The File Server might fail because some resource or component is predicted to fail
6 "Error"	An error has occurred causing the File Server to become unavailable. Operator intervention through SMI-S to restore the service may be possible.
6 "Error"	An error has occurred causing the File Server to become unavailable. Automated recovery may be in progress.
7 "Non-recoverable Error"	The File Server is not functioning. Operator intervention through SMI-S will not fix the problem.
8 "Starting"	The File Server is in process of initialization and is not yet available operationally.
9 "Stopping"	The File Server is in process of stopping, and is not available operationally.
10 "Stopped"	The File Server cannot be accessed operationally because it is stopped -- if this did not happened because of operator intervention or happened in real-time, the OperationalStatus would have been "Lost Communication" rather than "Stopped".
11 "In Service"	The File Server is offline in maintenance mode, and is not available operationally.
13 "Lost Communications"	The File Server cannot be accessed operationally -- if this happened because of operator intervention it would have been "Stopped" rather than "Lost Communication".
14 "Aborted"	The File Server is stopped but in a manner that may have left it in an inconsistent state.

Table 39 - Operational Status for File Server ComputerSystem

Primary OperationalStatus	Description
15 “Dormant”	The File Server is offline; and the reason for not being accessible is unknown.
16 “Supporting Entity in Error”	The File Server is in an error state, or may be OK but not accessible, because a supporting entity is not accessible.

6.2.4 Cascading Considerations

Not Applicable.

6.3 Supported Profiles, Subprofiles, and Packages

Table 40 describes the supported profiles for File Server Manipulation.

Table 40 - Supported Profiles for File Server Manipulation

Registered Profile Names	Mandatory	Version
Job Control	No	1.3.0

6.4 Methods of the Profile

6.4.1 Extrinsic Methods of the Profile

Table 41 - File Server Manipulation Methods

Method	Created Instances	Deleted Instances	Modified Instances
CreateFileServer	FileServerSettings NISSettingData DNSSettingData CIFSSettingData NFSSettingData IPInterfaceSettingData IPProtocolEndpoint NetworkVLAN	N/A	N/A
ModifyFileServer	N/A	N/A	FileServerSettings NISSettingData DNSSettingData CIFSSettingData NFSSettingData
DeleteFileServer	N/A	FileServerSettings NISSettingData DNSSettingData CIFSSettingData NFSSettingData IPInterfaceSettingData IPProtocolEndpoint NetworkVLA	N/A
CreateGoalSettings	N/A	N/A	N/A
AddIPinterface	IPInterfaceSettingData IPProtocolEndPoint NetworkVLAN	N/A	
ModifyIPInterface	N/A	N/A	IPInterfaceSettingData IPProtocolEndPoint NetworkVLAN
DeleteIPInterface	N/A	IPInterfaceSettingData IPProtocolEndPoint NetworkVLAN	N/A
CreateGoalSettings			

6.4.1.1 FileSystemCapabilities.CreateGoalSettings

This extrinsic method of the FileSystemCapabilities class validates support for a caller-proposed Settings.

The client shall pass six array elements in the TemplateGoalSettings parameter and six array elements in the SupportedGoalSettings parameter. Each array element represents a congifurable aspect of a FileServer. A given array element in index “y” in TemplateGoalSettings will be of the same class/type as that in array element in index “y” in SupportedGoalSettings. As each array element in both parameters takes an EmbeddedInstance, this implies that they do not exist in the provider’s implementation but are the responsibility of the client to create and manage.

Any or all of the TemplateGoalSetting array elements may be the empty string to represent a NULL entry. This method will return a default CIM_Settings subclass object in SupportedGoalSettings corresponding to each TemplateGoalSettings array element that is an empty string.

If any of the TemplateGoalSettings array elements specify values that cannot be supported, this method shall return an appropriate error and should return a best match in the corresponding SupportedGoalSettings array element.

When providing EmbeddedInstances as input for any of the SupportedGoalSettings array elements, the instance should specify a previously returned CIM_Setting that the implementation could support. On output, this same array element specifies a new CIM_Setting that the implementation can support. If the output array element is identical to the input array element, both client and implementation may conclude that this is the best match for that particular SupportedGoalSettings array element. If the output array elements do not match the corresponding TemplateGoalSettings array elements and if any of the input SupportedGoalSettings array elements do not match the output array elements provided in SupportedGoalSettings, then the method must return "Alternative Proposed". If any of the output array elements are empty strings (representing the fact that no valid CIM_Setting could be found), the method must return an “Failed”.

The client and the implementation can engage in a negotiation process that may require iterative calls to this method. As stated above, to assist the implementation in tracking the progress of the negotiation, the client may pass previously returned values of SupportedGoalSettings array elements as new input values of SupportedGoalSettings. The implementation may determine that a step has not resulted in progress if the input and output values of any SupportedGoalSettings array elements are the same. A client may infer from the same result that the TemplateGoalSettings array element(s) must be modified.

The array elements in TemplateGoalSettings and SupportedGoalSettings shall have the following index - EmbeddedInstance mappings:

Table 42 - Array Element Mappings for TemplateGoalSettings and SupportedGoalSettings

Array Indice	EmbeddedInstance
0	SNIA_FileServerSettings
1	SNIA_IPInterfaceSettingData
2	SNIA_CIFSSettingData
3	SNIA_NFSSettingData
4	SNIA_NISSettingData
5	SNIA_DNSSettingData

Table 43 shows the details of the method signature and return results.

Table 43 - Parameters for Extrinsic Method FileServerCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	This contains an array of 6 elements, each of which being an EmbeddedInstance of a CIM_Setting subclass. Each of the array elements shall contain either an empty string to represent a “NULL” entry, or shall contain an EmbeddedInstance. Each array element contains a specific CIM_Setting subclass as follows: 0: EmbeddedInstance ("SNIA_FileServerSettings") 1: EmbeddedInstance ("SNIA_IPInterfaceSettingData") 2: EmbeddedInstance ("SNIA_CIFSSettingData") 3: EmbeddedInstance ("SNIA_NFSSettingData") 4: EmbeddedInstance ("SNIA_NISSettingData") 5: EmbeddedInstance ("SNIA_DNSSettingData")
SupportedGoalSettings[]	INOUT	string	This contains an array of 6 elements, each of which being an EmbeddedInstance of a CIM_Setting subclass. On input, each of the array elements shall contain an either an empty string to represent a “NULL” entry, or shall contain an EmbeddedInstance. If it contains an EmbeddedInstance, then this instance specifies a previously returned CIM_Setting that the implementation could support. On output, it specifies a new CIM_Setting that the implementation can support. Each array element contains a specific CIM_Setting subclass as follows: 0: EmbeddedInstance ("SNIA_FileServerSettings") 1: EmbeddedInstance ("SNIA_IPInterfaceSettingData") 2: EmbeddedInstance ("SNIA_CIFSSettingData") 3: EmbeddedInstance ("SNIA_NFSSettingData") 4: EmbeddedInstance ("SNIA_NISSettingData") 5: EmbeddedInstance ("SNIA_DNSSettingData")
Normal Return			

Table 43 - Parameters for Extrinsic Method FileServerCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
Status		uint32	ValueMap{}, Values{} "Success", “Not Supported”, “Unknown”, "Failed", "Timeout", “Invalid Parameter”, "Alternative Proposed"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

6.4.1.2 Signature and Parameters of FileServerConfigurationService.CreateFileServer

This extrinsic creates a new FileServer. The method takes several “goal” parameters that represent different configurable aspects of the FileServer. Each of these parameters can be NULL, an empty string, or will contain an EmbeddedInstance.

If a given parameter is NULL or an empty string, a default instance will be selected by the provider using the corresponding element associated to the FileServerConfigurationService by the DefaultElementCapabilities association. This element that is used will be returned in the parameter.

When creating a new FileServer, the client can decide to what degree the new FileServer will be configured by providing the parameters of those aspects that should be configured. For example, to create a FileServer with a minimum configuration, the client could provide just the ElementName. The newly created FileServer will take on the configuration defaults as specified by the elements associated with FileServerService via the SettingsDefineCapabilities association (with ValueRole=”Default”). Later, the client may modify any of these default settings via the ModifyFileServer and ModifyIPInterface methods.

When creating a new FileServer, the client may associate a single IP Interface with the FileServer. If a client wishes to associate more than one IP Interface with the FileServer, the AddIPInterface method should be used. It allows the client to specify the additional IP information, Hosting ComputerSystem, and EthernetPort for the new IP Interface.

A client may change an existing IP Interface by using the ModifyIPInterface method. It allows the client to modify the IP Interface, Hosting ComputerSystem, and/or EtheretPort.

Table 44 - Parameters for Extrinsic Method FileServerConfigurationService.CreateFileServer

Parameter Name	Qualifier	Type	**Description & Notes**
ElementName	IN	string	An end user relevant name for the file server being created. The value shall be stored in the 'ElementName' property for the created element. This parameter shall not be NULL or the empty string.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
TheElement	OUT, REF	CIM_ComputerSystem	The newly created FileServer.
FileServerSettings	IN, OUT, EI, NULL allowed	string	EmbeddedInstance ("SNIA_FileServerSettings") The FileServerSettings for the newly created FileServer. If NULL or the empty string, a default FileServerSettings shall be used and returned on output.
IPInterfaceSettingData	IN,OUT, EI, NULL allowed	string	EmbeddedInstance ("CIM_IPInterfaceSettingData") The IPInterfaceSettingData that specifies the IP Interface that the FileServer will use for servicing all CIFS and NFS requests. If NULL or the empty string, a default IPInterfaceSettingData shall be used and returned on output.
CIFSSettingData	IN,OUT, EI, NULL allowed	string	EmbeddedInstance ("SNIA_CIFSSettingData") The CIFSSettingData that specifies the CIFS settings for the FileServer being created. If this is NULL, the FileServer shall not have CIFS enabled and the resulting CIFSSettingData instance created shall have its “Enabled” property set to false. The CIFSSettingData instance will be returned on output.
NFSSettingData	IN,OUT, EI, NULL allowed	string	EmbeddedInstance ("SNIA_NFSSettingData") The NFSSettingData that specifies the NFS settings for the FileServer being created. If this is NULL, the FileServer shall not have NFS enabled and the resulting NFSSettingData instance created shall have its “Enabled” property set to false. The NFSSettingData instance will be returned on output.

Table 44 - Parameters for Extrinsic Method FileServerConfigurationService.CreateFileServer

Parameter Name	Qualifier	Type	**Description & Notes**
DNSSettingData	IN, EI, NULL allowed,	string	EmbeddedInstance ("CIM_DNSSettingData") The DNSSettingData that specifies the DNS settings for the FileServer being created. If this is NULL, the FileServer shall not have access to a DNS server and a DNSSettingData instance shall not be instantiated for the FileServer.
NISSettingData	IN, EI, NULL allowed,	string	EmbeddedInstance ("CIM_DNSSettingData") The NISSettingData that specifies the NIS settings for the FileServer being created. If this is NULL, the FileServer shall not have access to a NIS server and a NISSettingData instance shall not be instantiated for the FileServer.
NASComputerSystem	IN, REF	CIM_ComputerSystem	Either the NAS Head or Self-contained NAS system that the FileServer shall be a component system of.
HostingComputerSystem	IN, REF	CIM_ComputerSystem	The HostingComputerSystem identifies the ComputerSystem that will host the FileServer.
EthernetPort	IN, REF	CIM_EthernetPort	Fil
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

6.4.1.3 Signature and Parameters of FileServerConfigurationService.ModifyFileServer

This extrinsic modifies the settings for an existing FileServer. All settings except IPInterfaceSettingData, Hosting ComputerSystem, and EthernetPort may be modified. To modify the IPInterfaceSettingData, Hosting ComputerSystem, and/or EthernetPort properties, use the ModifyIPInterface extrinsic.

Table 45 - Parameters for Extrinsic Method FileServerConfigurationService.ModifyFileServer

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN,OUT,REF	CIM_ComputerSystem	The FileServer that is to be modified.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
ElementName	IN, NULL allowed	string	An end user relevant name for the file server being modified.
FileServerSettings	IN, NULL allowed	string	EmbeddedInstance ("SNIA_FileServerSettings") If non-NULL, this specifies the new FileServerSettings for the FileServer If NULL, then the FileServerSettings of the FileServer shall ot be modified.
CIFSSettingData	IN, NULL allowed,	string	EmbeddedInstance ("SNIA_CIFSSettingData") IF non-NULL, this specifies the new CIFS settings for the FileServer. If the “Enabled” property set to false, CIFS will be disabled for the FileServer. If NULL, then the CIFS setting of the FileServer shall not be modified.
NFSSettingData	IN, NULL allowed,	string	EmbeddedInstance ("SNIA_NFSSettingData") IF non-NULL, this specifies the new NFS settings for the FileServer. If the “Enabled” property set to false, NFS will be disabled for the FileServer. If NULL, then the NFS setting of the FileServer shall not be modified.
DNSSettingData	IN, NULL allowed,	string	EmbeddedInstance ("CIM_DNSSettingData") IF non-NULL, this specifies the new DNS settings for the FileServer. If NULL, then the DNS setting of the FileServer shall not be modified.
NISSettingData	IN, NULL allowed,	string	EmbeddedInstance ("CIM_DNSSettingData") IF non-NULL, this specifies the new NIS settings for the FileServer. If NULL, then the NIS setting of the FileServer shall not be modified.
Normal Return			

Table 45 - Parameters for Extrinsic Method FileServerConfigurationService.ModifyFileServer

Parameter Name	Qualifier	Type	Description & Notes
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.
CannotModify	OUT, Indication	CIM_Error	The FileServer is in a state in which it cannot be modified.

6.4.1.4 Signature and Parameters of FileServerConfigurationService.DeleteFileServer

This extrinsics deletes an existing FileServer.

Table 46 - Parameters for Extrinsic Method FileServerConfigurationService.DeleteFileServer

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN,REF	CIM_ComputerSystem	The FileServer that is to be deleted.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
CannotDelete	OUT, Indication	CIM_Error	The FileServer is in a state in which it cannot be deleted.

6.4.1.5 Signature and Parameters of FileServerConfigurationService.AddIPInterface

This extrinsic adds a new IPInterface to an existing FileServer. The FileServer will respond to requests issued to this new IP address. The number of IP addresses that a FileServer can respond on is system dependent and the use of CreateGoalSettings to verify a new IP address is recommended.

Table 47 - Parameters for Extrinsic Method FileServerConfigurationService.AddIPInterface

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN,OUT,REF	CIM_ComputerSystem	The FileServer to which the IPInterface will be added.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
IPInterfaceSettingData	IN	string	EmbeddedInstance ("CIM_IPInterfaceSettingData") The IPInterfaceSettingData that specifies the settings of the IP Interface to be added to the FileServer.
HostingComputerSystem	IN, REF	CIM_ComputerSystem	The ComputerSystem that will host theFile Server for the new IP Interface
EthernetPort	IN, REF	CIM_EthernetPort	The EthernetPort identifies the hardware port that the File Server will use for mount requests on the new IPAddress.
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

6.4.1.6 Signature and Parameters of FileServerConfigurationService.ModifyIPInterface

This extrinsic modifies an existing IPInterface associated with a FileServer. The IPInterfaceSettingData, the Hosting ComputerSystem, and/or the EthernetPort may be modified.

Table 48 - Parameters for Extrinsic Method FileServerConfigurationService.ModifyIPInterface

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN,OUT,REF	CIM_ComputerSystem	The FileServer from which the IPInterface will be modified.
IPInterfaceSettingData	IN,REF	SNIA_IPInterfaceSettingData	The IPInterfaceSettingData that is to be modified. This is used to identify which IPInterfaceSettingData instance to modify.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
NewIPInterfaceSettingData	IN, NULL allowed	string	EmbeddedInstance ("CIM_IPInterfaceSettingData") If non-NULL, the IPInterfaceSettingData that will replace an existing IPInterfaceSettingData instance in the FileServer. If NULL, then the IPInterfaceSettingData will not be modified.
HostingComputerSystem	IN, REF, NULL allowed	CIM_ComputerSystem	If non-NULL, the new ComputerSystem that will host the IPInterface.. If NULL, the current ComputerSystem hosting the IPInterface will remain unchanged.
EthernetPort	IN, REF, NULL allowed	CIM_EthernetPort	If non-NULL, the EthernetPort identifies the new hardware port for the IPInterface . If NULL, the current EthernetPort setting will not be changed.
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value

6.4.1.7 Signature and Parameters of FileServerConfigurationService.DeleteIPInterface

This extrinsic deletes an existing IPInterface associated with a FileServer.

Table 49 - Parameters for Extrinsic Method FileServerConfigurationService.DeleteIPInterface

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN,OUT,REF	CIM_ComputerSystem	The FileServer from which the IPInterface will be deleted.
IPInterfaceSettingData	IN,REF	SNIA_IPInterfaceSettingData	The IPInterfaceSettingData that is to be deleted. This is used to identify which IPInterfaceSettingData instance to delete from the FileServer.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value

6.5 Client Considerations and Recipes

Under Consideration for a future standard.

6.6 Registered Name and Version

File Server Manipulation version 1.3.0

6.7 CIM Elements

Table 50 describes the CIM elements for File Server Manipulation.

Table 50 - CIM Elements for File Server Manipulation

Element Name	Requirement	Description
6.7.1 CIM_ConcreteComponent (FileServerSettings to CIFSSettingData)	Optional	Represents the association between a FileServerSettings and CIFSSettingData.
6.7.2 CIM_ConcreteComponent (FileServerSettings to DNSSettingData)	Conditional	Conditional requirement: The DNSSettingData has been defined.Represents the association between a FileServerSettings and DNSSettingData.
6.7.3 CIM_ConcreteComponent (FileServerSettings to IPInterfaceSettingData)	Mandatory	Represents the association between a FileServerSettings and IPInterfaceSettingData
6.7.4 CIM_ConcreteComponent (FileServerSettings to NFSSettingData)	Optional	Represents the association between a FileServerSettings and NFSSettingData.
6.7.5 CIM_ConcreteComponent (FileServerSettings to NISSettingData)	Conditional	Conditional requirement: The NISSettingData has been defined.Represents the association between a FileServerSettings and NISSettingData.
6.7.6 CIM_DNSSettingData	Optional	This element represents the DNS setting data to be used by a file server.
6.7.7 CIM_ElementCapabilities (FileServerConfigurationService to FileServerCapabilities)	Mandatory	This associates the File Server Configuration Service to the Capabilities element that represents the capabilities supported for the File Server.
6.7.8 CIM_ElementCapabilities (FileServerConfigurationService to FileServerConfigurationCapabilities)	Mandatory	This associates the File Server Configuration Service to the ConfigurationCapabilities element that represents the capabilities that it supports.
6.7.9 CIM_ElementSettingData (ComputerSystem FileServer to FileServerSettings)	Mandatory	Associates a FileServer with its FileServerSettings
6.7.10 CIM_ElementSettingData (IPInterfaceSettingData to IPProtocolEndpoint)	Mandatory	The IPProtocolEndpoint associated with the IPInterfaceSettingData
6.7.11 CIM_HostedDependency	Mandatory	Associates a File Server to the Computer System hosting it.
6.7.12 CIM_HostedService (Hosting Computer System to FileServerConfigurationService)	Mandatory	Associates the FileServerConfigurationService with the hosting computer system
6.7.13 CIM_MemberOfCollection (The IPProtocolEndpoint to NetworkVLAN.)	Conditional	Conditional requirement: The NetworkVLAN has been defined.Associates an IPProtocolEndPoint to NetworkVLAN

Table 50 - CIM Elements for File Server Manipulation

Element Name	Requirement	Description
6.7.14 CIM_NetworkVLAN	Optional	This element represents the virtual LAN (VLAN) tag settings for an IP interface. In the context of a file server, it represents the VLAN information
6.7.15 CIM_SettingsDefineCapabilities (CIFSettingData)	Mandatory	Associates CIFSSettingData with FileServerCapabilities
6.7.16 CIM_SettingsDefineCapabilities (DNSSettingData)	Mandatory	Associates DNSSSettingData with FileServerCapabilities
6.7.17 CIM_SettingsDefineCapabilities (FileServerSettings)	Mandatory	Associates FileServerSettings with FileServerCapabilities
6.7.18 CIM_SettingsDefineCapabilities (IPInterfaceSettingData)	Mandatory	Associates IPInterfaceSettingData with FileServerCapabilities
6.7.19 CIM_SettingsDefineCapabilities (NFSSettingData)	Mandatory	Associates NFSSSettingData with FileServerCapabilities
6.7.20 CIM_SettingsDefineCapabilities (NISSettingData)	Mandatory	Associates NISSSettingData with FileServerCapabilities
6.7.21 CIM_SettingsDefineState (ComputerSystem FileServer to FileServerSettings)	Mandatory	The FileServer's state represented by its FileServerSettings
6.7.22 SNIA_CIFSSettingData	Mandatory	This class contains the CIFS settings for the File Server
6.7.23 SNIA_FileServerCapabilities	Mandatory	The capabilities of the File Server.
6.7.24 SNIA_FileServerConfigurationCapabilities	Mandatory	This element represents the management Capabilities of the File Server Configuration Service.
6.7.25 SNIA_FileServerConfigurationService	Mandatory	The File Server Configuration Service provides the methods to manipulate File Servers.
6.7.26 SNIA_FileServerSettings	Mandatory	This class contains the settings for the File Server
6.7.27 SNIA_IPInterfaceSettingData	Mandatory	This class contains the settings for single IP interface.
6.7.28 SNIA_NFSSettingData	Mandatory	This class contains the NFS settings for the File Server
6.7.29 SNIA_NISSettingData	Optional	This class contains the NIS settings for the File Server

6.7.1 CIM_ConcreteComponent (FileServerSettings to CIFSSettingData)

Created By: External

Modified By: Static
Deleted By: External
Requirement: Optional

Table 51 describes class CIM_ConcreteComponent (FileServerSettings to CIFSSettingData).

Table 51 - SMI Referenced Properties/Methods for CIM_ConcreteComponent (FileServerSettings to CIFSSettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerSettings
PartComponent		Mandatory	The CIFSSettingData

6.7.2 CIM_ConcreteComponent (FileServerSettings to DNSSettingData)

Created By: External
Modified By: Static
Deleted By: External
Requirement: The DNSSettingData has been defined.

Table 52 describes class CIM_ConcreteComponent (FileServerSettings to DNSSettingData).

Table 52 - SMI Referenced Properties/Methods for CIM_ConcreteComponent (FileServerSettings to DNSSettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerSettings
PartComponent		Mandatory	The DNSSettingData

6.7.3 CIM_ConcreteComponent (FileServerSettings to IPInterfaceSettingData)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 53 describes class CIM_ConcreteComponent (FileServerSettings to IPInterfaceSettingData).

Table 53 - SMI Referenced Properties/Methods for CIM_ConcreteComponent (FileServerSettings to IPInterfaceSettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerSettings
PartComponent		Mandatory	The IPInterfaceSettingData

6.7.4 CIM_ConcreteComponent (FileServerSettings to NFSSettingData)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Optional

Table 54 describes class CIM_ConcreteComponent (FileServerSettings to NFSSettingData).

Table 54 - SMI Referenced Properties/Methods for CIM_ConcreteComponent (FileServerSettings to NFSSettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerSettings
PartComponent		Mandatory	The NFSSettingData

6.7.5 CIM_ConcreteComponent (FileServerSettings to NISSettingData)

Created By: External
Modified By: Static
Deleted By: External
Requirement: The NISSettingData has been defined.

Table 55 describes class CIM_ConcreteComponent (FileServerSettings to NISSettingData).

Table 55 - SMI Referenced Properties/Methods for CIM_ConcreteComponent (FileServerSettings to NISSettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerSettings
PartComponent		Mandatory	The NISSettingData

6.7.6 CIM_DNSSettingData

Created By: Extrinsic
Modified By: Extrinsic
Deleted By: Extrinsic
Requirement: Optional

Table 56 describes class CIM_DNSSettingData.

Table 56 - SMI Referenced Properties/Methods for CIM_DNSSettingData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the DNSSettingData.
DomainName		Mandatory	The DNS domain to use for looking up addresses.
DNSServerAddresses		Mandatory	The addresses of DNS servers to contact. The array specifies the order in which the DNS servers will be contacted.

6.7.7 CIM_ElementCapabilities (FileServerConfigurationService to FileServerCapabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 57 describes class CIM_ElementCapabilities (FileServerConfigurationService to FileServerCapabilities).

Table 57 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (FileServerConfigurationService to FileServerCapabilities)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The File Server Configuration Service
Capabilities		Mandatory	The File Server Capabilties element

6.7.8 CIM_ElementCapabilities (FileServerConfigurationService to FileServerConfigurationCapabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 58 describes class CIM_ElementCapabilities (FileServerConfigurationService to FileServerConfigurationCapabilities).

Table 58 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (FileServerConfigurationService to FileServerConfigurationCapabilities)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The File Server Configuration Service
Capabilities		Mandatory	The File Server Configuration Capabilties element

6.7.9 CIM_ElementSettingData (ComputerSystem FileServer to FileServerSettings)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 59 describes class CIM_ElementSettingData (ComputerSystem FileServer to FileServerSettings).

Table 59 - SMI Referenced Properties/Methods for CIM_ElementSettingData (ComputerSystem FileServer to FileServerSettings)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The File Server ComputerSystem.
SettingData		Mandatory	The FileServerSettings.

6.7.10 CIM_ElementSettingData (IPInterfaceSettingData to IPProtocolEndpoint)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 60 describes class CIM_ElementSettingData (IPInterfaceSettingData to IPProtocolEndpoint).

Table 60 - SMI Referenced Properties/Methods for CIM_ElementSettingData (IPInterfaceSetting-Data to IPProtocolEndpoint)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The IPProtocolEndpoint
SettingData		Mandatory	The IPInterfaceSettingData.

6.7.11 CIM_HostedDependency

Created By: Extrinsic: CreateFileServer
Modified By: Static
Deleted By: Extrinsic: DeleteFileServer
Requirement: Mandatory

Table 61 describes class CIM_HostedDependency.

Table 61 - SMI Referenced Properties/Methods for CIM_HostedDependency

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The File Server ComputerSystem (Dedicated="FileServer").
Antecedent		Mandatory	The hosting ComputerSystem

6.7.12 CIM_HostedService (Hosting Computer System to FileServerConfigurationService)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 62 describes class CIM_HostedService (Hosting Computer System to FileServerConfigurationService).

Table 62 - SMI Referenced Properties/Methods for CIM_HostedService (Hosting Computer System to FileServerConfigurationService)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The File Server Configuration Service.
Antecedent		Mandatory	The hosting ComputerSystem.

6.7.13 CIM_MemberOfCollection (The IPProtocolEndpoint to NetworkVLAN.)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: The NetworkVLAN has been defined.

Table 63 describes class CIM_MemberOfCollection (The IPProtocolEndpoint to NetworkVLAN.).

Table 63 - SMI Referenced Properties/Methods for CIM_MemberOfCollection (The IPProtocolEndpoint to NetworkVLAN.)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The IPProtocolEndPoint.
Member		Mandatory	The NetworkVLAN.

6.7.14 CIM_NetworkVLAN

Created By: Extrinsic
Modified By: Extrinsic
Deleted By: Extrinsic
Requirement: Optional

Table 64 describes class CIM_NetworkVLAN.

Table 64 - SMI Referenced Properties/Methods for CIM_NetworkVLAN

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the NetworkVLAN.
VLANId		Mandatory	The VLAN id that is to be associated with an IP interface. The id shall be included in all IP packets being sent through an IP interface.
TransmissionSize		Mandatory	The maximum transmission unit size that is associated with an IP Interface.

6.7.15 CIM_SettingsDefineCapabilities (CIFSettingData)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 65 describes class CIM_SettingsDefineCapabilities (CIFSettingData).

Table 65 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (CIFSetting-Data)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The CIFSSettingData reference.

6.7.16 CIM_SettingsDefineCapabilities (DNSSettingData)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 66 describes class CIM_SettingsDefineCapabilities (DNSSettingData).

Table 66 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (DNSSetting-Data)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The CIFSSettingData reference.

6.7.17 CIM_SettingsDefineCapabilities (FileServerSettings)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 67 describes class CIM_SettingsDefineCapabilities (FileServerSettings).

Table 67 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (FileServer-Settings)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The FileServerSetting reference.

6.7.18 CIM_SettingsDefineCapabilities (IPInterfaceSettingData)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 68 describes class CIM_SettingsDefineCapabilities (IPInterfaceSettingData).

Table 68 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (IPInterfaceS-ettingData)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The IPInterfaceSettingData reference.

6.7.19 CIM_SettingsDefineCapabilities (NFSSettingData)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 69 describes class CIM_SettingsDefineCapabilities (NFSSettingData).

Table 69 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (NFSSetting-Data)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The NFSSSettingData reference.

6.7.20 CIM_SettingsDefineCapabilities (NISSettingData)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 70 describes class CIM_SettingsDefineCapabilities (NISSettingData).

Table 70 - SMI Referenced Properties/Methods for CIM_SettingsDefineCapabilities (NISSetting-Data)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The FileServerCapabilities reference.
PartComponent		Mandatory	The NISSSettingData reference.

6.7.21 CIM_SettingsDefineState (ComputerSystem FileServer to FileServerSettings)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 71 describes class CIM_SettingsDefineState (ComputerSystem FileServer to FileServerSettings).

Table 71 - SMI Referenced Properties/Methods for CIM_SettingsDefineState (ComputerSystem FileServer to FileServerSettings)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The File Server ComputerSystem.
SettingData		Mandatory	The FileServerSettings.

6.7.22 SNIA_CIFSSettingData

Created By: Extrinsic: CreateFileServer
Modified By: Extrinsic: ModifyCIFS
Deleted By: Extrinsic: DeleteFileServer
Requirement: Mandatory

Table 72 describes class SNIA_CIFSSettingData.

Table 72 - SMI Referenced Properties/Methods for SNIA_CIFSSettingData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the CIFSSettingData.
Enabled		Mandatory	This boolean indicates if CIFS is enabled on the File Server.
Charset		Optional	Specifies the character set to be used by the File Server when servicing CIFS Shares. The values are 0|1|2 ('Standard-ASCII'|'IBM-437','IBM-850'). If absent, then "Standard-ASCII" is assumed.
UseTCPOnly		Optional	This boolean if set to 'true' allows only TCP transport connections. If 'false', then both TCP and Netbios transport connections are allowed. The default value is 'false'.
NETBIOSName		Optional	The NetBIOS name of the FileServer.
WINSIP		Optional	An array of IP Addresses of Windows Internet Name Servers.
AuthenticationDomain		Mandatory	Name of CIFS domain to which the File Server is joined. Represents either the NTLM domain or the ActiveDirectory domain.
AuthenticationMode		Mandatory	Specifies if authentication is to be performed against either NTLM or ActiveDirectory domains. Valid values are 'NTLM' or 'ActiveDirectory'.
UseKerberos		Optional	Determines how ActiveDirectory authentication is performed. If 'true', limit ActiveDirectory authentication to use Kerberos. Otherwise do not limit to Kerberos only.

Table 72 - SMI Referenced Properties/Methods for SNIA_CIFSSettingData

Properties	Flags	Requirement	Description & Notes
UseOpportunisticLocking		Optional	This boolean determines if opportunistic locking should be used by CIFS FileServer. If 'true', enable opportunistic locking.
SMBSigningOnly		Optional	This boolean determines if CIFS clients are allowed to connect if they use SMB signing for security. If 'true', then require clients to use SMB signing. Otherwise, do not require.
ClientsConnectAnonymously		Optional	This boolean dictates if the FileServer joins the CIFS Domain Controller anonymously or if a user and password are required. If 'true', then join anonymously. Otherwise, use DomainControllerUser and DomainControllerPassword to join.
JoinDomainAnonymously		Optional	This boolean dictates if the FileServer joins the CIFS Domain Controller anonymously or if a user and password are required. If 'true', then join anonymously. Otherwise, use DomainControllerUser and DomainControllerPassword to join.
DomainControllerUser		Optional	User name to use when the Fileserver joins the CIFS Domain Controller
DomainControllerPassword		Optional	Password to use when joining the CIFS Domain Controller
CIFSDomainController		Optional	Name of the CIFS Domain Controller

6.7.23 SNIA_FileServerCapabilities

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 73 describes class SNIA_FileServerCapabilities.

Table 73 - SMI Referenced Properties/Methods for SNIA_FileServerCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the FileServerCapabilities element of a File Server Configuration Service.

Table 73 - SMI Referenced Properties/Methods for SNIA_FileServerCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Capabilities element.
CreateGoalSettings()		Mandatory	This extrinsic method supports the creation of a set of Settings that are a supported variant of the Settings passed as embedded instances via IN parameters. The method returns the supported Settings in OUT parameters, each containing an array of embedded instances. Many of the IN parameters are optional, and if left NULL result in NULL being returned in the corresponding OUT parameters.

6.7.24 SNIA_FileServerConfigurationCapabilities

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 74 describes class SNIA_FileServerConfigurationCapabilities.

Table 74 - SMI Referenced Properties/Methods for SNIA_FileServerConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for this element representing the capabilities of a File Server Configuration Service.
ElementName		Mandatory	A user-friendly name for this Capabilities element.
SynchronousMethodsSupported	N	Mandatory	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called synchronously. Note: A supported method shall be listed in this property or in the AsynchronousMethodsSupported property or both.
AsynchronousMethodsSupported	N	Mandatory	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called asynchronously. Note: A supported method shall be listed in this property or in the SynchronousMethodsSupported property or both.

6.7.25 SNIA_FileServerConfigurationService

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 75 describes class SNIA_FileServerConfigurationService.

Table 75 - SMI Referenced Properties/Methods for SNIA_FileServerConfigurationService

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Service.
SystemCreationClassName		Mandatory	Key
SystemName		Mandatory	Key
CreationClassName		Mandatory	Key
Name		Mandatory	Key
CreateFileServer()		Mandatory	Create a new instance of File Server
ModifyFileServer()		Mandatory	Modify an existing File Server. This is used to modify FileServerSettings, CIFSSettingData, NFSSettingData, DNSSettingData, or NISSettingData.
DeleteFileServer()		Mandatory	Delete an existing File Server.
AddIPInterface()		Optional	Add a new IPInterface to an existing File Server
ModifyIPInterface()		Optional	Modify an IPInterface associated with an existing File Server
DeleteIPInterface()		Optional	Delete an IPInterface associated with an existing File Server

6.7.26 SNIA_FileServerSettings

Created By: Extrinsic: CreateFileServer
Modified By: Extrinsic: ModifyFileServer
Deleted By: Extrinsic: DeleteFileServer
Requirement: Mandatory

Table 76 describes class SNIA_FileServerSettings.

Table 76 - SMI Referenced Properties/Methods for SNIA_FileServerSettings

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the FileServerSettings.
HostLookupOrder		Optional	Specifies the services and order to use them for host lookup. An array of elements with these values: 'DNS','NIS', 'None', or 'UploadedFile'. 'UploadedFile' refers to the uploaded file of host names.
UserLoginLookupOrder		Optional	Specifies the services and order to use them for user lookup. An array of elements with these values: 'DNS','NIS', 'None', or 'UploadedFile'. 'file' 'UploadedFile' refers to the uploaded file of user passwords.

Table 76 - SMI Referenced Properties/Methods for SNIA_FileServerSettings

Properties	Flags	Requirement	Description & Notes
NFSCIFSAccountMapping		Optional	Controls the mapping of accounts between NFS and CIFS. Valid values are 'None', 'All', or 'Domain'. If 'None', then no account mapping is performed. If 'All', then mapping is done for all CIFS domains. If 'Domain', then mapping is done for the users in the CIFS domain specified in AccountMappingDomain.
AccountMappingDomain		Optional	If NFSCIFSAccountMapping = 'Domain', then this property will contain the name of the domain to use for NFS to CIFS account mapping.

6.7.27 SNIA_IPInterfaceSettingData

Created By: Extrinsic: CreateFileServer | AddIPInterface
Modified By: Extrinsic: ModifyIPInterface
Deleted By: Extrinsic: DeleteFileServer | DeleteIPInterface
Requirement: Mandatory

Table 77 describes class SNIA_IPInterfaceSettingData.

Table 77 - SMI Referenced Properties/Methods for SNIA_IPInterfaceSettingData

Properties	Flags	Requirement	Description & Notes
IPAddress		Mandatory	The IPAddress that will be used by the File Server. This can be either an IPv4 or IPv6 address.
AddressType		Mandatory	The IPAddress format. This can be either "IPv4" or "IPv6".
SubnetMask		Mandatory	The subnet mask that will be used by the File Server
IPv6PrefixLength		Conditional	Conditional requirement: Required if the array property SNIA_IPInterfaceSettingData.AddressType contains the string \IPv6\'.'If AddressType specifies IPv6, then this specifies the prefix length for the IPv6 address in IPAddress. Is this really needed???
VLANId		Optional	If present contains the ID of the VLAN that this IP setting will be associated with.
MTU		Optional	If present contains the maximum transmission unit to be used for this IP setting. If not present, then the default of 1500 will be used.

6.7.28 SNIA_NFSSettingData

Created By: Extrinsic: CreateFileServer
Modified By: Extrinsic: ModifyNFS
Deleted By: Extrinsic: DeleteFileServer

Requirement: Mandatory

Table 78 describes class SNIA_NFSSettingData.

Table 78 - SMI Referenced Properties/Methods for SNIA_NFSSettingData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the NFSSettingData.
Enabled		Mandatory	This boolean indicates if NFS is enabled on the File Server.
Charset		Optional	Specifies the character set to be used by the File Server when servicing CIFS Shares. The values are 0|1|2 ('Standard-ASCII'|'UTF8'|'ISO-8859-1'). If absent, then 'ISO-8859-1' is assumed.
MaximumTCPConnections		Optional	This specifies the number of concurrent TCP connections that are allowed for the NFS protocol. If set to 0, then TCP will be disabled for NFS.
Port		Optional	The port the File Server listens for mount requests. If absent, default to 2049.
NonNFSuid		Optional	User ID to use for requests from non-NFS access. If absent, default to -1.
NonNFSgid		Optional	Group ID to use for requests from non-NFS access. If absent, default to -1.
UseReservedPorts		Optional	This boolean specifies that the File Server will only allow NFS mount requests from client machine TCP/IP ports less than 1024. If 'true', only allow mount requests from ports less than 1024. Othewise, allow mount requests from any client port.
OnlyRootChown		Optional	This boolean specifies if the root user is allowed to issue chown (change ownership) requests. If 'true', then only let root user issue chown request. Otherwise, allow any user to issue chown requests.

6.7.29 SNIA_NISSettingData

Created By: Extrinsic: CreateFileServer
Modified By: Extrinsic: ModifyFileServer
Deleted By: Extrinsic: DeleteFileServer
Requirement: Optional

Table 79 describes class SNIA_NISSettingData.

Table 79 - SMI Referenced Properties/Methods for SNIA_NISSettingData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the NISSettingData.
DomainName		Mandatory	NIS Domain Name
ServerIP		Mandatory	An array of IP Addresses IP Addresses of NIS Servers

EXPERIMENTAL

STABLE

Clause 7: File Storage Profile

7.1 Description

7.1.1 Synopsis

Profile Name: File Storage

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: N/A

Scoping Class: ComputerSystem

7.1.2 Overview

The File Storage Profile is a subprofile for autonomous profiles that support filesystems. Specifically, in this release of SMI-S, this includes the NAS Head and Self-Contained NAS Profiles.

7.1.3 Implementation

Figure 10 illustrates the mandatory and optional classes for the modeling of file storage for the profiles that support filesystems. This profile is supported by the Self-contained NAS and the NAS Head Profiles.

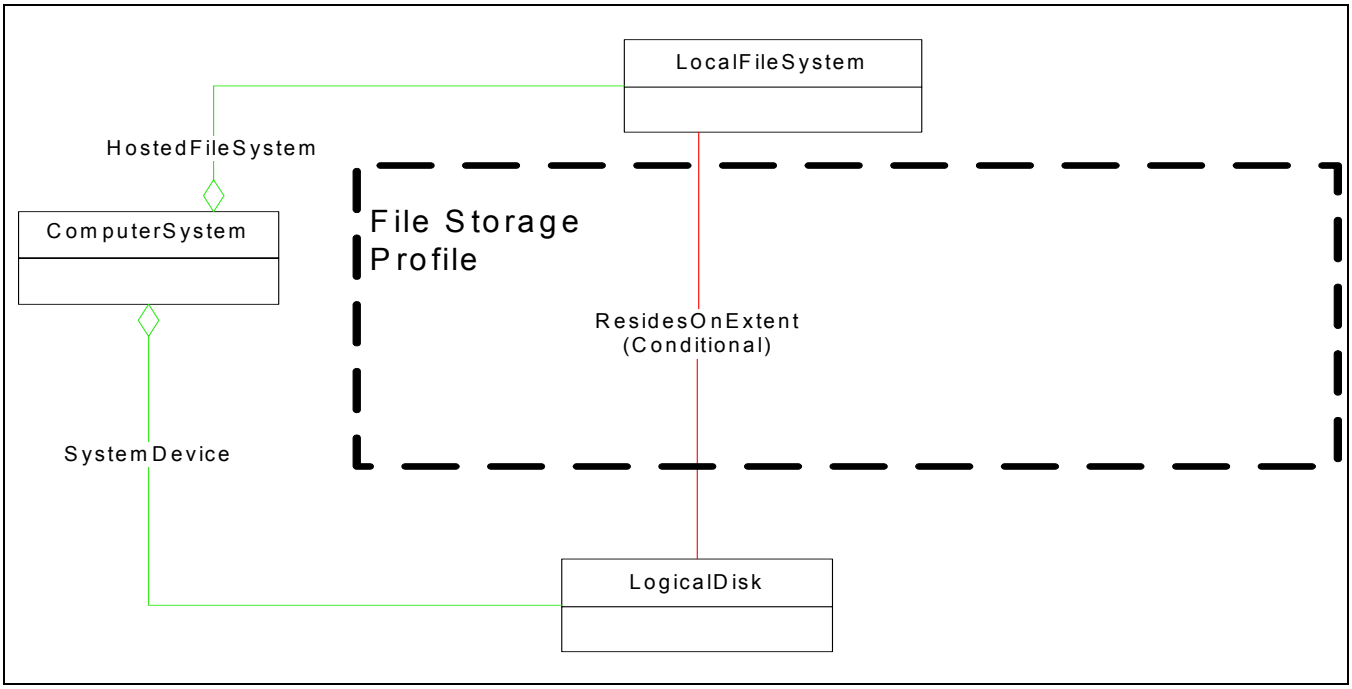

Figure 10 - File Storage Instance

The File Storage profile models the mapping of Filesystems to LogicalDisks. For the NAS Head and Self-contained NAS profiles each Filesystem shall be established on one LogicalDisk. The relationship between the LocalFileSystem and the LogicalDisk is represented by the ResidesOnExtent association. This association is listed as conditional on the parent profile being either the NAS Head or the Self-contained NAS profile.The LogicalDisk may be a LogicalDisk as defined in the Block Services Package or part of the parent profile.

The FileStorage Profile is a “read-only” profile. That is, the methods for creating, modifying or deleting a LocalFileSystem are external to the File Storage Profile. The SMI-S prescribed way of performing these functions are covered by the Filesystem Manipulation Profile.

7.2 Health and Fault Management Consideration

None.

EXPERIMENTAL

7.3 Cascading Considerations

In some cases, the parent profile does not implement Block Services Package. In this case, the parent profile would implement a LogicalDisk that is “imported” from another Profile (e.g., a Volume Management Profile). This section discusses those cascading considerations.

7.3.1 Cascaded Resources

A File Storage profile may get its storage from the operating system (HDR Profile), a volume manager, an Array or Storage Virtualizer. As such, there is a cascading relationship between the File Storage Profile and the Profiles (e.g., Volume Management Profiles) that provide the storage for the File Storage Profile. Figure 11 illustrates the constructs to be used to model this cascading relationship.

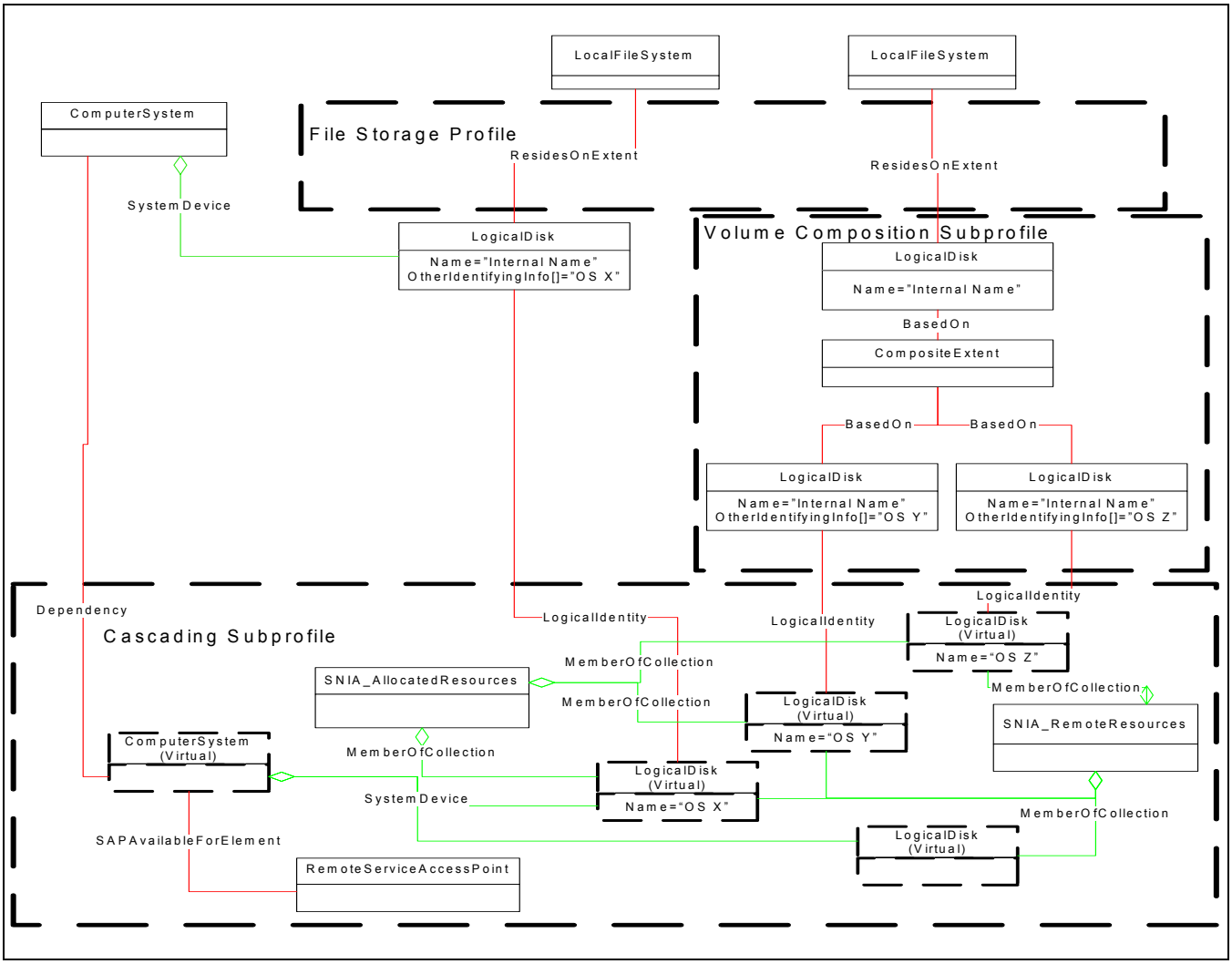

Figure 11 - Cascading File Storage

Figure 11 shows two filesystems (LocalFileSystem). Both reside on one LogicalDisk. But the LogicalDisk on the right is a composite of lower level LogicalDisks. The storage that is imported from the remote profile are LogicalDisks at the lowest level of the Filesystem Profile. So, in the first (left side) case, the LogicalIdentity is between the LogicalDisk on which the filesystem resides to the imported LogicalDisk (or StorageVolume). In the second case (the right side) the LogicalIdentity is between the “lowest level” LogicalDisks in Volume Composition and the imported LogicalDisks (or StorageVolumes).

Note: LogicalIdentity is an abstract class and would be subclassed by an implementation.

The ComputerSystem in the Filesystem Profile would be the computer system that hosts the filesystem. The “Virtual” ComputerSystem is the top level ComputerSystem of the HDR, Volume Manager, Array or Storage Virtualizer. There shall be a Dependency association between these computer systems. LogicalDisks (or StorageVolumes) that are in use by the Filesystem Profile would have a MemberOfCollection association to the SNIA_AllocatedResources collection. All the LogicalDisks (or StorageVolumes) that the Filesystem Profile can see (including the ones that are allocated) would have a MemberOfCollection association to the SNIA_RemoteResources instance.

The RemoteServiceAccessPoint associated to the virtual computer system via SAPAvailableForElement would be information on the management interface for the HDR, Volume Manager, Array or Storage Virtualizer.

Table 80 provides the specific cascading information for cascading file storage.

Table 80 - Cascaded Storage

File Storage Resource	Leaf Profile	Leaf Resource	Association	Notes
LogicalDisk	Volume Management or HDR	LogicalDisk	LogicalIdentity	
LogicalDisk	Array or Storage Virtualizer	StorageVolume	LogicalIdentity	

7.3.2 Ownership Privileges

In support of the cascading File Storage, an implementation may assert ownership over the LogicalDisks (or StorageVolumes) that they import. If the Volume Management implementation supports Ownership, the File Storage implementation may assert ownership using the following Privileges:

- Activity - Execute
- ActivityQualifier - CreateOrModifyFromStoragePool and ReturnToStoragePool
- FormatQualifier - Method

Note: HDR does not support Block Storage Resource Ownership, so this cannot be supported if the underlying profile is HDR.

7.3.3 Limitations on Cascading Subprofile

The File Storage Profile support for cascading places the following limitations and restrictions on the Cascading Subprofile:

- Dependency - The Dependency may exist, even when there are no resources that are imported. This signifies that the File Storage implementation has discovered the Volume Management or HDR profile, but has no access to any of their LogicalDisks.

EXPERIMENTAL

7.4 Supported Profiles, Subprofiles, and Packages

Related Profiles for File Storage: Not defined in this standard.

7.5 Methods of the Profile

7.5.1 Extrinsic Methods of the Profile

None

Note: The methods for defining the various mappings would be handled by the Filesystem Manipulation subprofile.

7.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

7.6 Client Considerations and Recipes

None.

7.7 Registered Name and Version

File Storage version 1.3.0

7.8 CIM Elements

Table 81 describes the CIM elements for File Storage.

Table 81 - CIM Elements for File Storage

Element Name	Requirement	Description
7.8.1 CIM_ResidesOnExtent	Conditional	Conditional requirement: NAS Profiles require that LocalFileSystems reside on one LogicalDisk. or NAS Profiles require that LocalFileSystems reside on one LogicalDisk..Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.

7.8.1 CIM_ResidesOnExtent

Created By: External

Modified By: Static

Deleted By: External

Requirement: NAS Profiles require that LocalFileSystems reside on one LogicalDisk. or NAS Profiles require that LocalFileSystems reside on one LogicalDisk..

Table 82 describes class CIM_ResidesOnExtent.

Table 82 - SMI Referenced Properties/Methods for CIM_ResidesOnExtent

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The LocalFileSystem that is built on top of a LogicalDIsk.
Antecedent		Mandatory	The LogicalDIsk that underlies a LocalFileSystem.

STABLE

STABLE

Clause 8: Filesystem Profile

8.1 Description

8.1.1 Synopsis

Profile Name: Filesystem

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: LocalFileSystem

Scoping Class: ComputerSystem

The Filesystem Profile is a subprofile for autonomous profiles that support filesystems. Specifically, in this release of SMI-S, this includes the NAS Head and the Self-Contained NAS profiles. A number of other profiles and subprofiles make use of elements of the Filesystem profile and will be referred to in this specification as “Filesystem related profiles” -- these include but are not limited to the Filesystem Manipulation subprofile, File Export subprofile, File Export Manipulation subprofile, NAS Head profile, and so on.

The state transitions involved when an appropriate storage element is transformed into a filesystem are described in Annex A: (Informative) State Transitions from Storage to File Shares.

8.1.2 Instance Diagrams

Figure 12 illustrates the mandatory, optional, and conditional classes for the modeling of filesystems for the profiles that support filesystems. This profile is supported by the Self-contained NAS and the NAS Head profiles. The dashed box contains the elements that this profile supports -- the elements outside the dashed box depend on

other profiles or subprofiles for their maintenance (creation, deletion, and modification). There are two

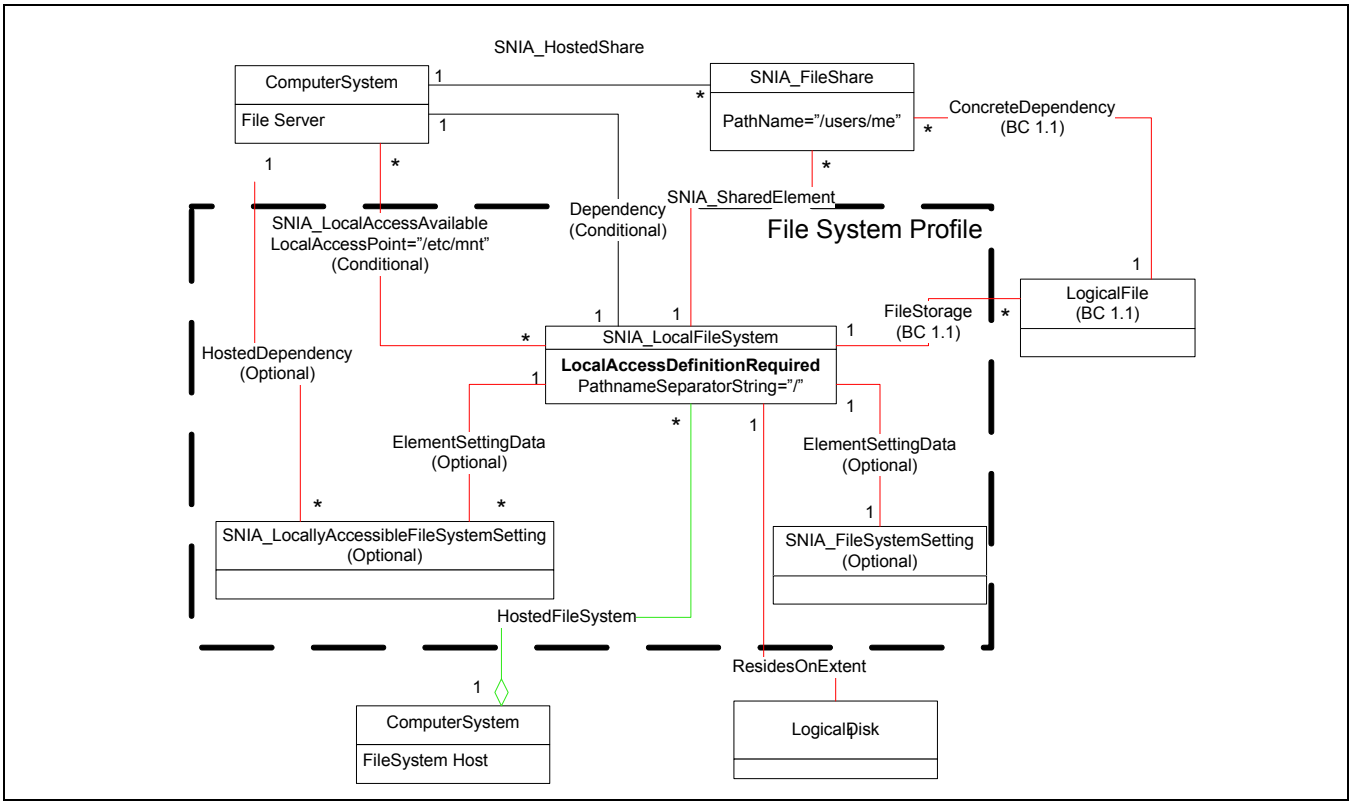

Figure 12 - Filesystem Instance

ComputerSystems shown outside the box that represent different dedicated roles that could be performed by different actual computers (or could be performed by a single computer).

A filesystem shall be represented in the model as an instance of LocalFileSystem. A LocalFileSystem instance shall have a ResidesOnExtent association to a LogicalDisk (shown, but outside this profile). A client would determine the size (in bytes) of a filesystem by inspecting the size of the LogicalDisk on which the LocalFileSystem resides.

Note: The Filesystem related profiles build LocalFileSystems on a LogicalDisk(s). In the cases supported in this release of SMI-S, one LocalFileSystem may be established on one LogicalDisk. In a future release, more elaborate mappings may exist between FileSystems and one or more LogicalDisks.

The LocalFileSystem shall have a HostedFileSystem association to a ComputerSystem. Normally this will be the top level ComputerSystem of the parent profile (typically one of the Filesystem related profiles such as the NAS Head or the Self-Contained NAS Profile). However, if the Multiple Computer System Subprofile is implemented, the HostedFileSystem may be associated to a component ComputerSystem. See Clause 30: Multiple Computer System Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*.

The LocalFileSystem element may also have an ElementSettingData association to the FileSystemSetting for that filesystem. However, the FileSystemSetting and ElementSettingData are optional in this profile.

There may be zero or more FileShare elements associated to the LocalFileSystem element via the SharedElement association. An implementation would be required to populate only those FileShare elements representing files (or directories) that are exported using a supported file sharing protocol (such as CIFS or NFS). The path to the file or directory from the root of the LocalFileSystem is specified by the FileShare.PathName property.

Note: In order to support backward compatibility with the NAS Head and Self-contained NAS profiles in SMI-S 1.1, the class LogicalFile (shown outside the dashed box in the figure) and two associations (ConcreteDependency outside the dashed box and FileStorage shown inside the dashed box) must be supported. These duplicate the functionality provided by specifying FileShare.PathName, at the cost of requiring that certain LogicalFiles, but not all, be surfaced.

EXPERIMENTAL

8.1.2.1 Local Access Requirement

In addition, if the property LocalFileSystem.LocalAccessDefinitionRequired is set to true, the filesystem must be made exportable via a file server. In that case, there shall be a LocalAccessAvailable association from the LocalFileSystem element to the file server ComputerSystem and, optionally, a LocallyAccessibleFileSystemSettings element associated via an ElementSettingData association to the LocalFileSystem. The LocallyAccessibleFileSystemSettings element is an instance of ScopedSettingData and is associated to the file server ComputerSystem via a ScopedSetting association. The ScopedSetting association indicates that this setting is constrained by the associated file server. The LocalAccessAvailable association is required but conditional on LocalAccessDefinitionRequired being true, while the LocallyAccessibleFileSystemSettings element and the ScopedSetting association are not required (i.e., optional).

Note: They are still conditional on LocalAccessDefinitionRequired being true, but in this version of SMI-S, that is not represented in the XML file.

Since LocalAccessAvailable is an association, there can only be ONE instance per LocalFileSystem for each FileServer. This is a common restriction . For each LocalAccessAvailable association, there should only be zero (if optionally not implemented) or one (if optionally implemented) instances of LocallyAccessibleFileSystemSettings.

EXPERIMENTAL

EXPERIMENTAL

8.1.2.2 Directory Service Use

A filesystem needs to be supported by a directory service that resolves user and group identifiers (referred to as UID, GID, or SID) to specific operational users and groups. The enumerated property LocalFileSystem.DirectoryServiceUsage indicates the kind of support a filesystem requires from a directory service -- the options include “Not Used”, “Optional”, and “Required”. If “Required”, the filesystem will be associated to a computer system that provides infrastructure support for such identity resolution.

The directory service may be hosted by any ComputerSystem, but it must be accessible in real-time to the ComputerSystem that makes the filesystem available to operational users -- this is either a file server ComputerSystem (one may already be required if LocalFileSystem.LocalAccessDefinitionRequired is true, but it is optional otherwise) or the ComputerSystem hosting the filesystem. The directory service may be “natively” hosted on that ComputerSystem (file server or filesystem host) or may be identified by that ComputerSystem in some way.

The support relationship between a LocalFileSystem element and the ComputerSystem that identifies and uses the directory service shall be represented by a Dependency association with the ComputerSystem element as the Antecedent and the LocalFileSystem element as the Dependent.

In the instance diagram above, the Dependency association has been shown between the LocalFileSystem and a file server ComputerSystem (with Dedicated[]=”16”). A LocalFileSystem element shall only identify one ComputerSystem for directory service access. In addition, the consistency of filesystem security implementation

requires that all the file server ComputerSystems that make a filesystem locally available must use the same directory service or use mutually consistent directory services.

EXPERIMENTAL

8.2 Health and Fault Management Consideration

The Filesystem Profile supports state information (e.g., OperationalStatus) on the following elements of the model:

- Local File Systems (See Table 94 - SMI Referenced Properties/Methods for CIM_LogicalFile)

8.2.1 OperationalStatus for Filesystems

Table 83 - Filesystem OperationalStatus

Primary OperationalStatus	Description
2 “OK”	The filesystem has good status
3 “Degraded”	The filesystem is operating in a degraded mode. This could be due to the health state of the underlying storage being degraded or in error.
4 “Stressed”	The filesystem resources are stressed
5 “Predictive Failure”	The filesystem might fail because some resource or component is predicted to fail
6 “Error”	An error has occurred causing the filesystem to become unavailable. Operator intervention through SMI-S (managing the LocalFileSystem) to restore the filesystem may be possible.
6 “Error”	An error has occurred causing the filesystem to become unavailable. Automated recovery may be in progress.
7 “Non-recoverable Error”	The filesystem is not functioning. Operator intervention through SMI-S will not fix the problem.
8 “Starting”	The filesystem is in process of initialization and is not yet available operationally.
9 “Stopping”	The filesystem is in process of stopping, and is not available operationally.
10 “Stopped”	The filesystem cannot be accessed operationally because it is stopped -- if this did not happened because of operator intervention or happened in real-time, the OperationalStatus would have been “Lost Communication” rather than “Stopped”.
11 “In Service”	The filesystem is offline in maintenance mode, and is not available operationally.
13 “Lost Communications”	The filesystem cannot be accessed operationally -- if this happened because of operator intervention it would have been “Stopped” rather than “Lost Communication”.
14 “Aborted”	The filesystem is stopped but in a manner that may have left it in an inconsistent state.
15 “Dormant”	The Filesystem is offline; and the reason for not being accessible is unknown.
16 “Supporting Entity in Error”	The filesystem is in an error state, or may be OK but not accessible, because a supporting entity is not accessible.

8.3 Cascading Considerations

None.

8.4 Supported Profiles, Subprofiles, and Packages

Table 84 describes the supported profiles for Filesystem.

Table 84 - Supported Profiles for Filesystem

Registered Profile Names	Mandatory	Version
Indication	Yes	1.3.0

8.5 Methods of the Profile

8.5.1 Extrinsic Methods of the Profile

None.

8.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

8.6 Client Considerations: Use Cases

The following client use cases are supported by this profile:

- List Existing Filesystems hosted by the Referencing Profile (parent Filesystem related profile).
- Get FileSystemSettings for a FileSystem
- Get the ComputerSystem that hosts a FileSystem
- Get all File Servers and Access Paths that have Local Access to this FileSystem
- Get the Access Path to this FileSystem on the specified File Server
- Get the Local Access Settings for this FileSystem on the specified File Server
- Get the FileShares and shared File path of this FileSystem on all File Servers
- Get the FileShares and shared File path of this FileSystem on the specified FileServer

EXPERIMENTAL

These use cases have been elaborated as prototype recipes in the following sections.

8.6.1 List Existing Filesystems hosted by the Referencing Profile (parent filesystem related profile

```
// DESCRIPTION
//   Goal: Locate all LocalFileSystems hosted on the top level
//      ComputerSystem of the Filesystem Profile.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the top level ComputerSystem was previously
//      discovered and is defined in the $System-> variable.
//
// FUNCTION ListFileSystems
// This function takes a given top level ComputerSystem and locates
// the LocalFileSystems which it hosts or are hosted by any component
// ComputerSystem.
// INPUT Parameters:
//   System: A reference to the top level ComputerSystem of
//      the Filesystem Profile.
// OUTPUT Parameters:
//   None
// RESULT:
//   Returns: An array of reference(s) to the LocalFileSystems
//      hosted by the top level ComputerSystem or component
//      ComputerSystems.  It returns NULL if it does not find
//      any hosted LocalFileSystems.
sub CIMInstance[] ListFileSystems(REF CIM_ComputerSystem $System->) {

    // Step 1. Locate the LocalFileSystems hosted directly by the
    // top-level ComputerSystem of the Filesystem Profile.
    #FSProps[] = {"CSCreationClassName", "CSName", "CreationClassName",
          "Name", "OperationalStatus", "CaseSensitive", "CasePreserved",
          "MaxFileNameLength", "FileSystemType",
          "MultipleDisksSupported",
          "LocalAccessDefinitionRequired",
          "PathNameSeparatorString" }
    $FileSystems[] = Associators($System->,
          "CIM_HostedFileSystem",
          "CIM_LocalFileSystem",
          "GroupComponent",
          "PartComponent",
          false,
          false,
          #FSProps[])

    // Step 2. Locate all the component ComputerSystems of the top level
```

```
    // ComputerSystem of the Filesystem Profile implementation.
    // This assumes that the top level ComputerSystem of the Filesystem
    // Profile is the same as the top level ComputerSystem of the
    // Multiple Computer System Subprofile.  This recipe does not
    // check if this assumption is correct.
    try {
        REF CIM_ComputerSystem $ComponentSystems->[] =
            AssociatorNames($System->,
                "CIM_ComponentCS,
                "CIM_ComputerSystem",
                "GroupComponent",
                "PartComponent")

        // Step 3. Locate the LocalFileSystems hosted by the component
        // ComputerSystem and add to the list of found LocalFileSystems.
        if ($ComponentSystems->[] != null &&
            $ComponentSystems->[].length > 0) {
                REF CIM_FileSystem $ComponentFS[]
                #fsCounter = $FileSystems[].length
                for (#i in $ComponentSystems->[]) {
                    $ComponentFS[] =
                        Associators($ComponentSystems->[#i],
                            "CIM_HostedFileSystem",
                            "CIM_LocalFileSystem",
                            "GroupComponent",
                            "PartComponent",
                            false,
                            false,
                            #FSProps[])
                    if ($ComponentFS[] != null && $ComponentFS[].length > 0) {
                        for (#j in $ComponentFS->[]) {
                            $FileSystems[#fsCounter] = $ComponentFS[#j]
                            #fsCounter++
                        }
                    }
                }
        }
    } catch (CIMException $Exception) {
        // ComponentCS may not be included in the model implemented at all if
        // the Multiple Computer System Subprofile is not supported.
        if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
            return $FileSystems[]
        }
        <ERROR! An unexpected failure occured>
    }
    return $FileSystems[]
}
```

```
// MAIN
$HostedFileSystems[] = ListFileSystems($TopLevelComputerSystem->)
```

8.6.2 Get FileSystemSettings for a FileSystem

```
// DESCRIPTION
//   Goal: Get the FileSystemSettings associated with a LocalFileSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//   2. There is only one setting for the file system
//
// FUNCTION GetFSSetting
//   This function takes a given LocalFileSystem and returns the
//   FileSystemSetting element that specifies its configuration.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem .
// OUTPUT Parameters:
//   setting: A reference to the FileSystemSetting element is returned.
// RESULT:
//   Returns: Nothing
//
sub GetFSSetting(IN REF CIM_LocalFileSystem $fs,
                 OUT CIM_FileSystemSetting  $setting)
{
    //
    // Get a reference to the FileSystemSetting associated with the
    // LocalFileSystem (via ElementSettingData association)
    $setting = Associators($fs,
                           “CIM_ElementSettingData”,
                           “CIM_FileSystemSetting”,
                           “ManagedElement”,
                           “SettingData”)->[0];
}
```

8.6.3 Get the ComputerSystem that hosts a FileSystem

```
// DESCRIPTION
//   Goal: Get the ComputerSystem that hosts a LocalFileSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//
// FUNCTION GetFileSystemHost
//   This function takes a given LocalFileSystem and returns the
//   ComputerSystem that hosts it.
// INPUT Parameters:
```

```
//   fs: A reference to the LocalFileSystem.
// OUTPUT Parameters:
//   system: A reference to the hosting ComputerSystem is returned.
// RESULT:
//   Returns: Nothing
//
sub GetFileSystemHost(IN REF CIM_LocalFileSystem $fs,
                      OUT CIM_ComputerSystem $system)
{
    $system = Associators($fs,
                          "CIM_HostedFileSystem",
                          "CIM_ComputerSystem",
                          "PartComponent",
                          "GroupComponent")->[0];
}

// Retained for backward compatability with SMI-S 1.1
sub GetFSServer(IN REF CIM_FileSystem $fs,
                OUT CIM_ComputerSystem $system)
{
     GetFileSystemHost($fs, $system);
}
```

8.6.4 Get all File Servers and Access Paths that have Local Access to this FileSystem

```
// DESCRIPTION
//   Goal: Get the file server ComputerSystems that access the
//      LocalFileSystem and the local access points on those
//      ComputerSystems
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//
// FUNCTION GetFileSystemServersAndPaths
//   This function takes a given LocalFileSystem and returns the
//   file server ComputerSystems that have local access to it
//   and the local access points on those ComputerSystems.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
// OUTPUT Parameters:
//   systems: An array of references to the file server ComputerSystems.
//   paths:   An array of strings that are the local access points on the
//            corresponding file server
// RESULT:
//   Returns: Number of entries in the returned arrays.
//
sub uint32 GetFileSystemServersAndPaths(IN REF CIM_LocalFileSystem $fs,
                                        OUT REF CIM_ComputerSystem $systems[],
```

```
                                        OUT string #paths[])
{
    REF CIM_LocalAccessAvailable $assocs->[] = References($fs,
                          "SNIA_LocalAccessAvailable",
                          "CIM_ComputerSystem",
                          "PartComponent",
                          "GroupComponent");
    #counter = 0;
    if ($assocs->[] != null && $assocs->[].length > 0) {
        #count = $assocs->[].length;
        for (#i in $assocs->[]) {
            $systems->[#counter] = $assocs->[#i].FileServer;
            #paths->[#counter] = $assocs->[#i].LocalAccessPoints;
            #counter++;
        }
    }
    return #counter;
}
```

8.6.5 Get the Access Path to this FileSystem on the specified File Server

```
// DESCRIPTION
//   Goal: Get the local access point to this LocalFileSystem on the
//      specified file server ComputerSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//   2. A reference to the file server ComputerSystem was previously
//      discovered and is defined in the $server-> variable.
//
// FUNCTION GetFileSystemServerPath
//   This function takes a given LocalFileSystem and file server
//   ComputerSystem that has access to the filesystem and returns
//   the local access point on that file server ComputerSystem.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
//   server: A reference to the file server ComputerSystem.
// OUTPUT Parameters:
//   None
// RESULT:
//   Returns: A string representing the local access path to the
//      filesystem on the file server
//
sub string GetFileSystemServerPath(IN REF CIM_FileSystem $fs,
                                   IN REF CIM_ComputerSystem $server)
{
    REF CIM_LocalAccessAvailable $assocs->[] = References($fs,
                          "SNIA_LocalAccessAvailable",
```

```
                            "CIM_ComputerSystem",
                            "PartComponent",
                            "GroupComponent");
    #path = "";
    if ($assocs->[] != null && $assocs->[].length > 0) {
        for (#i in $assocs->[]) {
            if ($server == $assocs->[#i].FileServer) {
                #path = $assocs->[#i].LocalAccessPoint;
                break;
            }
        }
    }
    return #path;
}
```

8.6.6 Get the Local Access Settings for this FileSystem on the specified File Server

```
// DESCRIPTION
//   Goal: Get the LocallyAccessibleFileSystemSetting for this
//      LocalFileSystem on the specified file server ComputerSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//   2. A reference to the file server ComputerSystem was previously
//      discovered and is defined in the $server-> variable.
//
// FUNCTION GetFileSystemServerAccessSettings
//   This function takes a given LocalFileSystem and file server
//   ComputerSystem that has access to the filesystem and returns
//   the LocallyAccessibleFileSystemSetting for that FileSystem
//   in the context of that file server ComputerSystem
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
//   server: A reference to the file server ComputerSystem.
// OUTPUT Parameters:
//   setting: A reference to the SNIA_LocallyAccessibleFileSystemSetting
// RESULT:
//   Returns: Nothing
//      (Optionally) A string containing the setting as an EmbeddedInstance
//
sub GetFileSystemServerAccessSettings(IN REF CIM_FileSystem $fs,
                                      IN REF CIM_ComputerSystem $server,
                                     OUT REF SNIA_LocallyAccessibleFileSystemSetting
                              setting)
{
    REF SNIA_LocallyAccessibleFileSystemSetting $settings->[] =
                              AssociatorNames($fs,
                             "CIM_ElementSettingData",
```

```
                                "SNIA_LocallyAccessibleFileSystemSetting",
                                "ManagedElement",
                                "SettingData");
        $setting = NULL;
        $settingEI = "";
        if ($settings->[] != null && $settings->[].length > 0) {
            for (#i in $settings->[]) {
                // Find the server that scopes this setting; assumes at least one is
                                returned
                REF CIM_ComputerSystem scope = AssociatorNames($settings->[#i],
                                                "CIM_ScopedSetting",
                                                "CIM_ComputerSystem",
                                                "ScopedSettingData,
                                                "ManagedElement")->[0];
                if ($server == $scope) {
                    $setting = $settings->[#i];
                    $settingEI = $setting->GetInstance();
                    break;
                }
            }

        } else {
            // There is no setting => it is defaulted by the server and opaque to the
                                client
            // Is this an Error?
            #ERROR("Cannot find LocallyAccessibleFileSystemSetting for
                                LocalFileSystem.");
        }
        return $settingEI;
    }
```

8.6.7 Get the FileShares and shared File path of this FileSystem on all File Servers

```
    // DESCRIPTION
    //   Goal: Get all the FileShare elements and filesystem-relative
    //      path to the shared file or directory of this LocalFileSystem
    //      on all file server ComputerSystems (that
    //      support local access to this LocalFileSystem)
    //
    // PRE-EXISTING CONDITIONS AND ASSUMPTIONS
    //   1. A reference to the LocalFileSystem was previously
    //      discovered and is defined in the $fs-> variable.
    //
    // FUNCTION GetFileSystemServersSharesAndSharedPaths
    //   This function takes a given LocalFileSystem and returns the
    //   FileShare elements that provide access to a file or directory
    //   of the FileSystem.  For each FileShare, this also returns
    //   the file server ComputerSystems that provides local access to
    //   it and the path to the shared file or directory relative to the
```

```
//   filesystem.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
// OUTPUT Parameters:
//   shares:  An array of references to the FileShares that provide access.
//   servers: An array of references to the file server ComputerSystems.
//   dirpaths:   An array of strings that are the filesystem-relative paths
//       to the shared directory or file
// RESULT:
//   Returns: Number of entries in the returned arrays.
//
sub uint32 GetFileSystemServersSharesAndSharedPaths(
              IN REF CIM_FileSystem $fs,
              OUT REF CIM_FileShare $shares[],
              OUT string #dirpaths[],
              OUT REF CIM_ComputerSystem $servers[])
{
    REF CIM_FileShares $shares->[] = Associators($fs,
                         "CIM_SharedElement",
                         "CIM_FileShare",
                         "SystemElement",
                         "SameElement");
    #counter = 0;
    if ($shares->[] != null && $shares->[].length > 0) {
        for (#i in $shares->[]) {
            // A share must be hosted
            $servers->[#counter] = AssociatorNames($shares->[#i],
                         "CIM_HostedShare",
                         "CIM_ComputerSystem",
                         "PartComponent",
                         "GroupComponent")->[0];
            $assoc = References($shares->[#i],
                         "CIM_SharedElement",
                         "CIM_FileSystem",
                         "SameElement",
                         "SystemElement")->[0];
            $dirpaths[#counter] = $assoc.PathName;
            #counter++;
        }
    }
    return #counter;
}
```

8.6.8 Get the FileShares and shared File path of this FileSystem on the specified FileServer

```
// DESCRIPTION
//   Goal: Get all the FileShare elements and filesystem-relative
//      path to the shared file or directory of this LocalFileSystem
//      on this file server ComputerSystem
```

```
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. A reference to the LocalFileSystem was previously
//      discovered and is defined in the $fs-> variable.
//   2. A reference to the file server ComputerSystem was previously
//      discovered and is defined in the $server-> variable.
//
// FUNCTION GetFileSystemSharesAndSharedPathsOnServer
//   This function takes a given LocalFileSystem and returns the
//   FileShare elements that provide access to a file or directory
//   of the FileSystem.  For each FileShare this also returns the
//   file server ComputerSystem that supports local access to it
//   and the filesystem-relative path to the shared file or directory.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
//   server: A reference to the file server ComputerSystem.
// OUTPUT Parameters:
//   shares:  An array of references to the FileShares that provide access.
//   dirpaths:   An array of strings that are the file system-relative paths
//       to the shared directory or file
// RESULT:
//   Returns: Number of entries in the returned arrays.
//
sub uint32 GetFileSystemSharesAndSharedPathsOnServer(
              IN REF CIM_FileSystem $fs,
              IN REF CIM_ComputerSystem $server,
              OUT REF CIM_FileShare $shares[],
              OUT string #dirpaths[])
{
    REF CIM_FileShares $allshares->[] = Associators($fs,
                          “CIM_SharedElement”,
                          “CIM_FileShare”,
                          “SystemElement”,
                          “SameElement”);
    #counter = 0;
    if ($allshares->[] != null && $allshares->[].length > 0) {
        for (#i in $shares->[]) {
            // A share must be hosted
            $host = AssociatorNames($allshares->[#i],
                          “CIM_HostedShare”,
                          “CIM_ComputerSystem”,
                          “PartComponent”,
                          “GroupComponent”)->[0];
            // Is this share hosted by the server?
            if ($host == $server) {
                $assoc = References($allshares->[#i],
                          “CIM_SharedElement”,
```

```
                                    "CIM_FileSystem",
                                    "SameElement",
                                    "SystemElement")->[0];
                            $shares[#counter] = $allshares->[#i];
                            $dirpaths[#counter] = $assoc.PathName;
                            #counter++;
                        }
                    }
                }
                return #counter;
            }
```

EXPERIMENTAL

8.7 Registered Name and Version

Filesystem version 1.3.0

8.8 CIM Elements

Table 85 describes the CIM elements for Filesystem.

Table 85 - CIM Elements for Filesystem

Element Name	Requirement	Description
8.8.1 CIM_Dependency (Uses Directory Services From)	Conditional	Conditional requirement: Required if LocalFileSystem.DirectoryServiceUsage is either "Required" or "Optional". Associates a ComputerSystem that indicates a directory service that supports the dependent LocalFileSystem.
8.8.2 CIM_ElementSettingData (FileSystem)	Optional	Associates a LocalFileSystem to its FileSystemSetting element.
8.8.3 CIM_ElementSettingData (Local Access Required)	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.Associates a LocalFileSystem to LocallyAccessibleFileSystemSetting elements, one for each file server that has local access.
8.8.4 CIM_FileStorage	Mandatory	Associates a LogicalFile (or Directory) to the LocalFileSystem that contains it. This is provided for backward compatibility with previous versions of SMI-S.
8.8.5 CIM_FileSystemSetting	Optional	This element represents the configuration settings of a filesystem represented by a LocalFileSystem.

Table 85 - CIM Elements for Filesystem

Element Name	Requirement	Description
8.8.6 CIM_HostedDependency (Local Access Required)	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true. Associates a file server ComputerSystem to the LocallyAccessibleFileSystemSetting elements that get scoping information from that file server.
8.8.7 CIM_HostedFileSystem (LocalFileSystem)	Mandatory	Associates a LocalFileSystem to the ComputerSystem that hosts it.
8.8.8 CIM_LocalFileSystem	Mandatory	Represents a filesystem in a Filesystem related profile.
8.8.9 CIM_LogicalFile	Mandatory	In an earlier release of SMI-S, the Filesystem related profiles made a limited set of LogicalFiles (or Directory subclass) instances visible (these were any file or directory that was exported as a share. This element is required by the profiles to maintain backward compatibility for clients conforming to earlier versions of SMI-S.
8.8.10 SNIA_LocalAccessAvailable	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.Associates a LocalFileSystem to a file server ComputerSystem that can export files or directories as shares.
8.8.11 SNIA_LocalFileSystem	Optional	Represents a filesystem in a Filesystem related profile.
8.8.12 SNIA_LocallyAccessibleFileSystemSetting	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.This element represents the configuration settings of a LocalFileSystem that can be made locally accessible (i.e., can have a file or directory made accessible to operational users) from a file server ComputerSystem. This Setting provides further details on the functionality supported and the parameters of that functionality when locally accessible.

Table 85 - CIM Elements for Filesystem

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LocalFileSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a Filesystem. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LocalFileSystem AND SourceInstance.CIM_LocalFileSystem::OperationalStatus <> PreviousInstance.CIM_LocalFileSystem::OperationalStatus	Optional	CQL -Change of Status of a Filesystem. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

8.8.1 CIM_Dependency (Uses Directory Services From)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Required if LocalFileSystem.DirectoryServiceUsage is either "Required" or "Optional".

Table 86 describes class CIM_Dependency (Uses Directory Services From).

Table 86 - SMI Referenced Properties/Methods for CIM_Dependency (Uses Directory Services From)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The ComputerSystem that indicates the directory service(s) that support user, group and other security principal identities for a filesystem.
Dependent		Mandatory	The LocalFileSystem whose use of user, group, and other security principal identities is supported by the antecedent ComputerSystem.

8.8.2 CIM_ElementSettingData (FileSystem)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Optional

Table 87 describes class CIM_ElementSettingData (FileSystem).

Table 87 - SMI Referenced Properties/Methods for CIM_ElementSettingData (FileSystem)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The LocalFileSystem.
SettingData		Mandatory	The settings established on the LocalFileSystem when first created or as modified.

8.8.3 CIM_ElementSettingData (Local Access Required)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.

Table 88 describes class CIM_ElementSettingData (Local Access Required).

Table 88 - SMI Referenced Properties/Methods for CIM_ElementSettingData (Local Access Required)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The LocalFileSystem that is being made locally accessible.
SettingData		Mandatory	The local access settings of the LocalFileSystem, specified when first created or established later.

8.8.4 CIM_FileStorage

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 89 describes class CIM_FileStorage.

Table 89 - SMI Referenced Properties/Methods for CIM_FileStorage

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The LocalFileSystem that contains the LogicalFile.
PartComponent		Mandatory	The LogicalFile contained in the LocalFileSystem.

8.8.5 CIM_FileSystemSetting

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 90 describes class CIM_FileSystemSetting.

Table 90 - SMI Referenced Properties/Methods for CIM_FileSystemSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for this FileSystemSetting element.
ElementName		Mandatory	A user-friendly name for this FileSystemSetting element.
ActualFileSystemType		Mandatory	This identifies the type of filesystem that this FileSystemSetting represents.
FilenameCaseAttributes		Mandatory	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		Mandatory	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
NumberOfObjectsMin		Mandatory	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjectsMax		Mandatory	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjects		Mandatory	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		Mandatory	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		Mandatory	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMax		Mandatory	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
FilenameReservedCharacterSet		Optional	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames of a filesystem with this setting.
DataExtentsSharing		Optional	This specifies whether a filesystem with this setting supports the creation of data blocks (or storage extents) that are shared between files.

Table 90 - SMI Referenced Properties/Methods for CIM_FileSystemSetting

Properties	Flags	Requirement	Description & Notes
CopyTarget		Optional	This specifies that, if possible, support should be provided for using a filesystem created with this setting as a target of a Copy operation.
FilenameStreamFormats		Optional	This is an array that specifies the stream formats (e.g., UTF-8) supported for filenames by a filesystem with this setting.
FilenameFormats		Optional	This is an array that specifies the formats (e.g. DOS 8.3 names) supported for filenames by a filesystem with this setting.
FilenameLengthMax		Optional	This specifies the maximum length of a filename supported by a filesystem with this setting.
SupportedLockingSemantics		Optional	This array specifies the set of file access/locking semantics supported by a filesystem with this setting.
SupportedAuthorizationProtocols		Optional	This array specifies the kind of file authorization protocols supported by a filesystem with this setting.
SupportedAuthenticationProtocols		Optional	This array specifies the kind of file authentication protocols supported by a filesystem with this setting.

8.8.6 CIM_HostedDependency (Local Access Required)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.

Table 91 describes class CIM_HostedDependency (Local Access Required).

Table 91 - SMI Referenced Properties/Methods for CIM_HostedDependency (Local Access Required)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The ComputerSystem that provides the scope for the LocallyAccessibleFileSystemSetting.
Dependent		Mandatory	The local access settings of the LocalFileSystem, established when first created or as modified later, that is dependent on some information provided by the file server that is the scoping ComputerSystem.

8.8.7 CIM_HostedFileSystem (LocalFileSystem)

Created By: External

Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 92 describes class CIM_HostedFileSystem (LocalFileSystem).

Table 92 - SMI Referenced Properties/Methods for CIM_HostedFileSystem (LocalFileSystem)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that hosts a LocalFileSystem.
PartComponent		Mandatory	The LocalFileSystem that represents the hosted filesystem.

8.8.8 CIM_LocalFileSystem

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 93 describes class CIM_LocalFileSystem.

Table 93 - SMI Referenced Properties/Methods for CIM_LocalFileSystem

Properties	Flags	Requirement	Description & Notes
CSCreationClassName		Mandatory	The CIM class of the hosting ComputerSystemelement.
CSName		Mandatory	The Name property of the hosting ComputerSystem element.
CreationClassName		Mandatory	The CIM class of this LocalFileSystem element.
Name		Mandatory	A unique name for this LocalFileSystem element in the context of the hosting ComputerSystem.
OperationalStatus		Mandatory	The current operational status of the filesystem represented by this LocalFileSystem element.
Root		Optional	A path that specifies the "mount point" of the filesystem in an unitary computer system that is both the host of the filesystem and is the file server that makes it available.
BlockSize		Optional	The size of a block in bytes for certain filesystem types that require a fixed block size when creating a filesystem.
FileSystemSize		Optional	The total current size of the filesystem in blocks.
AvailableSpace		Optional	The space available currently in the filesystem in blocks. NOTE: This value is an approximation as it can vary continuously when the filesystem is in use.

Table 93 - SMI Referenced Properties/Methods for CIM_LocalFileSystem

Properties	Flags	Requirement	Description & Notes
ReadOnly		Optional	Indicates that this is a read-only filesystem that does not allow modifications to contained files and directories.
EncryptionMethod		Optional	Indicates if files are encrypted by the filesystem implementation and the method of encryption.
CompressionMethod		Optional	Indicates if files are compressed by the filesystem implementation before being stored, and the methods of compression.
CaseSensitive		Mandatory	Whether this filesystem is sensitive to the case of characters in filenames.
CasePreserved		Mandatory	Whether this filesystem implementation preserves the case of characters in filenames when saving and restoring.
CodeSet		Optional	The codeset used in filenames by the filesystem implementation.
MaxFileNameLength		Mandatory	The length of the longest filename supported by the filesystem implementation.
FileSystemType		Mandatory	This is a string that matches FileSystemSetting.ActualFileSystemType property used to create the filesystem.
NumberOfFiles		Optional	The actual current number of files in the filesystem. NOTE: This value is an approximation as it can vary continuously when the filesystem is in use.

8.8.9 CIM_LogicalFile

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 94 describes class CIM_LogicalFile.

Table 94 - SMI Referenced Properties/Methods for CIM_LogicalFile

Properties	Flags	Requirement	Description & Notes
CSCreationClassName		Mandatory	Class Name of the ComputerSystem that hosts the filesystem containing this file.
CSName		Mandatory	The Name property of the ComputerSystem that hosts the filesystem containing this file.
FSCreationClassName		Mandatory	Class Name of the LocalFileSystem that represents the filesystem containing this file.

Table 94 - SMI Referenced Properties/Methods for CIM_LogicalFile

Properties	Flags	Requirement	Description & Notes
FSName		Mandatory	The Name property of the LocalFileSystem that represents the filesystem containing this file.
CreationClassName		Mandatory	Class Name of this instance of LogicalFile that represents the file.
Name		Mandatory	The Name property of the LogicalFile that represents the file.
ElementName		Mandatory	The pathname from the root of the containing LocalFileSystem to this LogicalFile. The root of the LocalFileSystem is indicated if this is NULL or the empty string. The format of the pathname is specific to the LocalFileSystem's FileSystemType. If it is a sequence of directories from the root, the separator string is specified by the SNIA_LocalFileSystem.PathNameSeparatorString property.

8.8.10 SNIA_LocalAccessAvailable

Created By: External
Modified By: Static
Deleted By: External
Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.

Table 95 describes class SNIA_LocalAccessAvailable.

Table 95 - SMI Referenced Properties/Methods for SNIA_LocalAccessAvailable

Properties	Flags	Requirement	Description & Notes
LocalAccessPoint		Optional	The name used by the file server ComputerSystem to identify the filesystem. Sometimes referred to as a mount-point. For many UNIX-based systems, this will be a qualified full pathname. For Windows systems this could also be the drive letter used for the LogicalDisk that the filesystem is resident on.
FileSystem		Mandatory	The LocalFileSystem that is being made available to the file server ComputerSystem.
FileServer		Mandatory	The ComputerSystem that will be able to export shares from this LocalFileSystem.

8.8.11 SNIA_LocalFileSystem

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 96 describes class SNIA_LocalFileSystem.

Table 96 - SMI Referenced Properties/Methods for SNIA_LocalFileSystem

Properties	Flags	Requirement	Description & Notes
LocalAccessDefinitionRequired		Mandatory	This boolean property indicates whether or not this LocalFileSystem must be made locally accessible ("mounted") from a file server ComputerSystem before it can be shared or otherwise made available to operational clients.
PathNameSeparatorString		Mandatory	This indicates the string of characters used to separate directory components of a canonically formatted path to a file from the root of the filesystem. This string is expected to be specific to the ActualFileSystemType and so is vendor/ implementation dependent. However, by surfacing it we make it possible for a client to parse a pathname into the hierarchical sequence of directories that compose it.
DirectoryServiceUsage		Optional	This enumeration indicates whether the filesystem supports security principal information and therefore requires support from a file server that uses one or more directory services. If the filesystem requires such support, there must be a concrete subclass of Dependency between the LocalFileSystem element and the specified file server ComputerSystem. The values supported by this property are: "Not Used" indicates that the filesystem will not support security principal information and so will not require support from a directory service. "Optional" indicates that the filesystem may support security principal information. If it does, it will require support from a directory service and the Dependency association described above must exist. "Required" indicates that the filesystem supports security principal information and will require support from a directory service. The Dependency association described above must exist.

8.8.12 SNIA_LocallyAccessibleFileSystemSetting

Created By: External
Modified By: External
Deleted By: External

Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.

Table 97 describes class SNIA_LocallyAccessibleFileSystemSetting.

Table 97 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for a LocallyAccessibleFileSystemSetting.
ElementName		Mandatory	A user-friendly name for this LocallyAccessibleFileSystemSetting element.
InitialEnabledState		Optional	InitialEnabledState is an integer enumeration that indicates the enabled/disabled states initially set for a locally accessible filesystem (LAFS). The element functions by passing commands onto the underlying filesystem, and so cannot indicate transitions between requested states because those states cannot be requested. The following text briefly summarizes the various enabled/disabled initial states: Enabled (2) indicates that the element will execute commands, will process any queued commands, and will queue new requests. Disabled (3) indicates that the element will not execute commands and will drop any new requests. In Test (7) indicates that the element will be in a test state. Deferred (8) indicates that the element will not process any commands but will queue new requests. Quiesce (9) indicates that the element is enabled but in a restricted mode. The element's behavior is similar to the Enabled state, but it only processes a restricted set of commands. All other requests are queued.
OtherEnabledState		Optional	A string describing the element's initial enabled/disabled state when the InitialEnabledState property is set to 1 ("Other"). This property MUST be set to NULL when InitialEnabledState is any value other than 1.
FailurePolicy		Optional	An enumerated value that specifies if the operation to make a filesystem locally accessible to a scoping ComputerSystem should be attempted one or more times in the foreground or tried repeatedly in the background until it succeeds. The number of attempts would be limited by the corresponding RetriesMax property of the setting.
RetriesMax		Optional	An integer specifying the maximum number of attempts that should be made by the scoping ComputerSystem to make a LocalFileSystem locally accessible. A value of '0' specifies an implementation-specific default.

Table 97 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
RequestRetryPolicy		Optional	An enumerated value representing the policy that is supported by the operational file server on a request to the operational filesystem that either failed or left the file server hanging. If the request is being performed in the foreground, the options are to try once and fail if a timeout happens, or, to try repeatedly. If the request can be performed in the background, the request will be tried repeatedly until stopped.
TransmissionRetries Max		Optional	An integer specifying the maximum number of retransmission attempts to be made from the operational file server to the operational filesystem when the transmission of a request fails or makes the file server hang. A value of '0' specifies an implementation-specific default. This is only relevant if there is a transmission channel between the file server and the underlying filesystem.
RetransmissionTime outMin		Optional	An integer specifying the minimum number of milliseconds that the operational file server must wait before assuming that a request to the operational filesystem has failed. '0' indicates an implementation-specific default. This is only relevant if there is a transmission channel between the operational file server and the operational filesystem.
CachingOptions		Optional	An enumerated value that specifies if a local cache is supported by the operational file server when accessing the underlying operational filesystem.
BuffersSupport		Optional	An array or enumerated values that specifies the buffering mechanisms supported by the operational file server for accessing the underlying operational filesystem." If supported, other properties will establish the level of support. If the property is NULL or the empty array, buffering is not supported.
ReadBufferSizeMin		Optional	An integer specifying the minimum number of bytes that must be allocated to each buffer used for reading. A value of '0' specifies an implementation-specific default.
ReadBufferSizeMax		Optional	An integer specifying the maximum number of bytes that may be allocated to each buffer used for reading. A value of '0' specifies an implementation-specific default.
WriteBufferSizeMin		Optional	An integer specifying the minimum number of bytes that must be allocated to each buffer used for writing. A value of '0' specifies an implementation-specific default.
WriteBufferSizeMax		Optional	An integer specifying the maximum number of bytes that may be allocated to each buffer used for writing. A value of '0' specifies an implementation-specific default.

Table 97 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
AttributeCaching		Optional	An array of enumerated values that specify whether attribute caching is (or is not) supported by the operational file server when accessing specific types of objects from the underlying operational filesystem. The object type and the support parameters are specified in the corresponding AttributeCachingObjects, AttributeCachingTimeMin, and AttributeCachingTimeMax array properties. Object types contained by a filesystem that can be accessed locally are represented by an entry in these arrays. The entry in the AttributeCaching array can be 'On', 'Off', or 'Unknown'. Implementation of this feature requires support from other system components, so it is quite possible that specifying 'On' may still not result in caching behavior. 'Unknown' indicates that the access operation will try to work with whatever options the operational file server and filesystem can support. In all cases, AttributeCachingTimeMin and AttributeCachingTimeMax provide the minimum and maximum time for which the attributes can be cached. When this Setting is used as a Goal, the client may specify 'Unknown', but the Setting in the created object should contain the supported setting, whether 'On' or 'Off'.
AttributeCachingObjects		Optional	An array of enumerated values that specify the attribute caching support provided to various object types by the operational file server when accessing the underlying operational filesystem. These", types represent the types of objects stored in a filesystem -- files and directories as well as others that may be defined in the future. The corresponding properties, AttributeCaching, AttributeCachingTimeMin, and AttributeCachingTimeMax provide the supported features for the type of object. 'None' and 'All' cannot both be specified; if either one is specified, it must be the first entry in the array and the entry is interpreted as the default setting for all objects. If neither 'None' or 'All' are specified, the caching settings for other objects are defaulted by the implementation. If 'Rest' is specified, the entry applies to all known object types other than the named ones. If 'Unknown' is specified it applies to object types not known to this application (this can happen when foreign filesystems are made locally accessible).
AttributeCachingTimeMin		Optional	An array of integers specifying, in milliseconds, the minimum time for which an object of the type specified by the corresponding AttributeCaching property must be retained in the attribute cache. When used as a Goal, a value of '0' indicates an implementation-specific default.

Table 97 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
AttributeCachingTimeMax		Optional	An array of integers specifying, in milliseconds, the maximum time for which an object of the type specified by the corresponding AttributeCaching property must be retained in the attribute cache. When used as a Goal, a value of '0' indicates an implementation-specific default.
ReadWritePolicy		Optional	An enumerated value that specifies the Read-Write policy set on the operational filesystem and supported by the operational file server when accessing it. 'Read Only' specifies that the access to the operational filesystem by the operational file server is set up solely for reading. 'Read/Write' specifies that the access to the operational filesystem by the operational file server is set up for both reading and writing. 'Force Read/Write' specifies that 'Read-Only' has been overridden by a client with write access to the operational filesystem. This option is intended for use when the associated filesystem has been made 'Read Only' by default, as might happen if it were created to be the target of a Synchronization or Mirror operation.
LockPolicy		Optional	An enumerated value that specifies the Locking that will be enforced on the operational filesystem by the operational file server when accessing it. 'Enforce None' does not enforce locks. 'Enforce Write' does not allow writes to locked files. 'Enforce Read/Write' does not allow reads or writes to locked files.
EnableOnSystemStart		Optional	An enumerated value that specifies if local access from the operational file server to the operational filesystem should be enabled when the file server is started.
ReadWritePref		Optional	An instance of a CIM_Privilege, encoded as a string, that expresses the client's expectations about access to elements contained in the operational filesystem. The provider is expected to surface this access using the CIM privilege model.

Table 97 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
ExecutePref		Optional	An enumerated value that specifies if support should be provided on the operational file server for executing elements contained in the operational filesystem accessed through this local access point. This may require setting up specialized paging or execution buffers either on the operational file server or on the operational filesystem side (as appropriate for the implementation). Note that this does not provide any rights to actually execute any element but only specifies support for such execution, if permitted.
RootAccessPref		Optional	An instance of a CIM_Privilege, encoded as a string, that expresses the client's expectations about privileged access by appropriately privileged System Administrative users on the operational file server ('root' or 'superuser') to the operational filesystem and its elements. The provider is expected to surface this access using the CIM privilege model. Support for the privileged access might require setup at both the operational file server as well as the operational filesystem, so there is no guarantee that the request can be satisfied.

STABLE

EXPERIMENTAL

Clause 9: Filesystem Manipulation Subprofile

9.1 Description

9.1.1 Synopsis

Profile Name: Filesystem Manipulation

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: FileSystemConfigurationService

Scoping Class: ComputerSystem

9.1.2 Overview

The Filesystem Manipulation Profile is a subprofile provides support for configuring and manipulating filesystems in the context of Filesystem Profiles (currently consisting of the NAS Head and the Self Contained NAS profiles). A number of other profiles and subprofiles make use of elements of the Filesystem profiles and will be referred to in this specification as “Filesystem related profiles” -- these include but are not limited to the Filesystem subprofile, File Export subprofile, File Export Manipulation subprofile, NAS Head profile, and so on.

The state transitions involved when an appropriate storage element is transformed into a filesystem are described in Annex A: (Informative) State Transitions from Storage to File Shares.

9.1.2.1 Backward Compatibility Note

This profile has seen some incompatible changes from SMI-S 1.1. It is still "Experimental". Three major changes to the methods CreateFileSystem and ModifyFileSystem are intended to accommodate requirements from a proposed Hosted Filesystem profile (now postponed to a future release of SMI-S) and to support the local access ("mount") related changes. First, SMI-S now allows a LocalFileSystem to be built at the same time that the LogicalDisk(s) are built; previously, a LogicalDisk had to be built first in an independent operation; second, multiple LogicalDisks can be specified in the method parameters and these are combined into a single LogicalDisk using the Volume Composition subprofile -- the old methods only supported a single LogicalDisk, which is still supported as a special case of the new method. Third, SMI-S now supports parameters that make the LocalFileSystem immediately available locally (i.e., "mount"-ed) at a File Server-provided pathname; the previous version assumed that this would be done in a vendor-specific default. Both these extensions in functionality are optional on new properties specified in the FileSystemSetting and LocalFileSystem, and the SMI-S 1.1 behavior is supported by the default values of these properties.

9.1.3 Instance Diagrams

9.1.3.1 Filesystem Creation classes and associations

Figure 13 illustrates the constructs involved with creating a LocalFileSystem for a Filesystem Profile. This summarizes the mandatory classes and associations for this subprofile. Specific areas are discussed in later sections.

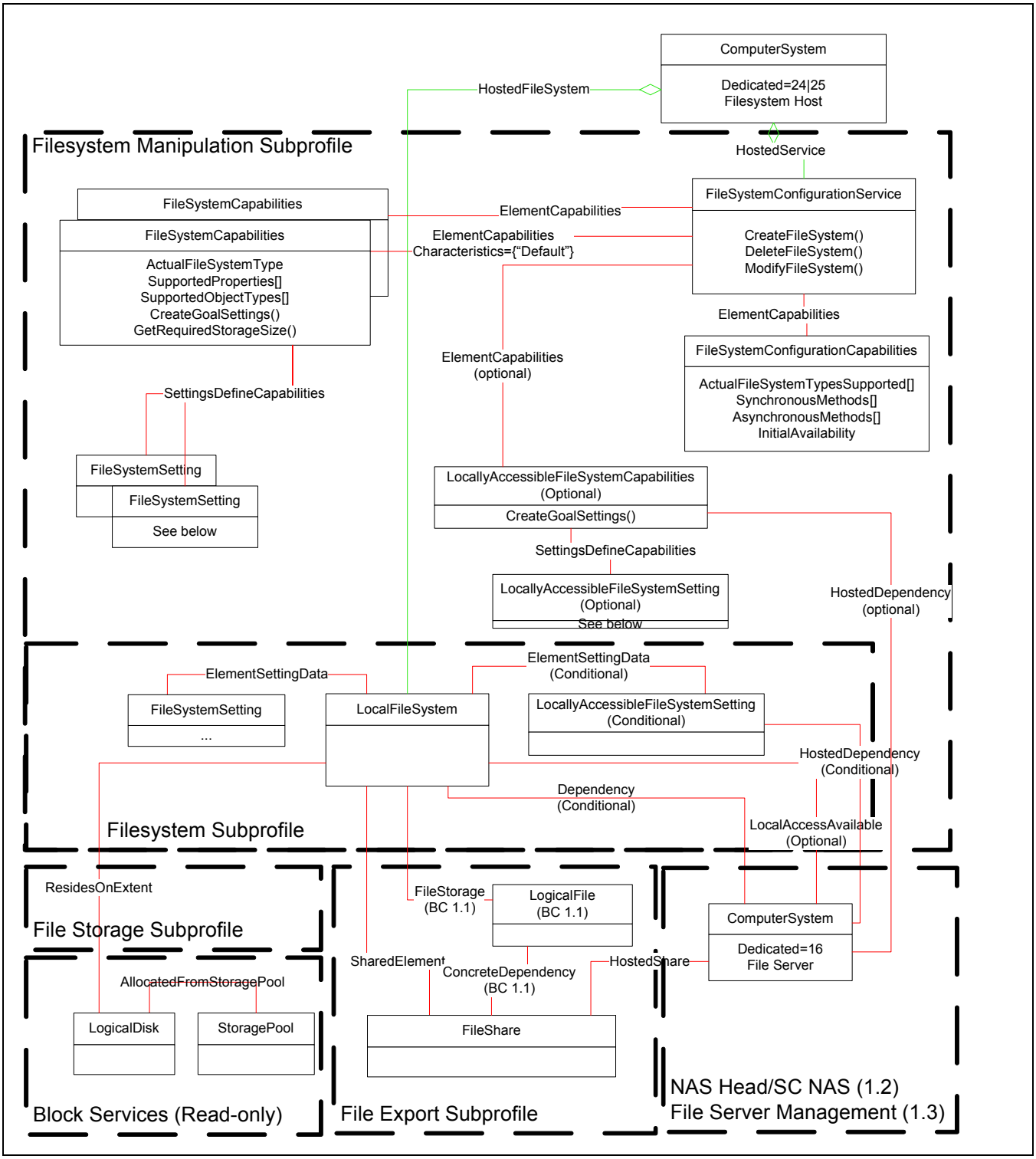

Figure 13 - LocalFileSystem Creation Instance Diagram

If a Filesystem-related Profile supports the Filesystem Manipulation Subprofile, it shall have at least one instance of the FileSystemConfigurationService. This service shall be hosted on the top level ComputerSystem of the Filesystem-related Profile. The methods offered are CreateFileSystem, ModifyFileSystem, and DeleteFileSystem.

Associated to the FileSystemConfigurationService (via ElementCapabilities) shall be one instance of FileSystemConfigurationCapabilities that describes the capabilities of the service. It identifies the methods supported, whether the methods support Job Control or not, the types of filesystems that are supported, and whether or not the filesystem shall be made operationally available after creation.

For each type of filesystem that can be created, there shall be one FileSystemCapabilities element that defines the range of capabilities supported for that particular filesystem type. An ElementCapabilities association links each FileSystemCapabilities to the FileSystemConfigurationService. One of these FileSystemCapabilities may also be identified as a default capability (by setting “Default” as one of the entries in the array property Characteristics of its ElementCapabilities association). This default FileSystemCapabilities element is used when the client does not specify a goal element when requesting the CreateFileSystem method. The default FileSystemCapabilities element implicitly indicates the default filesystem type to be used for creation.

For the convenience of clients a Filesystem Manipulation profile may populate a set of “pre-defined” FileSystemSettings for each of the FileSystemCapabilities. These shall be associated to the FileSystemCapabilities via the SettingsDefineCapabilities association (the association must have its SettingsDefineCapabilities.PropertyPolicy property set to "Correlated" and the SettingsDefineCapabilities.ValueRole property must include "Supported" as an entry) and shall be for the same filesystem type as the associated capabilities element (same value for the ActualFileSystemType property in both classes).

Note: That they are pre-defined and exist at all times does not imply that these FileSystemSettings must be made persistent by the implementation -- rather it should be possible for the implementation to regenerate them if requested, though a simple re-generating implementation may not necessarily scale.

The FileSystemCapabilities element supports two methods: CreateGoalSettings and GetRequiredStorageSize. These methods are described in detail in 9.5.1, "Extrinsic Methods of the Profile", but their basic function is to establish at least one client-approved FileSystemSettings element (as a goal) and to determine the size of the LogicalDisk required to support the desired Filesystem.

CreateGoalSettings takes an array of embedded-instance SettingData elements as the input TemplateGoalSettings and SupportedGoalSettings parameters and can generate an array of embedded-instance SettingData elements as the output SupportedGoalSettings parameter. However, in this profile, SMI-S only uses a single embedded-instance FileSystemSetting element in the input parameters (both TemplateGoalSettings and SupportedGoalSettings) and generate a single valid embedded-instance FileSystemSetting element as output (SupportedGoalSettings). If a client supplies a NULL (or the empty string) FileSystemSetting as input to this method, the returned FileSystemSetting embedded-instance shall be a default setting for the ActualFileSystemType of the FileSystemCapabilities. If the input (the embedded-instance FileSystemSetting element) is not NULL, the method may return a “best fit” to the requested setting. The client may iterate on this method until it acquires a setting that suits its needs. This embedded-instance settings structure may be used when the CreateFileSystem or ModifyFileSystem methods are invoked. The details of how iterative negotiation can work are discussed in 9.5.1.1, "FileSystemCapabilities.CreateGoalSettings". Note that the FileSystemType remains unchanged in all of these interactions. It is an error if the client or server changes the FileSystemType unilaterally.

Note: It is not possible to guarantee that negotiation will terminate with an agreed upon setting and a fall-back mechanism is needed. This profile does not require negotiation -- an implementation may support only a set of pre-defined correlated point settings that a client can preload and use without modification. The implementation could also support only settings whose properties are selectable from an arithmetic progression or from a fixed enumeration, so that the client may construct a goal setting that is guaranteed to be supported without negotiation.

Note: That a client has obtained a goal setting supported by the implementation does not guarantee that a create or modify request will succeed. Such a setting only specifies a supported static configuration not that the current dynamic environment has the resources to implement a specific request.

After obtaining a goal FileSystemSetting, the next step is to determine the LogicalDisk size required to support the FileSystemSetting. This is done by invoking the FileSystemCapabilities.GetRequiredStorageSize method of this subprofile. The inputs are the embedded-instance FileSystemSetting structure and an embedded-instance StorageSetting structure that describes the quality of service the client wants for the storage (e.g., data redundancy, package redundancy, etc.). The method returns three numbers corresponding to the StorageSetting: the expected size, the minimum size, and a maximum usable size. The client would use these numbers in specifying or evaluating the appropriate LogicalDisk(s) on which to create the FileSystem. The method also returns as output the actual StorageSetting used as an EmbeddedInstance structure (assuming that these can be substituted for the input StorageSetting).

Note: This profile recommends that GetRequiredStorageSize() be used only if a LocalFileSystem is to be created on a single LogicalDisk. If the intent is to use more than one LogicalDisk for the LocalFileSystem, this profile recommends using the CreateFileSystem method to make the implementation create or select the LogicalDisks to use.

- Armed with the FileSystem goal (the embedded-instance FileSystemSetting structure) and one or more LogicalDisks, the client can now use the CreateFileSystem method to create the filesystem. The CreateFileSystem method creates a LocalFileSystem instance, and a new FileSystemSetting instance as well as several necessary associations. These associations are:
- HostedFileSystem association between the LocalFileSystem and the ComputerSystem that hosts it
- ResidesOnExtent association between the Filesystem and one of the LogicalDisk(s) for the Filesystem data

  Note: Even when multiple LogicalDisks are created or used for the LocalFileSystem, only one LogicalDisk will have the ResidesOnExtent association.

- ElementSettingData to associate the Filesystem to the FileSystemSetting defined for it

CreateFileSystem allows LogicalDisks to be obtained from a number of StoragePools using an array of embedded-instance StorageSettings. The CreateFileSystem implementation must use the capabilities of the StoragePools (and the associated StorageConfigurationService) to create the necessary LogicalDisks. The LogicalDisks used for this purpose are returned as output values for the InExtents parameter.

If the property FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that CreateFileSystem method provides the optional parameters for establishing local access ("mounting") from file server ComputerSystems, the LocalFileSystem.LocalAccessDefinitionRequired will be set to true and the LocalFileSystem will need to be made locally accessible from the specified file server ComputerSystems. The following elements are created:

- A LocalAccessAvailable association representing local access by a file server ComputerSystem to the LocalFileSystem from within its namespace is created using the provided (or defaulted) LocallyAccessibleFileSystemSetting parameter (as an EmbeddedInstance parameter). This association goes between the File Server ComputerSystem and the LocalFileSystem; its LocalAccessPoint property specifies the local access point (or "mount-point") in the file server ComputerSystem.
- An instance of LocallyAccessibleFileSystemSetting is optionally created and associated to:
  - The LocalFileSystem via an optional ElementSettingData association.
  - The file server ComputerSystem via an optional HostedDependency (ScopedSetting) association (this represents that a LocalFileSystem could be mounted on many file server ComputerSystems with different "mount" parameters, so these parameters are specific to the pair of LocalFileSystem and file server ComputerSystem).
- For backward compatibility with the SMI-S 1.1 Filesystem subprofile:
  - The root directory of the LocalFileSystem is represented as a LogicalFile

- A FileStorage association is created between the LogicalFile and the LocalFileSystem

The DeleteFileSystem method is straightforward -- it deletes the LocalFileSystem and the FileSystemSetting**,** and the associations to those instances (HostedFileSystem, both ElementSettingData elements, ResidesOnExtent, LocalAccessAvailable, and LocallyAccessibleFileSystemSetting). Any created LogicalFiles associated to the LocalFileSystem via FileStorage will also be deleted as a side-effect of deleting the LocalFileSystem (so there is no separate requirement necessary for backward compatibility to the SMI-S 1.1 Filesystem subprofile). The implementation may delete the LogicalDisk(s), however, this is not required by this profile. If the LogicalDisk(s) are not deleted, they become available for use in another CreateFileSystem operation.

The ModifyFileSystem method modifies an existing LocalFileSystem -- this requires a new FileSystemSetting structure to be used as a goal. But not any FileSystemSetting structure will do -- the client must use one created with the same FileSystemCapabilities.CreateGoalSettings method that would have been used to create the Filesystem, or an appropriate compatible FileSystemCapabilities instance. The CreateGoalSettings method is used to establish a new FileSystemSetting goal (as with the original Filesystem creation, it may be necessary to iterate on the CreateGoalSettings method). Since only properties controlled by Settings can be changed by ModifyFileSystem (i.e., the LogicalDisk(s) already created cannot be changed, though new ones can be created and/or added), the effect of ModifyFileSystem is to change some properties of the LocalFileSystem or of the associated FileSystemSetting.

Note: Depending on what property is being modified, it may also be necessary to invoke the GetRequiredStorageSize method to verify that the current LogicalDisk still supports the new goals.

9.1.3.1.1 Dependency on support for Locally Accessible Filesystem Capabilities

Both CreateFileSystem and ModifyFileSystem need a LocallyAccessibleFileSystemSetting element for each file server ComputerSystem. The client first obtains a LocallyAccessibleFileSystemCapabilities element by following ElementCapabilities association from the FileSystemConfigurationService to a LocallyAccessibleFileSystemCapabilities that is associated via ScopedCapabilities (HostedDependency) to the File Server ComputerSystem.

Note: It is expected that there will only be one LocallyAccessibleFileSystemCapabilities element per file server ComputerSystem. All the variability can be found by following SettingsDefineCapabilities to LocallyAccessibleFileSystemSetting elements. It is a requirement that the LocallyAccessibleFileSystemSettings that define the LocallyAccessibleFileSystemCapabilities be associated via HostedDependency (ScopedSetting) to the same file server ComputerSystem as the one indicated by the HostedDependency (ScopedCapabilities).

The client calls LocallyAccessibleFileSystemCapabilities.CreateGoalSettings() with the appropriate parameters.to obtain an appropriate LocallyAccessibleFileSystemSetting element -- CreateGoalSettings can be used to negotiate if necessary.

9.1.3.1.2 Dependency on support for Directory Services

A filesystem may support security principal identifiers associated with filesystem objects for access (typically, read/write/execute) as well as for tracking usage (as would be needed for supporting user and/or group quotas). If the filesystem supports such identifiers, it would requires support from a directory service for validating these identifiers (relating them to accounts and other user-related information). Operationally, computer systems (and not filesystems) are associated to directory services or configured for directory services. Directory service configurations of computer systems are much more complex than needed or appropriate for filesystems. This makes it easier to make the filesystem depend on a computer system, usually a file servier, for providing access to directory services for resolving security principal identifiers.

A filesystem that requires support from a directory service will have the property.DirectoryServicesUsage of its LocalFileSystem element set to ”Required”. In that case, there shall be a Dependency association between the LocalFileSystem element and a file server ComputerSystem.element (with Dedicated=”16”). The associated file server must be configured for access to directory services that it provides for the filesystem.

Note: If DirectoryServicesUsage is “Optional”, a client can look for the Dependency association to determine if the filesystem supports security principal identifiers. This is not supported in this release of the profile.

9.1.3.1.3 Summary of this profile

This profile consists of the following classes, associations, and methods:

3) One LocallyAccessibleFileSystemCapabilities scoped to the file server ComputerSystem

4) ElementCapabilities association to the FileSystemConfigurationService

5) SettingsDefineCapabilities association to LocallyAccessibleFileSystemSetting

6) LocallyAccessibleFileSystemSetting for defining LocallyAccessibleFileSystemCapabilities

7) HostedDependency (ScopedSetting) association from the File Server ComputerSystem to LocallyAccessibleFileSystemSetting

8) A HostedDependency association from the same file server ComputerSystem to the defined LocallyAccessibleFileSystemCapabilities

9) LocallyAccessibleFileSystemCapabilities.CreateGoalSettings extrinsic method to create LocallyAccessibleFileSystemSetting elements scoped to the file server ComputerSystem to use as Goals. Note that this method is different from the method described as part of the FileSystemCapabilities element.

10) A Dependency association from the LocalFileSystem element to a file server ComputerSystem.

9.1.3.2 Finding FileSystem Configuration Services, Capabilities and Pre-defined Settings

When creating a filesystem the first step is to determine what can be created. Figure 14 illustrates an instance diagram showing the instances that shall exist for supporting filesystem creation.

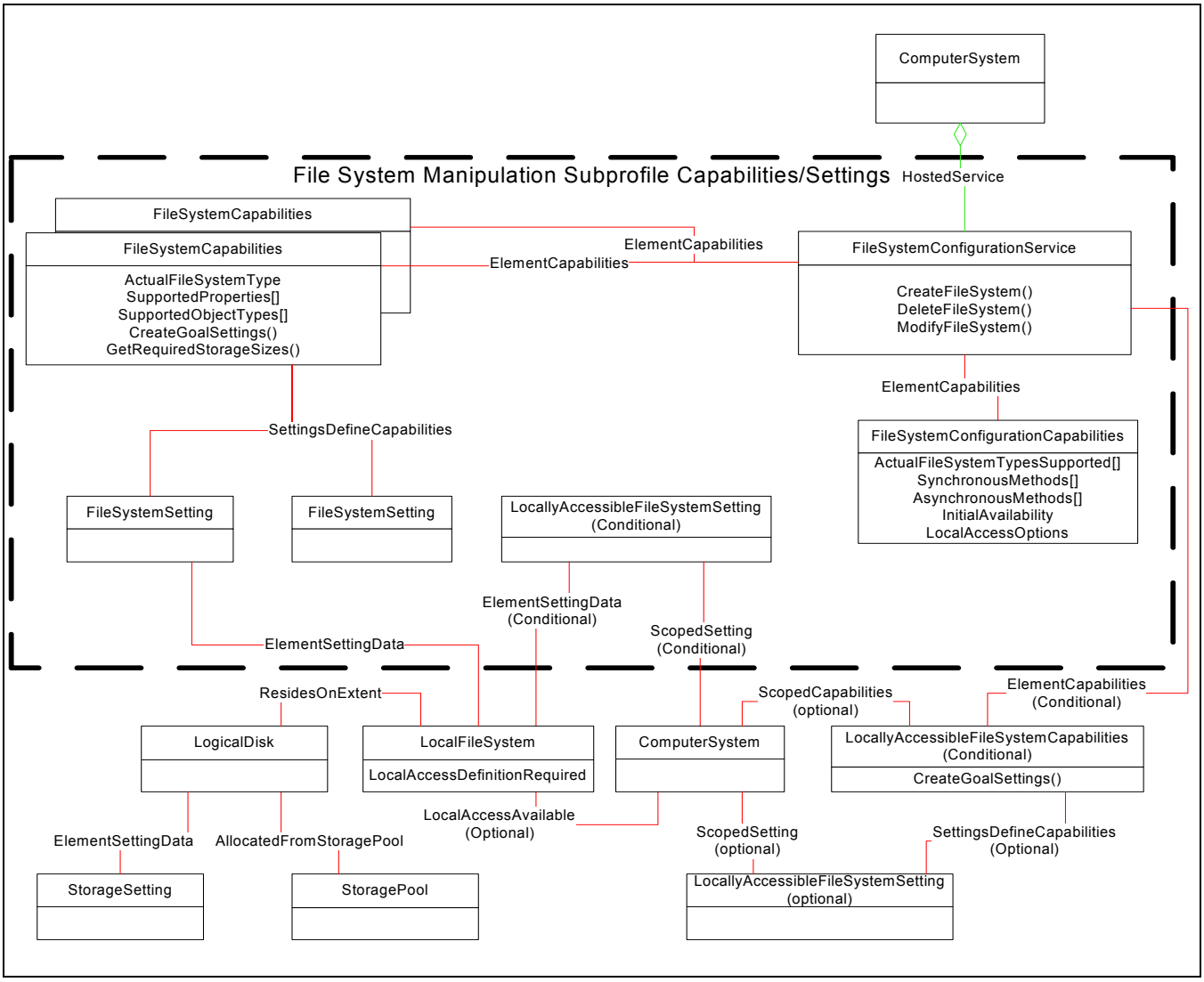

Figure 14 - Capabilities and Settings for Filesystem Creation

At least one FileSystemConfigurationService shall exist if the Filesystem profile has implemented the Filesystem Manipulation Subprofile. The instance(s) of this service can be found by following the HostedService association filtering on the target class of FileSystemConfigurationService.

Note: If no service is found from the Top Level ComputerSystem, the client should look for component computer systems that may be hosting the service. This is not recommended, but permitted for backward compatibility with SMI-S 1.1.

An instance of the FileSystemConfigurationCapabilities shall be associated to the FileSystemConfigurationService via the ElementCapabilities association. A client should follow this association (filtering on the result value of "FileSystemConfigurationCapabilities") to inspect the configuration capabilities that are supported. The client would choose between the filesystem types specified in the array property SupportedActualFileSystemTypes.

For each entry in the SupportedActualFileSystemTypes array, there shall be one instance of FileSystemCapabilities with the same value for the ActualFileSystemType property that shall be associated to the FileSystemConfigurationService using the ElementCapabilities association (filtering on the result value of FileSystemCapabilities). This FileSystemCapabilities element shall specify the supported capabilities for that ActualFileSystemType using a collection of FileSystemSettings. These FileSystemSettings shall be associated to the FileSystemCapabilities via SettingsDefineCapabilities.

An implementation may support a set of pre-defined FileSystemSettings that clients could use directly if desired. The SettingsDefineCapabilities association from the FileSystemCapabilities to the pre-defined FileSystemSettings shall have the PropertyPolicy property be "Correlated", the ValueRole property be "Supported" and the ValueRange property be "Point". Other pre-defined combinations of property values may be specified by FileSystemSettings whose SettingsDefineCapabilities association has the PropertyPolicy be "Independent", ValueRole property be "Supported" and the ValueRange array property contain "Minimums", "Maximums", or "Increment" (see 9.5.1.1.1 for further details on the interpretation of the ValueRange property). These settings can be used by the client to compose FileSystemSettings that are more likely to be directly usable.

9.2 Health and Fault Management Considerations

The key element of this profile is the FileSystemConfigurationService and the hosting ComputerSystem. The operational status of the hosting ComputerSystem should possibly be part of the referring autonomous profile (NAS Head or SC NAS), the Filesystem sub-profile or in the Multiple Computer System sub-profile.

9.2.1 OperationalStatus for FileSystemConfigurationService

9.2.2 OperationalStatus for LocalFileSystem

Table 98 - LocalFileSystem OperationalStatus

Primary OperationalStatus	Secondary OperationalStatus	Description
2 “OK”		The filesystem has good status
2 “OK”	4 “Stressed”	The filesystem resources are stressed
2 “OK”	5 “Predictive Failure”	The filesystem might fail because some resource or component is predicted to fail
2 “OK”	16 “Supporting Entity in Error”	The filesystem may be OK, but is not accessible because a supporting entity is not accessible.
3 “Degraded”		The filesystem is operating in a degraded mode. This could be due to the health state of the underlying storage being degraded or in error.

Table 98 - LocalFileSystem OperationalStatus

Primary OperationalStatus	Secondary OperationalStatus	Description
6 “Error”		An error has occurred causing the filesystem to become unavailable. Operator intervention through SMI-S (managing the LocalFileSystem) to restore the filesystem may be possible.
6 “Error”		An error has occurred causing the filesystem to become unavailable. Automated recovery may be in progress.
6 “Error”	7 “Non-recoverable Error”	The filesystem is not functioning. Operator intervention through SMI-S will not fix the problem.
6 “Error”	16 “Supporting Entity in Error”	The filesystem is in an error state because a supporting entity is not accessible.
8 “Starting”		The filesystem is in process of initialization and is not yet available operationally.
9 “Stopping”		The filesystem is in process of stopping, and is not available operationally.
10 “Stopped”		The filesystem cannot be accessed operationally because it is stopped -- if this did not happened because of operator intervention or happened in real-time, the OperationalStatus would have been “Lost Communication” rather than “Stopped”.
11 “In Service”		The filesystem is offline in maintenance mode, and is not available operationally.
13 “Lost Communications”		The filesystem cannot be accessed operationally -- if this happened because of operator intervention it would have been “Stopped” rather than “Lost Communication”.

Table 98 - LocalFileSystem OperationalStatus

Primary OperationalStatus	Secondary OperationalStatus	Description
14 “Aborted”		The filesystem is stopped but in a manner that may have left it in an inconsistent state.
15 “Dormant”		The Filesystem is offline; and the reason for not being accessible is unknown.

9.3 Cascading Considerations

Under Consideration for a future standard.

9.4 Supported Subprofiles and Packages

Table 99 describes the supported profiles for Filesystem Manipulation.

Table 99 - Supported Profiles for Filesystem Manipulation

Registered Profile Names	Mandatory	Version
Job Control	No	1.3.0
Filesystem	Yes	1.3.0
Indication	Yes	1.3.0
Volume Composition	No	1.3.0

9.5 Methods of the Profile

9.5.1 Extrinsic Methods of the Profile

Table 100 - Filesystem Manipulation Methods that cause Instance Creation, Deletion or Modification

Method	Created Instances	Deleted Instances	Modified Instances
FileSystemConfigurationService .CreateFileSystem	LocalFileSystem LogicalFile FileSystemSetting ElementSettingData FileStorage ResidesOnExtent HostedFileSystem LogicalDisk(s) StorageSetting(s) LocalAccessAvailable(s) LocallyAccessibleFileSystem Setting(s) ElementSettingData(s) HostedDependency LogicalFile (BC 1.1) FileStorage (BC 1.1) Dependency	N/A	StoragePool(s) LogicalDisk(s)
FileSystemConfigurationService .DeleteFileSystem		LocalFileSystem LogicalFile FileSystemSetting ElementSettingData FileStorage ResidesOnExtent HostedFileSystem LocalAccessAvailable(s) LocallyAccessibleFileSystem Setting(s) ElementSettingData(s) HostedDependency Dependency	N/A
FileSystemConfigurationService .ModifyFileSystem	(IF REQUESTED) LogicalDisk(s) StorageSetting(s) LocalAccessAvailable LocallyAccessibleFileSystem Setting ElementSettingData(s) HostedDependency	(if Local Access is modified) LocalAccessAvailable LocallyAccessibleFileSystem Setting ElementSettingData(s) HostedDependency	FileSystemSetting (if changed) ResidesOnExtent (if added)
FileSystemCapabilities.CreateGoalSettings	N/A	N/A	N/A
LocallyAccessibleFileSysstemCapabilities.CreateGoalSettings	N/A	N/A	N/A
FileSystemCapabilities.GetRequiredStorageSize	N/A	N/A	N/A

9.5.1.1 FileSystemCapabilities.CreateGoalSettings

This extrinsic method of the FileSystemCapabilities class validates support for a caller-proposed FileSystemSetting passed as the TemplateGoalSettings parameter. This profile restricts the usage of this method to a single entry array for both TemplateGoalSettings and SupportedGoalSettings parameters.

If the input TemplateGoalSettings is NULL or the empty string, this method returns a single default FileSystemSetting in the SupportedGoalSettings array. Both TemplateGoalSettings and SupportedGoalSettings are string arrays containing embedded instances of type FileSystemSetting. As such, these settings do not exist in the implementation but are the responsibility of the client.

If the TemplateGoalSettings specifies values that cannot be supported, this method shall return an appropriate error and should return a best match for a SupportedGoalSettings.

The client and the implementation can engage in a negotiation process that may require iterative calls to this method. To assist the implementation in tracking the progress of the negotiation, the client may pass previously returned values of SupportedGoalSettings as the new input value of SupportedGoalSettings. The implementation may determine that a step has not resulted in progress if the input and output values of SupportedGoalSettings are the same. A client may infer from the same result that the TemplateGoalSettings must be modified.

9.5.1.1.1 Client Considerations

It is important to understand that the client is acting as an agent for a human user -- either a "system" administrator, or other entity with administrative privileges with respect to the filesystem or the filesystem host. During negotiation, the client will show the current state to the user -- the SupportedGoalSettings received to date (either the latest or some subset), the TemplateGoalSettings proposed (the most recent, but possibly more). But the administrator needs a representation of what is available, possibly the range or sets of values that the different setting properties can take. Some decisions are assumed to have been made already, such as the type of Filesystem to be created and the number of LogicalDisks to use and their StorageSettings. It is possible that the LogicalDisks for use by this Filesystem have already been designated by the user; if not, the StoragePool(s) from which they will be created is already designated or will be selected by an independent process.

The SettingsDefineCapabilities association from the selected Capabilities element provides the information for the client to lay out these options. "Point" settings can be identified using FileSystemSettings -- these points can be further qualified to indicate whether these are supported (or not), and even whether they represent some ideal point in the space -- a "minimum", or a "maximum", or an "optimal" point. Other settings can provide ranges for properties -- by specifying a minimum, a maximum, and an increment an arithmetic progression of values can be specified (a continuous range can be specified with a zero increment). Specifying a set of supported values for a property that do not follow some pattern is possible, if a bit tedious.

The set of settings associated via SettingsDefineCapabilities are expected to be quite stable -- real systems do not continually vary the functionality they can support. Such variations do occur -- for instance, if a new PCMCIA card is added to a running system -- and the best way for a client to be able to add these to the set of choices presented to a user is to subscribe to indications on new Capabilities elements and new instances of SettingsDefineCapabilities.

There is no guarantee that a negotiation will terminate successfully with the client and the implementation achieving agreement. The implementation may support some simpler mechanisms, short of fully-fledged negotiation, that would be used by a client to obtain an acceptable TemplateGoalSettings. The following two use cases are easily covered:

1) Client considers only the canned settings that are specified exactly. The canned settings are specified by the FileSystemSettings that are associated to the FileSystemCapabilities via SettingDefinesCapabilities association with the following property values:

   - SettingDefinesCapabilities.PropertyPolicy = "Correlated"
   - SettingDefinesCapabilities.ValueRole = "Supported"
   - SettingDefinesCapabilities.ValueRange = "Point"

2) Client considers canned settings that range over values specified using minimum/increment/maximum for the Setting properties. For example, these could be specified by the FileSystemSettings that are associated to the FileSystemCapabilities via SettingDefinesCapabilities association with the following property values:

- SettingDefinesCapabilities.ValueRange = "Minimums" or "Maximums" or "Increments"
- The PropertyPolicy and ValueRole properties of SettingDefinesCapabilities will be appropriately specified

9.5.1.1.2 Signature and Parameters of FileSystemCapabilities.CreateGoalSettings

Table 101 - Parameters for Extrinsic Method FileSystemCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	EmbeddedInstance ("SNIA_FileSystemSetting") TemplateGoalSettings is a string array containing embedded instances of class FileSystemSetting, or a derived class. This parameter specifies the client's requirements and is used to locate matching settings that the implementation can support.
SupportedGoalSettings[]	INOUT	string	EmbeddedInstance("SNIA_FileSystemSetting") SupportedGoalSettings is a string array containing embedded instances of class FileSystemSetting, or a derived class. On input, it specifies a previously returned set of Settings that the implementation could support. On output, it specifies a new set of Settings that the implementation can support. If the output set is identical to the input set, both client and implementation may conclude that this is the best match for the TemplateGoalSettings that is available. If the output does not match the input and the non-NULL output does not match the non-NULL TemplateGoalSettings, then the method must return "Alternative Proposed". If the output is NULL, the method must return an "Failed".
Normal Return			
Status		uint32	ValueMap{}, Values{} "Success", "Failed", "Timeout", "Alternative Proposed"

Table 101 - Parameters for Extrinsic Method FileSystemCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.1.2 GetRequiredStorageSize

This extrinsic method returns the minimum, expected, and maximum size for a LogicalDisk that would support a Filesystem specified by the input FileSystemGoal parameter. The caller may provide relevant settings of the LogicalDisk via the ExtentSetting parameter. The minimum, maximum, and expected sizes are returned as output parameters.

If the input FileSystemGoal is NULL, the implementation may return an error or use a default FileSystemSetting associated with this FileSystemCapabilities element. The actual FileSystemSetting used is returned as an OUT parameter.

If the input ExtentSetting parameter is NULL or is an empty array, the implementation may use a default StorageSetting associated with the StorageConfigurationService hosted on the same ComputerSystem as the FileSystemConfigurationService associated with this FileSystemCapabilities element. The actual StorageSetting used is returned as an OUT parameter.

Note: The actual FileSystemSetting and StorageSetting used are being returned as OUT parameters. This is a non-backward-compatible change from SMI-S 1.1.

9.5.1.2.1 Signature and Parameters of GetRequiredStorageSize

Table 102 - Parameters for Extrinsic Method FileSystemCapabilities.GetRequiredStorageSize

Parameter Name	Qualifier	Type	Description & Notes
FileSystemGoal	INOUT, EI	string	EmbeddedInstance ("SNIA_FileSystemSetting") FileSystemGoal is an Embedded Instance element of class CIM_FileSystemSetting, or a derived class, that specifies the settings for the FileSystem to be created. If NULL on input, a default for this FileSystemCapabilities is used. On output, this returns the actual FileSystemSetting that was used.
ExtentSetting	INOUT, EI	string	EmbeddedInstance("CIM_StorageSetting") ExtentSetting is an Embedded Instance element of class CIM_StorageSetting, or a derived class, that specifies the settings for the LogicalDisk to be used for building this FileSystem. If NULL on input, a default StorageSetting will be obtained from a StorageConfigurationService hosted on the same ComputerSystem as this FileSystemConfigurationService. On output, this returns the actual StorageSetting that was used. If the output is NULL, the method must return an “Failed”.
ExpectedSize	OUT	uint64	An integer that indicates the size of the storage extent that this FileSystem is expected to need. An entry value of 0 indicates that there is no expected size.
MinimumSizeAcceptable	OUT	uint64	An integer that indicates the size of the smallest storage extent that would support the specified FileSystem. A value of 0 indicates that there is no minimum size.
MaximumSizeUsable	OUT	uint64	An integer that indicates the size of the largest storage extent that would be usable for the specified FileSystem. A value of 0 indicates that there is no maximum size.
Normal Return			
Status		uint32	ValueMap{}, Values{} "Success", "Failed", "Timeout"

Table 102 - Parameters for Extrinsic Method FileSystemCapabilities.GetRequiredStorageSize

Parameter Name	Qualifier	Type	Description & Notes
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.1.3 LocallyAccessibleFileSystemCapabilities.CreateGoalSettings

This extrinsic method of the LocallyAccessibleFileSystemCapabilities class validates support for a caller-proposed LocallyAccessibleFileSystemSetting passed as the TemplateGoalSettings parameter. This profile restricts the usage of this method to a single entry array for both TemplateGoalSettings and SupportedGoalSettings parameters.

If the input TemplateGoalSettings is NULL or the empty string, this method returns a single default LocallyAccessibleFileSystemSetting in the SupportedGoalSettings array. Both TemplateGoalSettings and SupportedGoalSettings are string arrays containing embedded instances of type LocallyAccessibleFileSystemSetting. As such, these settings do not exist in the implementation but are the responsibility of the client.

If the TemplateGoalSettings specifies values that cannot be supported, this method shall return an appropriate error and should return a best match for a SupportedGoalSettings.

The client and the implementation engage in a negotiation process that may require iterative calls to this method. To assist the implementation in tracking the progress of the negotiation, the client may pass previously returned values of SupportedGoalSettings as the new input value of SupportedGoalSettings. The implementation may determine that a step has not resulted in progress if the input and output values of SupportedGoalSettings are the same. A client may infer from the same result that the TemplateGoalSettings must be modified.

9.5.1.3.1 Client Considerations

It is important to understand that the client is acting as an agent for a human user -- either a "system" administrator, or other entity with administrative privileges to the Filesystem. During negotiation, the client will show the current state to the user -- the SupportedGoalSettings received to date (either the latest or some subset), the TemplateGoalSettings proposed (the most recent, but possibly more). But the administrator needs a representation of what is available, possibly the range or sets of values that the different setting properties can take. Some decisions are assumed to have been made already, such as whether the local access is read-only or the file server that is going to access the Filesystem.

The SettingsDefineCapabilities association from the selected Capabilities element provides the information for the client to lay out these options. "Point" settings can be identified supported points in the space of properties -- these points can be further qualified to indicate whether these are supported or not, or whether they represent some ideal point in the space -- a "minimum", or a "maximum", or an "optimal" point. Other settings can provide ranges for properties -- by specifying a minimum, a maximum, and an increment an arithmetic progression of values can be specified (a continuous range can be specified with a zero increment). Specifying a set of supported values for a property that do not follow some pattern is possible, if a bit tedious.

The set of settings associated via SettingsDefineCapabilities are expected to be quite stable -- real systems do not continually vary the functionality they can support. Such variations do occur -- for instance, if a new PCMCIA card

is added to a running system -- and the best way for a client to be able to add these to the set of choices presented to a user is to subscribe to indications on new Capabilities elements and new instances of SettingsDefineCapabilities.

There is no guarantee that a negotiation will terminate successfully with the client and the implementation achieving agreement. The implementation may support some simpler mechanisms, short of fully-fledged negotiation, that would be used by a client to obtain an acceptable TemplateGoalSettings. The following two use cases are easily covered:

1) Client only considers only the canned settings specified exactly. The canned settings are specified by the LocallyAccessibleFileSystemSetting elements that are associated to the LocallyAccessibleFileSystemCapabilities via SettingDefinesCapabilities association with the following property values:

   - SettingDefinesCapabilities.PropertyPolicy = "Correlated"
   - SettingDefinesCapabilities.ValueRole = "Supported"
   - SettingDefinesCapabilities.ValueRange = "Point"

2) Client considers canned settings that range over values specified using minimum/increment/maximum for the Setting properties. For example, these could be specified by the LocallyAccessibleFileSystemSetting elements that are associated to the LocallyAccessibleFileSystemCapabilities via SettingDefinesCapabilities association with the following property values:

   - SettingDefinesCapabilities.ValueRange = "Minimums" or "Maximums" or "Increments"
   - The PropertyPolicy and ValueRole properties of SettingDefinesCapabilities will be appropriately specified

Comparing the two CreateGoalSettings extrinsic methods of this profile, the FileSystemCapabilities.CreateGoalSettings() can be significantly more complex than LocallyAccessibleFileSystemCapabilities.CreateGoalSettings(). The client can choose to implement a simpler negotiation protocol for one -- this specification does not mandate the extent to which the client must use this protocol.

9.5.1.3.2 Signature and Parameters of CreateGoalSettings

Table 103 - Parameters for Extrinsic Method LocallyAccessibleFileSystemCapabilities.CreateGoalSettings

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	EmbeddedInstance ("SNIA_LocallyAccessibleFileSystemSetting") TemplateGoalSettings is a string array containing embedded instances of class LocallyAccessibleFileSystemSetting, or a derived class. This parameter specifies the client's requirements that is used to locate matching settings that the implementation can support.

Table 103 - Parameters for Extrinsic Method LocallyAccessibleFileSystemCapabilities.Create-GoalSettings (Continued)

Parameter Name	Qualifier	Type	Description & Notes
SupportedGoalSettings[]	INOUT	string	EmbeddedInstance("SNIA_LocallyAccessibleFileSystemSetting") SupportedGoalSettings is a string array containing embedded instances of class LocallyAccessibleFileSystemSetting, or a derived class. On input, it specifies a previously returned set of Settings that the implementation could support. On output, it specifies a new set of Settings that the implementation can support. If the output set is identical to the input set, both client and implementation may conclude that this is the best match for the TemplateGoalSettings that is available. If the output does not match the input and the non-NULL output does not match the non-NULL TemplateGoalSettings, then the method must return \"Alternative Proposed\". If the output is NULL, the method must return an “Failed”.
Normal Return			
Status		uint32	ValueMap{}, Values{} "Success", "Failed", "Timeout", "Alternative Proposed"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.1.4 FileSystemConfigurationService.CreateFileSystem

This extrinsic method creates a LocalFileSystem and returns it as the value of the parameter TheElement. The desired settings for the LocalFileSystem are specified by the Goal parameter (a string-valued EmbeddedInstance object of class FileSystemSetting).

Filesystem vendors differ in their models for creating a filesystem. Some vendors require that the storage element already exist; others create the storage element at the same time as the filesystem. Some vendors require a local access point ("mount-point") that supports defining a name or pathname that allows a file server to access the filesystem; others do not require any such object (though it could be argued that they provide a default local access

mechanism). This extrinsic method supports variant mechanisms for specifying, at create time, storage element creation as well as local access by a file server. The FileSystemConfigurationCapabilities associated with the FileSystemConfigurationServices contains the property BlockStorageCreationSupport that specifies support for create-time storage element creation; the property LocalAccessibilitySupport that specifies support for local access by a file server at creation; the property DirectoryServerParameterSupported that specifies support for specifying a file server that provides access to a Directory Service (if enabled separarely).

To support backward compatibility with the SMI-S 1.1 Filesystem subprofile, an instance of Directory (a derived class of LogicalFile) is created representing the root directory of the newly created filesystem. This Directory element is associated to the LocalFileSystem via FileStorage.

A FileSystemSetting element that represents the settings of the LocalFileSystem (either identical to the Goal or equivalent) shall be associated via ElementSettingData to the LocalFileSystem. The implementation shall create a new FileSystemSetting for this purpose.

The output parameter TheElement shall contain a reference to the newly created LocalFileSystem. Even if this operation does not complete but creates a job, an implementation may return a valid reference in TheElement. If the job fails subsequently, it is possible for this reference to become invalid.

9.5.1.4.1 Specifying Storage Underlying the Filesystem

BlockStorageCreationSupport is an array of enumerated values that specifies if:

- An enumerated value that specifies whether the extrinsic methods may use an already existing LogicalDisk -- this is either required, optional, or not allowed. If "Not Allowed", the Pools and ExtentSettings parameters must be used to create LogicalDisk(s) for this LocalFileSystem and the InExtents parameter must be NULL. If "Optional", either the Pools and ExtentSettings parameters or the InExtents parameter should be specified, but not both. If "Required", on the InExtents parameter may be specified and the Pools and ExtentSettings parameters must be NULL.
- (optional) An integer that specifies an upper limit to the number of storage elements that can be specified, either as InExtents parameters or as Pools and ExtentSettings.
- (optional) An integer that specifies the number of distinct storage pools that the Pools parameters can specify -- zero, if Pools is not supported or if there is no limit, and a specific number if there is a limit. In practice it is expected that the value will be either zero or one.
- (optional) A truth value represented as ’0’ for false and ’1’ for true that indicates whether an entry in the ExtentSettings array parameter can be NULL (indicating that a default setting is to be used).

The storage used for creating the LocalFileSystem is specified by the InExtents parameter that must be an array of LogicalDisks. If InExtents is NULL and BlockStorageCreationSupport indicates that Pools are optional or required, the parameter Pools must specify an array of StoragePools from which storage may be allocated -- the requirements for the LogicalDisks allocated from this Pool is specified in the ExtentSettings array parameter. The Pools may use an associated StorageConfigurationService. The LocalFileSystem is associated to one of the LogicalDisk(s) via the ResidesOnExtent association. The other LogicalDisks extend the distinguished LogicalDisk (as modeled by the Volume Composition Sub-Profile).

9.5.1.4.2 Specifying Local Access to the Filesystem

LocalAccessibilitySupport is an enumeration that specifies whether the implementation requires a local access specification, or makes it optional (thus using a vendor default), or does not require one ("local access" does not have a meaning for the vendor).

The LocalFileSystem shall be hosted on the same ComputerSystem as the FileSystemConfigurationService.

Note: The requirement that the LocalFileSystem have the same host as the Service is too restrictive but this method can be extended in the future with a FileSystemHost parameter (implicitly NULL in 1.2).

If LocalAccess is supported, whether optional or required, this method supports specifying one file server ComputerSystem (the reference parameter FileServer) that is given local access to this Filesystem. If LocalAccess is optional, the FileServer parameter may be NULL. The local access name on the FileServer is specified in the LocalAccessPoint string parameter -- if the implementation uses pathnames, this will be formatted as a pathname (directory names separated by the PathNameSeparatorString). The implementation could also use a differently formatted local access name (for instance, a simple name). The settings to be used for this are specified in the LocalAccessSetting, an EmbeddedInstance element of class LocallyAccessibleFileSystemSetting.

Note: If a second file server ComputerSystem is to be given local access, the ModifyFileSystem method would be used.

Corresponding to the LocalAccessPoint parameter, a LocalAccessAvailable association instance and a LocallyAccessibleFileSystemSettings element are created with the following properties and associations:

- The LocalAccessAvailable association goes between the FileServer parameter and the created LocalFileSystem (TheElement parameter).
- The LocalAccessAvailable.LocalAccessPoints property is set to the value of the LocalAccessPoint string.
- The LocallyAccessibleFileSystemSetting element has an ElementSettingData association to the LocalFileSystem (TheElement).
- The LocallyAccessibleFileSystemSetting element has a HostedDependency (ScopedSetting) association to the FileServer parameter.

If LocalAccess is supported, the FileServer parameter should not be NULL.

Note: If it is NULL, the implementation may leave the filesystem operationally inaccessible -- however, this can be corrected by calling the ModifyFileSystem method. This is not an Error.

If LocalAccess is supported and the FileServer parameter is not NULL, the LocalAccessPoint string may be NULL or the empty string. In this case, the LocalAccessSetting parameter should indicate the implementation-specific default format. The default value that is used is returned as the OUT value of the LocalAccessPoint parameter. It is an Error if the LocalAccessSetting parameter does not provide an appropriate default mechanism for constructing a local access name.

The LocalAccessSetting parameter will return an EmbeddedInstance of the LocallyAccessibleFileSystemSetting actually used on output.

9.5.1.4.3 Specifying access to Directory Services

DirectoryServerParameterSupported is a property that specifies whether the filesystem must have access to a file server that provides access to directory services so that security principal information may be supported. If the newly created filesystem must be able to resolve such information, the DirectoryServer parameter must be specified to the CreateFileSystem method.

The DirectoryServer parameter specifies a file server ComputerSystem that is configured to access a directory service that resolves security principal identifies (UIDs, GIDs, or SIDs) used by the filesystem. This profile does not specify the configuration of any directory service (if there is one), any directory server, or the file server that is specified by the DirectoryServer parameter. For operational efficiency reasons, this must be a file server since security principal information such as usage and detection of threshold violations must happen in real-time.

If the DirectoryServer is required and is specified, an association, a concrete subclass of Dependency, shall be surfaced between the newly created LocalFileSystem element (as Dependent) and the specified file server (as Antecedent). The CreateFileSystem method will return a reference to this file server as the return value of the parameter. In this case, the LocalFileSystem.DirectoryServiceUsage property shall be set to “Supported”.

Any file server that is configured with write access to a filesystem must use the same or a compatible directory service (effectively the same) as the file server indicated by the Dependency association.

9.5.2 Signature and Parameters of CreateFileSystem.

Table 104 - Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem

Parameter Name	Qualifier	Type	Description & Notes
ElementName	IN	string	An end user relevant name for the FileSystem being created. The value shall be stored in the 'ElementName' property for the created element. This parameter shall not be NULL or the empty string.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Goal	IN, OUT, EI	string	EmbeddedInstance ("CIM_FileSystemSetting") The FileSystemSetting requirements for the FileSystem. If NULL or the empty string, a default FileSystemSetting shall be specified by the FileSystemCapabiltiies element associated to the FileSystemConfigurationService by the DefaultElementCapabilities association. The actual FileSystemSetting used is returned on output.
TheElement	OUT, REF	CIM_LocalFileSystem	The newly created FileSystem.
InExtents[]	IN, OUT, REF, NULL allowed,	CIM_LogicalDisk	The LogicalDisk(s) on which the created FileSystem shall reside. If this is NULL, the Pools and ExtentSettings parameters cannot be NULL and are used to create LogicalDisk(s). The LogicalDisk(s) actually used will be returned on output.
Pools[]	IN, REF, NULL allowed	CIM_StoragePool	An array of concrete StoragePool elements corresponding to the ExtentSettings parameter from which to create LogicalDisks in case the InExtents parameter is NULL. If InExtents is not NULL, this must be NULL.
ExtentSettings[]	IN, EI, NULL Allowed	string	EmbeddedInstance ("CIM_StorageSetting") An array of embedded StorageSetting structures that specify the settings to use for creating LogicalDisks if the InExtents parameter is NULL and Pools is specified. Each LogicalDisk will be created from the corresponding entry in Pools, so each StorageSetting entry must be supported by the capabilities of the corresponding Pools entry.
Sizes[]	IN, OUT, NULL Allowed	uint64	An array of numbers that specifies the size in bytes of the LogicalDisks to be created corresponding to the Pools and ExtentSettings parameters. The sum of Sizes should be at least as much as (or greater than) the FileSystem size needed.

Table 104 - Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN, OUT, REF, NULL Allowed	ComputerSystem	A reference to a ComputerSystem element that will access the created LocalFileSystem and is capable of exporting the filesystem as a file share. The local access point with respect to the file server is specified by the LocalAccessPoint parameter. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points are supported but implementation-defaulted, the corresponding entry in the LocalAccessPoint parameter should be NULL or the empty string as the LocalAccessPoint name is constructed as per the vendor default algorithm. A LocalAccessAvailable association is created between the FileServer and the LocalFIleSystem. The parameters for local access are specified by the LocalAccessSetting parameter. Since this Filesystem has just been created, the LocalAccessSetting can support Write privileges. If the LocalAccessSetting entry is NULL or the empty string, the implementation uses a default associated with the LocallyAccessibleFileSystemCapabilities associated to the FileServer. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that a local access point is required and FileServer is NULL, no LocalAccessAvailable associations are created and the Filesystem may not be accessible. This shall not cause an Error. On output, this parameter contains a reference to the actual FileServer that has access to the created LocalFileSystem.

Table 104 - Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem

Parameter Name	Qualifier	Type	Description & Notes
LocalAccessPoint	IN, OUT, REF, NULL Allowed	string	A string to use as a pathname in the name space of the file server ComputerSystem. The format of the string is vendor-dependent and it should be considered opaque from the client's standpoint. It could be interpreted as a hierarchical fully-qualified name for the local access point (say in a Unix-based operating environment), or it could be a drive letter (as in a Windows operating environment). A LocalAccessAvailable association is created going between the new LocalFileSystem and the FileServer parameter. The LocalAccessAvailable.LocalAccessPoint property will be set to this parameter. The parameters for local access are specified by the LocalAccessSetting parameter. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points are required, then LocalAccessPoint shall not be NULL or an empty string. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points can be vendor-defaulted, then LocalAccessPoint can be NULL or an empty string and the implementation shall create a name using a vendor-specific algorithm. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points cannot be vendor-defaulted, then LocalAccessPoint shall not be NULL and the implementation shall not create a default pathname. This is an Error. On output, this parameter contains the actual LocalAccessPoint used (including any name created by vendor-default).

Table 104 - Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem

Parameter Name	Qualifier	Type	Description & Notes
LocalAccessSetting	IN, EI, OUT, NULL Allowed	string	EmbeddedInstance ("CIM_LocallyAccessibleFileSystemSetting") An embedded LocallyAccessibleFileSystemSetting element that specifies the settings to use to establish a local access point. This element will be used to create a LocalAccessAvailable association and will be cloned to create a LocallyAccessibleFileSystemSetting element that is scoped via HostedDependency (ScopedSetting) to the FileServer and associated via ElementSettingData to the LocalFileSystem. If a LocallyAccessibleFileSystemSetting entry is NULL or the empty string, the implementation shall use the default provided by the LocallyAccessibleFileSystemCapabilities element of the FileSystemConfigurationService that is associated to the FileServer via CIM_Dependency. The LocalAccessSetting may specify a Write Privilege. The LocalAccessSetting actually used is returned as the OUT EmbeddedInstance parameter.

Table 104 - Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem

Parameter Name	Qualifier	Type	Description & Notes
DirectoryServer	IN, OUT, NULL Allowed	ComputerSystem	A reference to a ComputerSystem element that has access to directory services. The newly created filesystem can use it to support security principal information associated with filesystem objects, such as quotas for users and groups. This is represented by providing a Dependency association between the LocalFileSystem element and the ComputerSystem indicated by this parameter. The requirements for this parameter are further specified by FileSystemConfigurationCapabilities.DirectoryServerParameterSupported. If DirectoryServerParameterSupported specifies 'Not Used', or 'Supported, Defaulted to FileServer', or 'Supported, Defaulted to FileSystem host', it is an Error if DirectoryServer is not NULL. Otherwise, (i.e., if DirectoryServerParameterSupported specifies 'Supported'), and if the DirectoryServer is not NULL, the new filesystem will use the directory services made available by the specified DirectoryServer. If DirectoryServer is NULL, it will be defaulted to the FileServer parameter. If the FileServer parameter is also NULL, the DirectoryServer will be defaulted to the host of the newly created filesystem. On output, this parameter contains a reference to the actual DirectoryServer if one was established.
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.2.1 FileSystemConfigurationService.ModifyFileSystem

This extrinsic method modifies a LocalFileSystem specified by the parameter TheElement. The desired settings for the LocalFileSystem are specified by the Goal parameter (a string-valued EmbeddedInstance object of class FileSystemSetting).

As with CreateFileSystem, the FileSystemConfigurationCapabilities associated with the FileSystemConfigurationService specifies support for some of the optional behaviors of this method. BlockStorageCreationSupport indicates whether this method only uses previously existing storage elements or if it can create them at the same time as modifying or creating the filesystem. In addition this can specify if additional LogicalDisks can be added to the existing set of LogicalDisks and whether the implementation limits the number of LogicalDisks underlying a filesystem. LocalAccessibilitySupport indicates whether the implementation requires support for local access points (or if they are optional or not required at all).

This element shall have a FileSystemSetting use ActualFileSystemType property is supported by the FileSystemConfigurationService (as specified by the SupportedActualFileSystemTypes property of the associated FileSystemConfigurationCapabilities). The existing LogicalDisks used by the LocalFileSystem cannot be released by this method, but this method may add LogicalDisks. These LogicalDisks may be specified by the InExtents parameter (if that is either required or optional) or, if InExtents is NULL (if Pools are optional or required), the set of LogicalDisks is not changed. New LogicalDisks may also be added by specifying an array of StoragePools in the Pools parameter and an array of StorageSettings that can be used to create them.

If the operation would result in a change in the size of a filesystem, the ResidesOnExtent association shall be used to determine how to implement the change. If the existing or additional LogicalDisk(s) specified, or any additional LogicalDisks created, cannot support the goal size, an appropriate error value shall be returned, and no action shall be taken. If the operation succeeds, the ResidesOnExtent association shall reference the same LogicalDisk as before (however, the LogicalDisk will be built upon a larger number of underlying LogicalDisks, as modeled by the Volume Composition subprofile).

If the new Goal is different from the old FileSystemSetting element associated to the LocalFileSystem element, then the implementation must change the setting properties of the LocalFileSystem. This may be accomplished by modifying the old FileSystemSetting element directly, or by deleting it and then re-creating a new FileSystemSetting element with the same InstanceId. Just like the old element, the new FileSystemSetting element shall be associated to the LocalFileSystem element via an ElementSettingData association.

If local access is supported, whether optional or required, any change is specified by providing the FileServer parameter (which shall be a reference to a ComputerSystem). If a FileServer is not being added to the set or modified or removed from the set, the FileServer parameter may be NULL.

If the specified FileServer does not duplicate a ComputerSystem that has already been specified as having local access, this method adds it to the set. The pathname is specified by the LocalAccessPoint string array parameter. The settings to be used for these are specified in the LocalAccessSetting, an EmbeddedInstance element of class LocallyAccessibleFileSystemSetting.

If the specified FileServer duplicates a ComputerSystem that has already been specified as having local access, this method either modifies the local access or removes it from the set. If the LocalAccessPoint parameter is NULL or consists of an empty string, this call removes the FileServer from the set. If the LocalAccessPoint parameter is not NULL but specifies the current path, then this call modifies the settings of the local access -- the new settings are specified by the LocalAccessSetting parameter. If the LocalAccessPoint parameter is not NULL but specifies a path other than the current path, then this call modifies the pathname as well as the settings. If this filesystem is in operational use when such a request is made, the request may have to be suspended until the filesystem can be put into an appropriate state for making the change.

The settings to be used for adding or modifying are specified by the LocalAccessSetting parameter, an EmbeddedInstance element of class LocallyAccessibleFileSystemSetting.

To add a local access point, a LocalAccessAvailable association and a LocallyAccessibleFileSystemSettings element are created with the following properties and associations:

- A LocalAccessAvailable association goes between the FileServer parameter and the LocalFileSystem (TheElement parameter).
- A LocalAccessAvailable.LocalAccessPoint property is set to the LocalAccessPoint string.
- A LocallyAccessibleFileSystemSetting element with a ElementSettingData association to the LocalFileSystem (TheElement parameter).
- The LocallyAccessibleFileSystemSetting element has a HostedDependency (ScopedSetting) association to the FileServer parameter.

Note: The WaitTime and InUseOptions parameters are supported if the FileSystemCapabilities.SupportedProperties includes the "RequireInUseOptions" option.

Note: A client can identify all local access specifications for a filesystem by looking for the LocalAccessAvailable association from the LocalFileSystem to a file server ComputerSystem and the LocallyAccessibleFileSystemSetting associated to the LocalFileSystem via ElementSettingData and the same file server ComputerSystem via HostedDependency (ScopedSetting).

9.5.3 Signature and Parameters of ModifyFileSystem.

Table 105 - Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem

Parameter Name	Qualifier	Type	Description & Notes
ElementName	IN, OUT	string	An end user relevant name for the filesystem being modified. If NULL, the existing TheElement.ElementName property is not changed. If not NULL, this parameter will supply a new name for the Element parameter. The actual ElementName is returned as the output value.
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
Goal	IN, OUT, EI	string	EmbeddedInstance ("CIM_FileSystemSetting") The FileSystemSetting requirements for the filesystem specified by the TheElement parameter. If NULL or the empty string, the settings for TheElement will not be changed. If not NULL, this parameter will supply new settings that replace or are merged with the current settings of TheElement.
TheElement	IN, REF	CIM_LocalFileSystem	The LocalFileSystem element to modify.
InExtents[]	IN, OUT, REF, NULL allowed,	CIM_LogicalDisk	The LogicalDisk(s) used to extend the current set of LogicalDisks used for the TheElement filesystem. If this is not NULL, the Pool and ExtentSettings must be NULL. If both this and Pool are NULL, the current set will not be changed. The current set of LogicalDisk(s) will be returned as the output.
Pools[]	IN, REF, NULL allowed	CIM_StoragePool	An array of concrete storage pools corresponding to the ExtentSettings array parameter. These storage pools are used to create additional LogicalDisks to extend the TheElement filesystem. The InExtents parameter must be NULL and the ExtentSettings parameter must not be NULL. Otherwise, the current set of LogicalDisks is not changed.
ExtentSettings[]	IN, EI, NULL Allowed	string	EmbeddedInstance ("CIM_StorageSetting") An array of embedded StorageSetting structures that specify the settings to use for creating additional LogicalDisks for the TheElement filesystem. The InExtents parameter must be NULL and Pools must be specified. Each LogicalDisk will be created from the corresponding Pool, so each StorageSetting entry must be supported by the capabilities of the corresponding Pool entry.
Sizes[]	IN,NULL Allowed	uint64	An array of numbers that specifies the size in bytes of the LogicalDisks to be created corresponding to the ExtentSettings array parameter.

Table 105 - Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN, OUT, REF, NULL Allowed	REF ComputerSystem	A reference to a ComputerSystem element representing a file server. If this parameter is NULL, no change is made to the local access configuration. If it is not NULL, the change to the configuration consists of the following cases: 1.) If the FileServer does not already support local access to the TheElement, it will be added and made capable of exporting the filesystem as file shares. The local access point is specified by the LocalAccessPoint parameter. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points are vendor-defaulted, the corresponding entry in the LocalAccessPoints parameter should be NULL or the empty string as the LocalAccessPoint name is constructed by a vendor-default algorithm. A LocalAccessAvailable association is created between the FileServer and the TheElement. The parameters for local access are specified by the LocalAccessSetting parameter, an EmbeddedInstance element of class LocallyAccessibleFileSystemSetting. If the LocalAccessSetting parameter is NULL or the empty string, the implementation uses a default associated with the LocallyAccessibleFileSystemCapabilities of the FileSystemConfigurationService associated to the FileServer by HostedDependency (ScopedSetting). At most one of the LocalAccessSettings entries for all the file server ComputerSystems that provide local access to this filesystem shall specify an element with Write Privileges. 2) If FileServer already supports local access to the TheElement, and the LocalAccessPoint parameter is NULL or a set of empty strings, this will remove the FileServer from the configured set. If there are existing operational users of the TheElement filesystem, they will need to be informed and the implementation might have to wait to reach a consistent state before the request can be completed. 3) If FileServer already supports local access to the TheElement, and the LocalAccessPoint parameter is the same as the current configuration, then this is a request to change the settings but not the local access point. The LocalAccessSetting parameter will specify the new setting. Depending on the precise change, the filesystem may need to suspend operations. If there are existing operational users of the filesystem, they will need to be informed and the implementation might have to wait to reach a consistent state before the request can be completed. 4) If FileServer already supports local access to the TheElement, and the LocalAccessPoint parameter is different from the current configuration, then this is equivalent to removing local access and then restoring it with different settings. If there are existing operational users of the filesystem, they will need to be informed and the implementation might have to wait to reach a consistent state before the request can be completed. Note that existing operational users will not be able to reconnect as the share name may have changed.

Table 105 - Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem

Parameter Name	Qualifier	Type	Description & Notes
LocalAccessPoint	IN, OUT, REF, NULL Allowed	string	A string to use as a pathname in the name space of the file server ComputerSystem specified by the FileServer parameter. The format of the string is vendor-dependent and it should be considered opaque to the client. It could be interpreted as a hierarchical fully-qualified name for the local access point (say in a Unix-based operating environment), or it could be a drive letter (as in a Windows operating environment). A LocalAccessAvailable association is created going between the TheElement and the FileServer. The LocalAccessAvailable.LocalAccessPoint property will be set to the value of this parameter. The parameters for local access are specified by the LocalAccessSetting parameter. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points are required, then LocalAccessPoint shall not be NULL or an empty string if this is a new FileServer that does not have local access to TheElement. This is an Error. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points can be vendor-defaulted, then LocalAccessPoint can be NULL or an empty string and the implementation shall create a name using a vendor-specific algorithm. If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points cannot be vendor-defaulted, and this is a new FileServer that does not have local access to TheElement, then LocalAccessPoint shall not be NULL and the implementation shall not create a default pathname. This is an Error. On output, this parameter contains the actual LocalAccessPoint used (including any name created by vendor-default).
LocalAccessSetting	IN, EI, OUT, NULL Allowed	string	EmbeddedInstance ("SNIA_LocallyAccessibleFileSystemSetting") An embedded LocallyAccessibleFileSystemSetting element that specifies the settings to use for establishing a local access point. Each entry will be used to create or modify a LocalAccessAvailable association and will be cloned to create a LocallyAccessibleFileSystemSetting element that is scoped via ScopedSetting (or HostedDependency) to the file server ComputerSystem specified by the FileServer parameter. The clone will be associated via ElementSettingData to the LocalFileSystem. If this parameter is NULL or the empty string, and a new value is needed, the implementation shall use the default provided by the LocallyAccessibleFileSystemCapabilities associated to the FileServer parameter. The LocalAccessSetting actually used is returned as the OUT parameter.

Table 105 - Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem

Parameter Name	Qualifier	Type	Description & Notes
InUseOptions	IN	uint16	An enumerated integer that specifies the action to take if the filesystem is still in operational use when this request is made. This option is only relevant if the FileSystem needs to be made unavailable while the request is being executed.
WaitTime	IN	uint16	An integer that indicates the time in seconds to wait before performing the request on this filesystem. The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request.
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.3.1 FileSystemConfigurationService.DeleteFileSystem

This is an extrinsic method that shall delete a LocalFileSystem specified by the parameter TheElement and delete any associated elements and associations that are no longer needed. The deleted elements include the LogicalFile/Directory and FileStorage, if they were surfaced; the LocalAccessAvailable association, the LocallyAccessibleFileSystemSetting element and its associations, ElementSettingData, HostedDependency (ScopedSetting); HostedFileSystem, ResidesOnExtent, and any associations that might be orphaned by the deletion of TheElement. The LogicalDisk(s) that TheElement used shall be released but an implementation is not required to delete or re-allocate it.

Note: The WaitTime and InUseOptions parameters are supported if the FileSystemCapabilities.SupportedProperties includes the "RequireInUseOptions" option.

9.5.4 Signature and Parameters of DeleteFileSystem.

Table 106 - Parameters for Extrinsic Method FileSystemConfigurationService.DeleteFileSystem

Parameter Name	Qualifier	Type	Description & Notes
Job	OUT, REF	CIM_ConcreteJob	Reference to the job (may be null if job completed).
TheElement	IN, REF	CIM_LocalFileSystem	The filesystem element to delete.
InUseOptions	IN	uint16	An enumerated integer that specifies the action to take if TheElement is still in use when this request is made. This option is only relevant if the filesystem needs to be made unavailable while the request is being executed.
WaitTime	IN	uint16	An integer that indicates the time in seconds to wait before performing the request on TheElement filesystem. The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request.
Normal Return			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
Error Returns			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value
Invalid Combination of Values	OUT, Indication	CIM_Error	An invalid combination of named properties of an instance parameter (either reference or embedded) has been requested.

9.5.5 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

9.6 Client Considerations and Recipes

EXPERIMENTAL

Conventions used in the NAS recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is (often) used without further validation. Real code will need to be more robust.
- SMI-S uses Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.
- Error returns using the CIM_Error mechanism are not explicitly handled -- the client must implement handlers for these asynchronous returns.
- These recipes do not show the details of negotiating a setting acceptable to both client and provider.
- The recipes do not show the details of managing a Job if a method returns after setting one up.
- All the recipes show very simple examples of the operations that should be supported. Some recipes have been simplified so that they would not even be minimally useful to a real client, but only show how more complete functionality would be implemented.

In the Filesystem Manipulation Profile recipes, the following subroutines are used (and provided here as forward declarations):

```
sub CreateGoal(
        IN REF CIM_FileSystemCapabilities $fscapability,
        IN String $goalSetting,
        INOUT String $supportedFileSystemSetting);
// The above subroutine uses the $fscapability.CreateGoalSettings method
// to get the single $supportedFileSystemSetting used in these recipes.

sub GetRequiredStorageSize(
     IN REF CIM_FileSystemCapabilities $fscapability,
     IN String $fssgoal,
     IN String $ldSetting,
     OUT uint64 $expectedsize,
     OUT uint64 $minsize,
     OUT uint64 $maxsize);
// The above subroutine uses the $fscapability.GetRequiredStorageSize
// method to get the single output size used in these recipes.
```

9.6.1 Creation of a FileSystem on a Storage Extent

```
//
// DESCRIPTION
//   Goal: Create a LocalFilesystem on a LogicalDisk
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      will also be the host of the created LocalFileSystem.
//   2. The client does not negotiate to get an acceptable setting but
```

```
//      fails if one is not found
//   3. We do not use the FSCS to create a LogicalDisk from a StoragePool
//   4. We do not set up local access to a file server at this time
//
// FUNCTION CreateFileSystem
//   This function takes a given ComputerSystem and LogicalDisk and
//   constructs a filesystem that satisfies the requested property values.
// INPUT Parameters:
//   hostsystem: A reference to the ComputerSystem.
//   disk: A reference to the LogicalDisk on which to build the
//      filesystem.
//   desiredsize: An integer specifying the size of filesystem to
//      build in bytes
//   fsname: The string name of the filesystem
//   filesystemtype: An integer enumeration of the filesystem type
//      to construct
//   otherpropertyname: An array of property names with corresponding
//      values in the otherpropertyvalue parameter.
//   otherpropertyvalue: An array of property values corresponding to the
//      names in the otherpropertyname parameter.
// OUTPUT Parameters:
//   fs: A reference to the LocalFileSystem that is built by this
//      function.
//   job: A reference to a job created by the implementation if this
//      function will take a long time to complete.
// RESULT:
//   Failure return consists of fs=NULL and job=NULL
// NOTES
//   1. This recipe does not show how to use the LocalAccess functionality
//      to "mount" the file system to a mount-point of a file server.

sub CreateFileSystem(IN REF CIM_System $hostsystem,
                     IN REF CIM_LogicalDisk $disk,
                     IN uint64 $desiredsize,
                     IN String $fsname,
                     IN String $filesystemtype,
                    IN String $otherpropertyname[],    // array of property names
                     IN String $otherpropertyvalue[],   // corresponding array of
                           values
                     OUT REF CIM_FileSystem $fs,
                     OUT REF CIM_Job $job)
{
    //
    // Get the FileSystemConfigurationService of the NAS server using
    // a HostedService association
    //
    $fsconfigurators->[] = Associators($hostsystem,
                                       "CIM_HostedService",
```

```
                                    “CIM_FileSystemConfigurationService”,
                                    “Antecedent”,
                                    “Dependent”);
if ($fsconfigurators->[] == null || $fsconfigurators->[].length == 0) {
   // No FileSystemConfigurationService found -- error
   $fs = NULL;
   $job = NULL;
   return;
}
$fsconfigurator = $fsconfigurators->[0];

//
// Find FSCapabilities that supports $filesystemtype
// as the ActualFileSystemType using ElementCapabilities
// association from FSConfigurationService.
//
// There is only one Capability of a particular ActualFileSystemType
$capabilities->[] = Associators($fsconfigurator,
                                    “CIM_ElementCapabilities”,
                                    “CIM_FileSystemCapabilities”,
                                    “ManagedElement”,
                                    “Capabilities”);
if ($capabilities->[] == null || $capabilities->[].length == 0) {
   // No Capabilities found -- error
   $fs = NULL;
   $job = NULL;
   return;
}
#j = 0;
while($capability = $capabilities->[#j]) {
    if ( ($capability.ActualFileSystemType == $filesystemtype) ||
         ( ($filesystemtype == NULL) && ($capability.IsDefault) ) ) {
       if ($otherpropertyname->[] == NULL || $otherpropertyname->[].length ==
                    “” ||
           Contains(%capability.SupportedProperties, $otherpropertyname->[]))
                    {
            // This Contains function is left to the client to implement
            // found a matching capabilities element
            //
            break;
        } else {
            // Found capabilities element failed to match
           $fs = NULL;
           $job = NULL;
           return;
    }
    #j++;
}
```

```
$capability = $capabilities->[#j];

// If $filesystemtype was NULL or empty string the default was returned
if ($filesystemtype == NULL || $filesystemtype == "")
    $filesystemtype = $capability.ActualFileSystemType;

// At this point the $capability will be for $filesystemtype

//
// Call FileSystemCapabilities.CreateGoalSettings(nullTemplate, Goal) to
// get a seed goal for FileSystemSetting, or just use one of the provided
// default settings associated with the FileSystemCapabilities via
// SettingsDefineCapabilities.
//
// The function used is CreateGoal instead of CreateGoalSettings
// because the CreateGoalSettings method takes arrays
// as parameters and we only want to pass single-entry arrays
// The implementation details are left to the client.
$fssgoal = NULL;
CreateGoal($capability, NULL, $fssgoal);

//
// Inspect Goal and modify properties as desired.
//
#i = 0;
while ($otherpropertyname[#i]) {
    // funky syntax on left-hand side -- dot-operator on an a variable
    $fssgoal.$otherpropertyname[#i] = $otherpropertyvalue[#i];
    #i++;
}

//
// Call FSCSCapabilities.CreateGoalSettings(Goal-N', Goal-N) to get
// the next goal for FSSetting -- iterate until satisfied or give up
// (beware of infinite loops)  Note: we don't iterate here, just give
//  up if we don't get what we want.
//
// The function used is CreateGoal instead of CreateGoalSettings
// because the CreateGoalSettings method takes arrays
// as parameters and we only want to pass single-entry arrays
// The implementation details are left to the client.
CreateGoal($capability, $fssgoal, $fssgoal2);

#i = 0;
while ($otherpropertyname[#i]) {
    //
    // Note: this pseudocode doesn't check to see if the property named
```

```
    // in $otherpropertyname[#i] is an array.  This additional level
    // of horsing around is left as an exercise for the reader.
    //
    if ($fssgoal.$otherpropertyname[#i] != $otherpropertyvalue[#i] {
        { return NULL; }          // give up
    }
}

//
// Call FileSystemCapabilities.GetRequiredStorageSize(Goal,
// DesiredUsableCapacity) to find out how large of a
// LogicalDisk is needed.
//
// GetRequiredStorageSize returns the maximum and minimum
// sizes that might be needed to satisfy the fssgoal2 request
// If the LogicalDisk in use for the FileSystem cannot be grown
// upon demand, then it might be worth growing to $minsize (which
// would be optimistic); if there is any reason to believe that
// the user is underestimating what they will need, then it might
// be worth growing to $maxsize (pessimistic); in the normal case,
// plan to grow to $expectedsize.
//
$ldsetting = NULL;
$requiredsize = $capability.GetRequiredStorageSize(
                                        $fssgoal2,
                                        $ldsetting,   // NULL input, returns
                        setting
                                        $expectedsize,
                                        $minsize,
                                        $maxsize);

//
// If a disk of the required size is already available
//     Call CreateFileSystem(Goal, LogicalDisk)
// else
//     Create LogicalDisk (see StorageExtent recipes)
//     Call CreateFileSystem(Goal, LogicalDisk)
//
if ($requiredsize > $disk.BlockSize * $disk.NumberOfBlocks) {
    <CreateDisk>($requiredsize, $newdisk);
    $disk = $newdisk;
}
$diskArray->[0] = $disk;
$status = $fsconfigurator.CreateFileSystem(
    $fsname,
    $job,              // Job returned if necessary
    $fssgoal2,         // Filesystem Setting
    $fs,               // Filesystem returned
```

```
        $diskArray->[],    // LogicalDisk to use
        NULL               // No storagepools
        NULL,              // No settings to create LDs
        NULL,              // No size parameters
        NULL,              // No File server specified for Local Access
        NULL,              // No local access points provided
        NULL               // No local access settings
        );

    //
    // not shown:
    //    1) Managing the $job if it's not NULL,
    //    2) Looking at the status result to figure out what to do
    //    3) Managing any CIM_Errors that get returned asynchronously.
    //
    return $fs;
}
```

9.6.2 Increase the size of a FileSystem

```
//
// DESCRIPTION
//   Goal: Increase the size of a FileSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      is also the host of the LocalFileSystem being modified.
//   2. The client does not negotiate to get an acceptable setting but
//      fails if one is not found
//   3. Then desiredsize is greater than the current size
//
// FUNCTION CreateFileSystem
//   This function takes a given LocalFileSystem and a desired
//   increase in size in bytes and expands the size of the
//   filesystem by at least the desired size.
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
//   desiredsize: The desired size of the filesystem
// OUTPUT Parameters:
//   job: A reference to a job created by the implementation if this
//      function will take a long time to complete.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub IncreaseFileSystemSize(IN REF CIM_FileSystem $fs,
                           IN REF uint64 $desiredsize,
```

```
                                OUT CIM_Job $job)
{
    //
    // Get a client-side copy of the FileSystemSetting
    // associated with the CIM_FileSystem (via ElementSettingData
    // association) using GetInstance
    //
    $settings = Associators($fs,
                            "CIM_ElementSettingData",
                            "CIM_FileSystemSetting",
                            "ManagedElement",
                            "SettingData");
    if ($settings ->[] == NULL || $settings ->[].length == 0) {
       // No FileSystemSetting found -- error
       $job = NULL;
       return;
    }
// One of the settings must be marked IsCurrent -- if not, there is an error
    #i = 0;
    $setting = NULL;
    while ($settings->[#i] != NULL) {
        if ($settings->[#i].IsCurrent) {
            $setting = GetInstance($settings->[#i]);
            break;
        }
        #i++;
    }
    if ($setting == NULL) {
       $job = NULL;
       return;
    }
    $fssnewgoal = $setting;

    // Note that this syntax conflicts with earlier use of funky syntax for
    // accessing properties.  Also "add" method applied to an array-value
    // changes the array in-place
    $fssnewgoal.ObjectTypes->[].add("Bytes");
    $fssnewgoal.ObjectSizeMin->[].add($desiredsize);

    // Get the FileSystemCapabilities element from the hosting NAS Server
    //
    // a) Get the ActualFileSystemType from the FileSystemSetting
    //
    $filesystemtype = $fssnewgoal.ActualFileSystemType;

    //
    // Get the ComputerSystem for the filesystem (via HostedFileSystem association)
```

```
//  There should be exactly one.
$system = Associators($fs,
                      “CIM_HostedFileSystem”,
                      “CIM_ComputerSystem”,
                      “PartComponent”,
                      “GroupComponent”)->[0];

//
// Get the FileSystemConfigurationService from the ComputerSystem
//   via the HostedService association.  There is exactly one,
//   but check that one is found.
//
$fsconfigurators->[] = Associators($system,
                                   “CIM_HostedService”,
                                   “CIM_FileSystemConfigurationService”,
                                   “Antecedent”,
                                   “Dependent”);
if ($fsconfigurators->[] == null || $fsconfigurators->[].length == 0) {
   // No FileSystemConfigurationService found -- error
   $job = NULL;
   return;
}

$fsconfigurator = $fsconfigurators->[0];

//
// Find FSCapabilities that supports $filesystemtype
// as the ActualFileSystemType using ElementCapabilities
// association from FSConfigurationService.
//
// There is only one Capability of a particular ActualFileSystemType
$capabilities->[] = Associators($fsconfigurator,
                                  “CIM_ElementCapabilities”,
                                  “CIM_FileSystemCapabilities”,
                                  “ManagedElement”,
                                  “Capabilities”);
if ($capabilities->[] == null || $capabilities->[].length == 0) {
   // No Capabilities found -- error
   $job = NULL;
   return;
}
#j = 0;
while($capability = $capabilities->[#j]) {
    if ($capability.ActualFileSystemType == $filesystemtype)
        break;
    #j++;
}
```

```
if (#j == $capabilities->[].length) {
   // No Capabilities for this filesystem type was found -- error
   $job = NULL;
   return;
} else
   $capability = $capabilities->[#j];

//
// Call FileSystemCapabilities.GetRequiredStorageSize(NewGoal,
// DesiredUsableCapacity) to find out how large of a
// LogicalDisk is needed
//
// Changed from: $requiredsize =
                        $capability.GetRequiredStorageSize($fssnewgoal,
$ldsetting = "";
$requiredsize = GetRequiredStorageSize($capability,
                      $fssnewgoal,
                               $ldsetting,     // Returns actual setting used
                               $disksize,
                               $diskminsize,
                               $diskmaxsize);

//
// Get Underlying LogicalDisk using ResidesOnExtent association
// There must be exactly one
//
$disk = Associators($fs,
                  "CIM_ResidesOnExtent",
                  "CIM_LogicalDisk",
                  "Dependent",
                  "Antecedent")->[0];

//
// If disk is not large enough, increase size of underlying SE
//
$job = NULL;
if ($disksize < $disk.BlockSize * $disk.NumberOfBlocks) {
     <increase size of logical disk, returning a job in $job if
                        necessary -- see storage extent recipes>
}
//
// The filesystem itself doesn't need modification, so we're done
//
// This is NOT correct.  The ModifyFileSystem method must be called
// with the new file system setting so that the filesystem can be
// modified as needed.

// It isn't clear what the call would be -- probably specify NULL for
```

```
    // the InExtents parameter and the desiredsize parameter would indicate
    // that the filesystem was being resized.
    // Operationally, the appended storage space would need to be formatted
    // as inodes and their inode numbers would need to be legitimized in
    // the filesystem meta-data.
    //
    // The call would be
    //   $fsconfigurator.ModifyFileSystem(
    //     NULL,              // Keep the old element name for the filesystem
    //     $job,              // return Job if created
    //     $fssgoal,          // Goal setting
    //     $fs,               // filesystem
    //     NULL,              // Don't add any logicaldisks
    //     NULL,              // No storage pools
    //     NULL,              // No LogicalDisk settings
    //     $disksize,         // New LD size
    //     NULL,              // No File server for local access
    //     NULL,              // No Local access point name
    //     NULL,              // No Local access setting
    //     NULL,              // Default in use option
    //     NULL,              // Default wait time
    //     );
    //
}
```

9.6.3 Modify a FileSystem's Settings

```
//
// DESCRIPTION
//   Goal: Modify the settings and other properties of a LocalFileSystem
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      will also be the host of the LocalFileSystem to be modified.
//   2. The client does not negotiate to get an acceptable setting but
//      fails if one is not found.
//   3. This recipe only shows how the number of supported objects
//      of a particular type is modified.  The model can be easily
//      extended to other individual properties of the LocalFileSystem.
//   4. The CreateFileSystem method uses an array of property names
//      and values and can be useful to show how ModifyFileSystem
//      may change many propertynames in a single call at the same time.
//
// FUNCTION ModifyFileSystemObjectLimits
//   This function takes a given LocalFileSystem and a specification
//   of an object type (file and/or directories) to be supported
//   and modifies the filesystem (increases its size) so that it
//   satisfies the newly requested size.
// INPUT Parameters:
```

```
//   fs: A reference to the LocalFileSystem.
//   objecttype: The object type whose support is being modified
//   minobjects: The minimum number of objects of the specified
//      type to be supported.
//   maxobjects: The maximum number of objects of the specified
//      type to be supported.
//   expectedobjects: The client's expectations of the number of
//      objects of the specified type to be supported.
// OUTPUT Parameters:
//   objecttype: The object type whose support has being modified
//   minobjects: The minimum number of objects of the specified
//      type that will be supported by the implementation.
//   maxobjects: The maximum number of objects of the specified
//      type that will be supported by the implementation.
//   expectedobjects: The implementation's expectations of the
//      number of objects of the specified type to be supported.
//   job: A reference to the job implementing the ModifyFileSystem
//      method, if necessary.
// RESULT:
//   None
// NOTES
//   1. This recipe does not show how to specify multiple object
//      types at the same time.
//   2. This recipe does not show how to change the local access
//      setup (add or delete local access).

sub ModifyFileSystemObjectLimits(IN REF CIM_FileSystem $fs,
                                 IN OUT uint64 $objecttype,
                                 IN OUT uint64 $minobjects,
                                 IN OUT uint64 $maxobjects,
                                 IN OUT uint64 $expectedobjects,
                                 OUT REF CIM_Job $job)
{
    //
    // Get a client-side copy of the FileSystemSetting
    // associated with the CIM_FileSystem (via ElementSettingData
    // association) using GetInstance
    //
    $settings = Associators($fs,
                            "CIM_ElementSettingData",
                            "CIM_FileSystemSetting",
                            "ManagedElement",
                            "SettingData");
    if ($settings ->[] == NULL || $settings ->[].length == 0) {
       // No FileSystemSetting found -- error
       $job = NULL;
       return;
```

```
    }
// One of the settings must be marked IsCurrent -- if not, there is an error
    #i = 0;
    $setting = NULL;
    while ($settings->[#i] != NULL) {
        if ($settings->[#i].IsCurrent) {
            $setting = GetInstance($settings->[#i]);
            break;
        }
        #i++;
    }
    if ($setting == NULL) {
       $job = NULL;
       return;
    }
    $fssnewgoal = $setting;

    // Get the FileSystemCapabilities element from the hosting NAS Server
    //
    // a) Get the ActualFileSystemType from the FileSystemSetting
    //
    $filesystemtype = $setting.ActualFileSystemType;

    //
    // Get the ComputerSystem for the filesystem (via HostedFileSystem association)
    //  There should be exactly one.
    $system = Associators($fs,
                          "CIM_HostedFileSystem",
                          "CIM_ComputerSystem",
                          "PartComponent",
                          "GroupComponent")->[0];

    //
    // Get the FileSystemConfigurationService from the ComputerSystem
    //   via the HostedService association.  There is exactly one.
    //
    $fsconfigurators->[] = Associators($system,
                                   "CIM_HostedService",
                                   "CIM_FileSystemConfigurationService",
                                   "Antecedent",
                                   "Dependent");
    if ($fsconfigurators->[] == null || $fsconfigurators->[].length == 0) {
       // No FileSystemConfigurationService found -- error
       $job = NULL;
       return;
    }
```

```
$fsconfigurator = $fsconfigurators->[0];

//
// Find FSCapabilities that supports $filesystemtype
// as the ActualFileSystemType using ElementCapabilities
// association from FSConfigurationService.
//
// There is only one Capability of a particular ActualFileSystemType
$capabilities->[] = Associators($fsconfigurator,
                                “CIM_ElementCapabilities”,
                                “CIM_FileSystemCapabilities”,
                                “ManagedElement”,
                                “Capabilities”);
if ($capabilities->[] == null || $capabilities->[].length == 0) {
   // No Capabilities found -- error
   $job = NULL;
   return;
}
#j = 0;
while($capability = $capabilities->[#j]) {
    if ($capability.ActualFileSystemType == $filesystemtype)
        break;
    #j++;
}
if (#j == $capabilities->[].length) {
   $job = NULL;
   return;
} else
   $capability = $capabilities->[#j];

//
// Find the index in the object arrays that contains
// the object type of interest
//
#i = 0;
while($typ = $fssnewgoal.ObjectTypes->[#i]) {
    if ($typ == $objecttype)
        { break; }
    #i++;
}
//
// if the specified type isn’t there, add it
//
if ($typ != $objecttype) {
    $fssnewgoal.ObjectTypes->[#i] = $objecttype;
}
```

```
//
// modify the other params associated with the object type
//
$fssnewgoal.NumberOfObjectsMin->[#i] = $minobjects;
$fssnewgoal.NumberOfObjectsMax->[#i] = $maxobjects;
$fssnewgoal.NumberOfObjects->[#i] = $expectedobjects;

//
// Call FSCSCapabilities.CreateGoalSettings(Goal-N', Goal-N) to get the next
// goal for FSSetting -- iterate until satisfied or give up (beware
// infinite loops)  Note: we don't iterate here, just give up.
//
// The function used is CreateGoal instead of CreateGoalSettings
// because the CreateGoalSettings method takes arrays
// as parameters and we only want to pass single-entry arrays
// The implementation details are left to the client.
CreateGoal($capability, $fssnewgoal, $fssgoal2);
if ($fssgoal2.ActualFileSystemType != $filesystemtype) {
   $job = NULL;
   return;
}

// Since this may increase the size of the file system it is necessary to
// pass in a new extent or a new logical disk or a pool that can provide
// the storage.

//
// call ModifyFilesystem (management of $job and any CIM_Error not
// shown)
//
$fsconfigurator.ModifyFileSystem(
       NULL,              // Keep the old element name for the filesystem
       $job,              // return Job if created
       $fssgoal2,         // Goal setting
       $fs,               // filesystem
       NULL,              // Don't add any logicaldisks
       NULL,              // No storage pools
       NULL,              // No LogicalDisk settings
       NULL,              // No LD sizes
       NULL,              // No File server for local access
       NULL,              // No Local access point name
       NULL,              // No Local access setting
       NULL,              // Default in use option
       NULL,              // Default wait time
       );

return $fs;
```

```
}
```

9.6.4 Delete a FileSystem and return underlying StorageExtent

```
//
// DESCRIPTION
//   Goal: Delete a FileSystem and return underlying LogicalDisk
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      is also the host of the created LocalFileSystem.
//   2. The filesystem is built on a single LogicalDisk
//   3. The LogicalDisk is not automatically returned to a StoragePool
//      but is left allocated to the NAS Server and available for use
//      by a filesystem client.
//   4. No job is needed
//
// FUNCTION DeleteFileSystem
//   This function deletes a given LocalFileSystem and
//   returns a reference to the LogicalDisk on which it resided
// INPUT Parameters:
//   fs: A reference to the LocalFileSystem.
// OUTPUT Parameters:
//   disk: A reference to the LogicalDisk is returned.
// RESULT:
//   Success or Failure
// NOTES
//   1. This recipe does not show how to clean up any local access
//      or file shares that may have been set up for accessing the
//      filesystem.
//   2. To "wipe" or zero out the filesystem, the client must either
//      use client-level operations over a FileSystem or FileShare
//      prior to deleting the filesystem, or by vendor-specific
//      operations on the LogicalDisk after deleting the filesystem.
//

sub DeleteFileSystem(IN REF CIM_FileSystem $fs, OUT REF CIM_LogicalDisk $disk)
{
    //
    // Get underlying LogicalDisk using ResidesOnExtent association
    // In SMI-S 1.2. we assume that there will be exactly one
    //
    $disks->[] = Associators($fs,
                         "CIM_ResidesOnExtent",
                         "CIM_LogicalDisk",
                         "Dependent",
                         "Antecedent")->[0];
    if ($disks->[] == null || $disks->[].length == 0) {
       // No LogicalDisk found -- error
```

```
      $disk = NULL;
      return;
   }
   $disk = $disks->[0];

   //
   // Get the NAS Server of the FileSystem using
   // a HostedFileSystem association. There should be
   // exactly one filesystem host.
   $hosts->[] = Associators($fs,
                            "CIM_HostedFileSystem",
                            "CIM_ComputerSystem",
                            "Antecedent",
                            "Dependent");
   if ($hosts->[] == null || $hosts->[].length == 0) {
      // No ComputerSystem found -- error
      $disk = NULL;
      return;
   }
   $hostsystem= $hosts->[0];

   //
   // Get the FileSystemConfigurationService of the NAS server using
   // a HostedService association
   //
   $fsconfigurators->[] = Associators($hostsystem,
                                "CIM_HostedService",
                                "CIM_FileSystemConfigurationService",
                                "Antecedent",
                                "Dependent");
   if ($fsconfigurators->[] == null || $fsconfigurators->[].length == 0) {
      // No FileSystemConfigurationService found -- error
      $fs = NULL;
      $job = NULL;
      return;
   }
   $fsconfigurator = $fsconfigurators->[0];

   //
   // Call DeleteFileSystem(FS) (error checking not shown)
   //
   $fsconfigurator.DeleteFileSystem($job, $fs);

   return;
}
```

9.6.5 Make a FileSystem Locally Accessible from a File Server ComputerSystem

```
//
// DESCRIPTION
//   GOAL: Get a LocallyAccessibleFileSystemCapabilities from a
//         filesystem host that is dependent on a specific file server
//         and supports the properties specified in the array
//         parameter $propertynames[].
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      will also be the host of any LocalFileSystem that will be
//      made locally accessible using a capabilities element.
//
// FUNCTION GetLocallyAccessibleFileSystemCapabilities
//   This function takes a filesystem host ComputerSystem and
//      gets a capabilities element for making a filesystem
//      locally accessible on a file server ComputerSystem.
// INPUT Parameters:
//   hostsystem: A reference to the ComputerSystem that hosts
//      filesystems.
//   fileserver: A reference to the file server ComputerSystem that
//      provides local access to filesystems.
//   propertynames: An array of property names that the capabilities
//      element should support.
// OUTPUT Parameters:
//   allcapabilities: An array of references to the capabilities
//      for local access on the file server.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub GetLocallyAccessibleFileSystemCapabilities(
         IN REF CIM_ComputerSystem $hostsystem,
         IN REF CIM_ComputerSystem $fileserver,
         IN String $propertynames[],
         OUT REF SNIA_LocallyAccessibleFileSystemCapabilities $allcapabilities[])
{
    //
    // Get the FileSystemConfigurationService from the ComputerSystem
    //   $hostsystem via the HostedService association
    //
    $fsconfigurators->[] = Associators($hostsystem,
                                  “CIM_HostedService”,
                                  “CIM_FileSystemConfigurationService”,
                                  “Antecedent”,
                                  “Dependent”);
```

```
    #i = 0;
    #k = 0; // the index for $allcapabilities.
    while ($fsconfigurator = $fsconfigurators->[#i]) {
        #i++;
        //
        // Find LocallyAccessibleFileSystemCapabilities that supports the
        // file server using ElementCapabilities association from
        // FSConfigurationService.
        // If client does not care about the file server ($fileserver = NULL),
        // return all the LocallyAccessibleFileSystemCapabilities that
        // are associated to the FileSystemConfigurationService
        // There is one and only one LocallyAccessibleFileSystemCapabilities
        // for each server+FileSystemConfigurationService pair.
        //  The SupportedProperties property lists the supported setting
        // properties.
        //
        $capabilities->[] = Associators($fsconfigurator,
                                        “CIM_ElementCapabilities”,

                          “SNIA_LocallyAccessibleFileSystemCapabilities”,
                                        “ManagedElement”,
                                        “Capabilities”);
        // Skip to next if empty
        if ($capabilities->[] == NULL ||$capabilities->[].length == 0) continue;
        #j = 0;
        while($capability = $capabilities->[#j]) {
            #j++;
            if (propertyname == NULL || propertyname == “” ||
                Contains($capability.SupportedProperties, propertyname)) {
                // If the server is null then skip the next step
                if ($server != NULL) {
                    $capservers[] = Associators($capability,
                                                “SNIA_ScopedCapability”,
                                                “CIM_ComputerSystem”,
                                                “Dependent”,
                                                “Antecedent”);
                    if ($capservers == NULL || $capservers->[].length != 1 ||
                        $server != $capservers->[0])
                        continue;
                }
                $allcapabilities->[#k] = $capability;
                #k++;
            }
        }
    }
    return;
}
```

9.6.6 Get the Local Access Setting for a FileSystem on a File Server ComputerSystem

```
// DESCRIPTION
//   GOAL: Get a LocallyAccessibleFileSystemSetting from a
//         filesystem host that is dependent on a specific file server
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//   1. The ComputerSystem host of the FileSystemConfigurationService
//      will also be the host of any LocalFileSystem that will be
//      made locally accessible using a capabilities element.
//
// FUNCTION GetLocallyAccessibleFileSystemSetting
//   This function takes a filesystem host ComputerSystem and
//      gets a capabilities element for making a filesystem
//      locally accessible on a file server ComputerSystem.
// INPUT Parameters:
//   filesystem: A reference to the LocalFileSystem that is to
//      be made locally accessible from a file server.
//   fileserver: A reference to the file server ComputerSystem that
//      provides local access to filesystems.
// OUTPUT Parameters:
//   setting: An embedded instance of a LocallyAccessibleFileSystemSetting
//      that supports making the filesystem locally accessible.
// RESULT:
//   Success or Failure
// NOTES
//   1.

sub GetLocallyAccessibleFileSystemSetting(
              IN REF CIM_FileSystem $filesystem,
              IN REF CIM_ComputerSystem $fileserver,
              OUT String("SNIA_LocallyAccessibleFileSystemSetting") $setting)
{
    // Does this fileserver have local access to this filesystem
    //   -- if not, there is no setting!
    $localaccess->[] = ReferenceNames($filesystem,
                                "SNIA_LocalAccessAvailable",
                                "FileSystem");
    if ($localaccess->[] == NULL || $localaccess->[].length == 0)
        return;

    //
    // Get all the LocallyAccessibleFileSystemSettings
    // associated with the CIM_FileSystem (via ElementSettingData
    //
    $assoc = References($filesystem,
                        "CIM_ElementSettingData",
                        "ManagedElement");
```

```
    if ($assoc->[] == NULL || $assoc->[].length == 0) {
        // This is an ERROR but for now we return with no results
        return;
    }

    #i = 0;
    while ($assoc->[#i] != NULL) {
        if ($assoc->[#i].IsCurrent) {
            // Is this scoped to the fileserver?
            $servers = Associators($assoc->[#i].SettingData,
                                   "CIM_ScopedSetting",
                                   "CIM_ComputerSystem",
                                   "Dependent",
                                   "Antecedent");
            if ($servers->[] != NULL && $servers->[].length != 0 && $servers->[0]
                          == $fileserver) {
                $setting = GetInstance($assoc->[#i].SettingData);
                return;
            }
        }
        #i++;
    }
    $setting = NULL;
}
```

EXPERIMENTAL

9.6.7 Filesystem Manipulation Supported Capabilities Patterns

Table 107, “Filesystem Manipulation Supported Capabilities Patterns” lists the patterns that are formally recognized by SMI-S 1.1.0 for determining capabilities of various NAS implementations:

Table 107 - Filesystem Manipulation Supported Capabilities Patterns

Supported ActualFileSystem Types	Supported Synchronous Methods	Supported Asynchronous Methods	Initial Availability
Any	none	none	none
Any	CreateFileSystem, DeleteFileSystem, ModifyFileSystem, CreateGoalSettings, GetRequiredStorageSizes	none	Any
Any	CreateGoalSettings, GetRequiredStorageSizes	CreateFileSystem, DeleteFileSystem, ModifyFileSystem	Any

9.7 Registered Name and Version

Filesystem Manipulation version 1.3.0

9.8 CIM Elements

Table 99 describes the CIM elements for Filesystem Manipulation.

Table 108 - CIM Elements for Filesystem Manipulation

Element Name	Requirement	Description
9.8.1 CIM_Dependency (Uses Directory Services From)	Conditional	Conditional requirement: Required if LocalFileSystem.DirectoryServiceUsage is either \Required\'or\'Optional\'.' Associates a ComputerSystem that indicates a directory service that supports the dependent LocalFileSystem.
9.8.2 CIM_Directory (Root Directory)	Conditional	Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1.. The root directory of a LocalFileSystem that is always present when a FileSystem is created. This is retained for backward compatibility with SMI Specification 1.1.
9.8.3 CIM_ElementCapabilities (FS Configuration Capabilities)	Mandatory	In this subprofile, associates the Filesystem Configuration Service to the Capabilities element that represents the capabilities that it supports.
9.8.4 CIM_ElementCapabilities (Local Access Configuration Capabilities)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either \LocalAccessRequired,Defaulted\'or\'LocalAccessRequired,NotDefaulted\'.' In this subprofile, associates the Filesystem Configuration Service to the Capabilities instance that represents the capabilities for Local Access that it supports.
9.8.5 CIM_ElementCapabilities (Non-Default)	Optional	In this subprofile, associates the Filesystem Configuration Service to the FileSystemCapabilities elements that represent all the types of filesystems that are not the default type of file system and can be configured.
9.8.6 CIM_ElementSettingData (Attached to Filesystem)	Optional	Associates a FileSystemSetting element to a LocalFileSystem. One of these association elements is created by CreateFileSystem when the LocalFileSystem is first created. The profile does not specify how other instances of this association may be surfaced by the implementation.

Table 108 - CIM Elements for Filesystem Manipulation

Element Name	Requirement	Description
9.8.7 CIM_ElementSettingData (Local Access Required)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either \LocalAccessRequired,Defaulted\'or\'LocalAccessRequired,NotDefaulted\'.'Associates a LocalFileSystem and the LocallyAccessibleFileSystemSetting elements.
9.8.8 CIM_FileStorage (Root Directory)	Conditional	Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1.. Associates the root Directory to its parent LocalFileSystem.
9.8.9 CIM_FileStorage (Shared Files and Directories)	Conditional	Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1.. Associates an exported Logical File or Directory to the LocalFileSystem that contains it.
9.8.10 CIM_HostedDependency (Attached to File System)	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true.. Associates a Local Access configuration setting to the file server ComputerSystem that provides the operational scope for its functionality.
9.8.11 CIM_HostedDependency (Predefined Capabilities)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either \LocalAccessRequired,Defaulted\'or\'LocalAccessRequired,NotDefaulted\'.' Associates a Local Access Capabilities to the File Server that provides the operational scope for its functionality. All of the Settings associated to the referenced Capabilities element must be scoped by the same File Server ComputerSystem. This scoping allows the CreateGoalSetting method of the Capabilities element to know which File Server provides the scope for any Goal element that it creates.
9.8.12 CIM_HostedDependency (Predefined Setting)	Optional	Associates a predefined SNIA_LocallyAccessibleFileSystemSetting to the file server ComputerSystem that provides the operational scope for its functionality.
9.8.13 CIM_HostedFileSystem	Mandatory	Associates a LocalFileSystem to the ComputerSystem that hosts it.

Table 108 - CIM Elements for Filesystem Manipulation

Element Name	Requirement	Description
9.8.14 CIM_HostedService	Mandatory	In this subprofile, associates the Filesystem Configuration Service to the hosting ComputerSystem. This is expected to be the top-level ComputerSystem of the parent Filesystem Profile.
9.8.15 CIM_LogicalFile (Shared Files and Directories)	Conditional	Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1.. A LogicalFile (or Directory subclass) that is exported as a FileShare is also visible as a sub-element of the LocalFileSystem. Maybe this class should be defined only in the File Export subprofile.
9.8.16 SNIA_ElementCapabilities (Default)	Optional	This entry represents the single default FileSystemCapabilities element for the Filesystem Configuration Service.
9.8.17 SNIA_FileSystemCapabilities	Mandatory	This element represents the Capabilities of the Filesystem Configuration Service for managing Filesystems. The Service can be associated with multiple FileSystemCapabilities elements, one per ActualFileSystemType property value. For each value that is in the array property FileSystenConfigurationCapabilities.SupportedActualFileSystemTypes, there will be exactly one corresponding FileSystemCapabilities element with the matching ActualFileSystemType property.
9.8.18 SNIA_FileSystemConfigurationCapabilities	Mandatory	This element represents the management Capabilities of the Filesystem Configuration Service.
9.8.19 SNIA_FileSystemConfigurationService	Mandatory	The Filesystem Configuration Service provides the methods to manipulate file systems.
9.8.20 SNIA_FileSystemSetting (Attached to FileSystem)	Optional	This element represents the configuration settings of a LocalFileSystem. One instance of this class is created by the CreateFileSystem extrinsic method when the LocalFileSystem was created. This profile does not specify how other instances of this class might be created.

Table 108 - CIM Elements for Filesystem Manipulation

Element Name	Requirement	Description
9.8.21 SNIA_FileSystemSetting (Predefined FS Settings)	Optional	This element represents sample configuration settings usable for creating or modifying a LocalFileSystem. It represents "predefined" settings supported by the FileSystemConfigurationService and is associated with a FileSystemCapabilities element by a SettingsDefineCapabilities association. The FileSystemSetting.ActualFileSystemType property must specify the same value as the associated FileSystemCapabilities.ActualFileSystemType property.
9.8.22 SNIA_LocalAccessAvailable	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..Associates a LocalFileSystem to a File Server Computer System that can export files or directories as shares.
9.8.23 SNIA_LocalFileSystem	Mandatory	Represents a LocalFileSystem hosted by and made available through a ComputerSystem (usually the top-level ComputerSystem of a Filesystem Profile).
9.8.24 SNIA_LocallyAccessibleFileSystemCapabilities	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either \LocalAccessRequired,Defaulted\'or\'LocalAccessRequired,NotDefaulted\'.'The element represents the Local Access configuration Capabilities of the File System Configuration Service. This class provides a CreateGoalSettings method that will return a SNIA_LocallyAccessibleFileSystemSetting element as an EmbedddInstance that may be used for making a file system locally accessible to a file server ComputerSystem (by the methods CreateFileSystem and ModifyFileSystem). Since the returned EmbeddedInstance setting element is an instance of a ScopedSetting class, it must be associated with a ComputerSystem via ScopedSettingData when it is instantiated.

Table 108 - CIM Elements for Filesystem Manipulation

Element Name	Requirement	Description
9.8.25 SNIA_LocallyAccessibleFileSystemSetting	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..This element represents the configuration settings of a LocalFileSystem that has a contained file or directory that has been made locally accessible from a file server ComputerSystem. This Setting provides further details on the functionality supported and the parameters of that functionality when locally accessible.
9.8.26 SNIA_SettingsDefineCapabilities (Predefined FS Settings)	Optional	These Setting elements provide detailed information about the FileSystemSettings supported by the associated FileSystemCapabilities element.
9.8.27 SNIA_SettingsDefineCapabilities (Predefined Local Access Settings)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either \LocalAccessRequired,Defaulted\'or\'LocalAccessRequired,NotDefaulted\'.' The Setting elements that are associated to this Capabilities element are scoped to the File Server ComputerSystem that provides the operational context for local access.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA SNIA_LocalFileSystem	Mandatory	CQL -Creation of a LocalFileSystem element.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_LocalFileSystem	Mandatory	Modification of a LocalFileSystem element.

9.8.1 CIM_Dependency (Uses Directory Services From)

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Requirement: Required if LocalFileSystem.DirectoryServiceUsage is either 'Required' or 'Optional'.

Table 109 describes class CIM_Dependency (Uses Directory Services From).

Table 109 - SMI Referenced Properties/Methods for CIM_Dependency (Uses Directory Services From)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The ComputerSystem that indicates the directory service(s) that support user, group and other security principal identities for a filesystem.
Dependent		Mandatory	The LocalFileSystem whose use of user, group, and other security principal identities is supported by the antecedent ComputerSystem.

9.8.2 CIM_Directory (Root Directory)

Created By: Extrinsic: CreateFileSystem
Modified By: Extrinsic: ModifyFileSystem
Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem
Requirement: Required if parent profile is backward compatible to SMI Specification v1.1..

9.8.3 CIM_ElementCapabilities (FS Configuration Capabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 110 describes class CIM_ElementCapabilities (FS Configuration Capabilities).

Table 110 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (FS Configuration Capabilities)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The Filesystem Configuration Service
Capabilities		Mandatory	The Filesystem Configuration Capabilities element

9.8.4 CIM_ElementCapabilities (Local Access Configuration Capabilities)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either 'Local Access Required, Defaulted' or 'Local Access Required, Not Defaulted'.

Table 111 describes class CIM_ElementCapabilities (Local Access Configuration Capabilities).

Table 111 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (Local Access Configuration Capabilities)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The Filesystem Configuration Service
Capabilities		Mandatory	The Filesystem Configuration Capabilities element

9.8.5 CIM_ElementCapabilities (Non-Default)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 112 describes class CIM_ElementCapabilities (Non-Default).

Table 112 - SMI Referenced Properties/Methods for CIM_ElementCapabilities (Non-Default)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	
ManagedElement		Mandatory	

9.8.6 CIM_ElementSettingData (Attached to Filesystem)

Created By: Extrinsic: CreateFileSystem
Modified By: Extrinsic: ModifyFileSystem
Deleted By: Extrinsic: DeleteFileSystem
Requirement: Optional

Table 113 describes class CIM_ElementSettingData (Attached to Filesystem).

Table 113 - SMI Referenced Properties/Methods for CIM_ElementSettingData (Attached to Filesystem)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The LocalFileSystem element representing a filesystem.
SettingData		Mandatory	The configuration of the LocalFileSystem.

9.8.7 CIM_ElementSettingData (Local Access Required)

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either 'Local Access Required, Defaulted' or 'Local Access Required, Not Defaulted'.

Table 114 describes class CIM_ElementSettingData (Local Access Required).

Table 114 - SMI Referenced Properties/Methods for CIM_ElementSettingData (Local Access Required)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The LocalFileSystem that is being made locally accessible.
SettingData		Mandatory	The local access settings of the LocalFileSystem, specified on creation or modification.

9.8.8 CIM_FileStorage (Root Directory)

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem

Requirement: Required if parent profile is backward compatible to SMI Specification v1.1..

Table 115 describes class CIM_FileStorage (Root Directory).

Table 115 - SMI Referenced Properties/Methods for CIM_FileStorage (Root Directory)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The LocalFileSystem that contains the associated root Directory.
PartComponent		Mandatory	The Root Directory of the LocalFileSystem.

9.8.9 CIM_FileStorage (Shared Files and Directories)

Created By: Extrinsic: CreateExportedShare or ModifyExportedShare

Modified By: Static

Deleted By: Extrinsic: DeleteExportedShare or ModifyExportedShare

Requirement: Required if parent profile is backward compatible to SMI Specification v1.1..

Table 116 describes class CIM_FileStorage (Shared Files and Directories).

Table 116 - SMI Referenced Properties/Methods for CIM_FileStorage (Shared Files and Directories)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The LocalFileSystem that contains the LogicalFile or Directory.
PartComponent		Mandatory	An exported File or Directory of the LocalFileSystem.

9.8.10 CIM_HostedDependency (Attached to File System)

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..

Table 117 describes class CIM_HostedDependency (Attached to File System).

Table 117 - SMI Referenced Properties/Methods for CIM_HostedDependency (Attached to File System)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The Scoping File Server ComputerSystem.
Dependent		Mandatory	The Local Access Setting that is scoped by the file server ComputerSystem.

9.8.11 CIM_HostedDependency (Predefined Capabilities)

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either 'Local Access Required, Defaulted' or 'Local Access Required, Not Defaulted'.

Table 118 describes class CIM_HostedDependency (Predefined Capabilities).

Table 118 - SMI Referenced Properties/Methods for CIM_HostedDependency (Predefined Capabilities)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The Scoping file server ComputerSystem.
Dependent		Mandatory	The SNIA_LocallyAccessibleFileSystemCapabilities that is scoped by the file server ComputerSystem.

9.8.12 CIM_HostedDependency (Predefined Setting)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 119 describes class CIM_HostedDependency (Predefined Setting).

Table 119 - SMI Referenced Properties/Methods for CIM_HostedDependency (Predefined Setting)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The Scoping file server ComputerSystem.
Dependent		Mandatory	The Local Access Setting that is scoped by the file server ComputerSystem.

9.8.13 CIM_HostedFileSystem

Created By: Extrinsic: CreateFileSystem
Modified By: Extrinsic: ModifyFileSystem
Deleted By: Extrinsic: DeleteFileSystem
Requirement: Mandatory

Table 120 describes class CIM_HostedFileSystem.

Table 120 - SMI Referenced Properties/Methods for CIM_HostedFileSystem

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The ComputerSystem that hosts a LocalFileSystem.
PartComponent		Mandatory	The hosted filesystem.

9.8.14 CIM_HostedService

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 121 describes class CIM_HostedService.

Table 121 - SMI Referenced Properties/Methods for CIM_HostedService

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The Filesystem Configuration Service.
Antecedent		Mandatory	The hosting ComputerSystem.

9.8.15 CIM_LogicalFile (Shared Files and Directories)

Created By: Extrinsic: CreateExportedShare or ModifyExportedShare
Modified By: Extrinsic: CreateExportedShare or ModifyExportedShare
Deleted By: Extrinsic: DeleteExportedShare or ModifyExportedShare
Requirement: Required if parent profile is backward compatible to SMI Specification v1.1..

Table 122 describes class CIM_LogicalFile (Shared Files and Directories).

Table 122 - SMI Referenced Properties/Methods for CIM_LogicalFile (Shared Files and Directories)

Properties	Flags	Requirement	Description & Notes
CSCreationClassName		Mandatory	CIM Class Name of the ComputerSystem that hosts the Filesystem containing this file.
CSName		Mandatory	Name property of the ComputerSystem that hosts the Filesystem of this file.
FSCreationClassName		Mandatory	CIM Class Name of the LocalFileSystem on the ComputerSystem that contains this file.
FSName		Mandatory	Name of the LocalFileSystem that contains this file.
CreationClassName		Mandatory	CIM Class of this instance of LogicalFile.
Name		Mandatory	The unique Name of this LogicalFile, weak with respect to a containing Directory.
ElementName		Mandatory	The pathname from the root of the containing LocalFileSystem to this LogicalFile. The root of the LocalFileSystem is indicated if this is NULL or the empty string. The format of the pathname is specific to the LocalFileSystem's FileSystemType. If it is a sequence of directories from the root, the separator string is specified by the SNIA_LocalFileSystem.PathNameSeparatorString property.
FileSize		Optional	The size of the file, in bytes.
CreationDate		Optional	A timestamp indicating when the file was created.
LastModified		Optional	A timestamp indicating when the file was last modified.

9.8.16 SNIA_ElementCapabilities (Default)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 123 describes class SNIA_ElementCapabilities (Default).

Table 123 - SMI Referenced Properties/Methods for SNIA_ElementCapabilities (Default)

Properties	Flags	Requirement	Description & Notes
Characteristics		Optional	
Capabilities		Mandatory	
ManagedElement		Mandatory	

9.8.17 SNIA_FileSystemCapabilities

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 124 describes class SNIA_FileSystemCapabilities.

Table 124 - SMI Referenced Properties/Methods for SNIA_FileSystemCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the FileSystemCapabilities element of a Filesystem Configuration Service.
ElementName		Mandatory	A user-friendly name for this Capabilities element.
ActualFileSystemType		Mandatory	This identifies the type of filesystem that this FileSystemCapabilities represents.
SupportedProperties		Mandatory	This is the list of configuration properties (of FileSystemSetting) that are supported for specification at creation time by this FileSystemCapabilities element.

Table 124 - SMI Referenced Properties/Methods for SNIA_FileSystemCapabilities

Properties	Flags	Requirement	Description & Notes
CreateGoalSettings()		Mandatory	This extrinsic method supports the creation of a set of FileSystemSettings that is a supported variant of an array of FileSystemSettings passed in as an embedded IN parameter. The method returns the supported FileSystemSetting in an array of embedded OUT parameters. This profile only supports arrays with a single entry.
GetRequiredStorage Size()		Optional	This extrinsic method supports determining the storage space requirements for a filesystem specified by the combination of a FileSystemSetting and a StorageSetting. The StorageSetting specifies the required redundancy, multiple Logical Disk usage, and other storage mapping considerations, while the FileSystemSetting transforms client quality-of-service specifications to storage resource requirements.

9.8.18 SNIA_FileSystemConfigurationCapabilities

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 125 describes class SNIA_FileSystemConfigurationCapabilities.

Table 125 - SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for this element representing the capabilities of a Filesystem Configuration Service.
ElementName		Mandatory	A user-friendly name for this Capabilities element.
SupportedActualFile SystemTypes		Mandatory	The Service can be associated with multiple Capabilities elements, one per ActualFileSystemType property value. This property lists all of the supported ActualFileSystemTypes. Each entry in this array must have exactly one corresponding FileSystemCapabilities element with that entry as the value of the ActualFileSystemType property.
SupportedSynchrono usMethods	N	Mandatory	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called synchronously. A supported method shall be listed in this property or in the SupportedAsynchronousMethods property or both.

Table 125 - SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedAsynchronousMethods	N	Mandatory	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called asynchronously. A supported method shall be listed in this property or in the SupportedSynchronousMethods property or both.
InitialAvailability		Mandatory	This property represents the state of availability of a LocalFileSystem on initial creation using the FileSystemConfigurationService associated with this Capabilities element.
LocalAccessibilitySupport		Optional	This specifies whether a LocalFileSystem created or modified by this FileSystemConfigurationService needs to be made locally accessible at a local access point before a file server ComputerSystem can make it available to operational clients or for export as a share. This is typical of some NAS and filesystem implementations. If not specified, the default is "Local Access Not Required".

Table 125 - SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
BlockStorageCreationSupport		Optional	BlockStorageCreationSupport is an ordered array of enumerated values that place a number of restrictions on the use of parameters for CreateFileSystem and ModifyFileSystem. 1. The first entry is an enumerated value that specifies if an already existing LogicalDIsk may be used -- this is either required, optional, or not allowed. "Not Allowed" indicates that the Pools and ExtentSettings parameters must be used to create LogicalDisk(s) for this filesystem and the InExtents parameter must be NULL. "Optional" indicates that either the Pools and ExtentSettings parameters or the InExtents parameter should be specified, but not both. "Required" indicates that the InExtents parameter may be specified and the Pools and ExtentSettings parameters must be NULL. 2. (optional) An integer that specifies an upper limit to the number of StorageElements that can be specified, either as InExtents parameters or as Pools and ExtentSettings. 3. (optional) An integer that specifies the number of distinct pools that the Pools parameters can specify -- zero, if Pools is not supported or if there is no limit, and a specific number if there is a limit. In practice we expect that the value will be either zero or one. 4. (optional) A boolean value, represented by '0' for false and '1' for true, that indicates whether an entry in the ExtentSettings array parameter can be NULL (indicating that a default setting is to be used).

Table 125 - SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
DirectoryServerParameterSupported		Optional	This enumeration indicates support for the DirectoryServer parameter to the extrinsic method FileSystemConfigurationService.CreateFileSystem(). The options are: 'Not Used' indicates that the filesystem does not support security principal information associated with filesystem objects. The LocalFileSystem will not be associated to a DirectoryServer. 'Supported' indicates that the filesystem supports security principal information associated with filesystem objects. The LocalFileSystem will be associated to a directory server ComputerSystem. And the DirectoryServer parameter of CreateFileSystem is required. If it is not specified, it will be defaulted to the FileServer parameter in the same call. If the FileServer parameter is also not specified, the DirectoryServer parameter will be defaulted to the host of the FileSystemConfigurationService. 'Supported, Defaulted to FileServer' indicates that the filesystem supports security principal information associated with filesystem objects. The LocalFileSystem will be associated to a directory server ComputerSystem. The DirectoryServer parameter of CreateFileSystem is NOT supported, but is automatically defaulted to the FileServer parameter of the same call. If the FileServer parameter is not specified, the DirectoryServer parameter will be defaulted to the host of the FileSystemConfigurationService. 'Supported, Defaulted to FileSystem host' indicates that the filesystem supports security principal information associated with filesystem objects. The LocalFileSystem will be associated to a directory server ComputerSystem. The DirectoryServer parameter of CreateFileSystem is NOT supported, but is automatically defaulted to the host of the FileSystem created by CreateFileSystem().

9.8.19 SNIA_FileSystemConfigurationService

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 126 describes class SNIA_FileSystemConfigurationService.

Table 126 - SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationService

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Service.
SystemCreationClassName		Mandatory	The CIM Class name of the ComputerSystem hosting the Service.
SystemName		Mandatory	The Name property of the ComputerSystem hosting the Service.
CreationClassName		Mandatory	The CIM Class name of the Service.
Name		Mandatory	The unique name of the Service.
CreateFileSystem()		Mandatory	Creates a LocalFileSystem as specified by parameters and Capabilities of the service and returns a reference to it. If appropriate and supported, a Job may be created and a reference to the Job will be returned.
ModifyFileSystem()		Optional	Modifies a LocalFileSystem indicated by a reference and as specified by referenceparameters and Capabilities of the service. If appropriate and supported, a Job may be created and a reference to the Job will be returned.
DeleteFileSystem()		Mandatory	Deletes a LocalFileSystem indicated by reference. If appropriate and supported, a Job may be created and a reference to the Job will be returned.

9.8.20 SNIA_FileSystemSetting (Attached to FileSystem)

Created By: Extrinsic: CreateFileSystem
Modified By: Extrinsic: ModifyFileSystem
Deleted By: Extrinsic: DeleteFileSystem
Requirement: Optional

Table 127 describes class SNIA_FileSystemSetting (Attached to FileSystem).

Table 127 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Attached to File-System)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for a FileSystemSetting element.
ElementName		Mandatory	A client defined user-friendly name for this FileSystemSetting element.
ActualFileSystemType		Mandatory	This identifies the type of filesystem that this FileSystemSetting represents.
DataExtentsSharing		Optional	This allows the creation of data blocks (or storage extents) that are shared between files.

Table 127 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Attached to File-System)

Properties	Flags	Requirement	Description & Notes
CopyTarget		Optional	This specifies if support should be provided for using the created filesystem as a target of a Copy operation.
FilenameCaseAttributes		Mandatory	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		Mandatory	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
NumberOfObjectsMin		Optional	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[] that will be supportable by the LocalFileSystem configured by this FileSystemSetting element.
NumberOfObjectsMax		Optional	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[] that can be supported by the LocalFileSystem configured by this FileSystemSetting element.
NumberOfObjects		Optional	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		Optional	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		Optional	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[] that will be supported by the LocalFileSystem configured by this FileSystemSetting element.
ObjectSizeMax		Optional	This is an array that specifies the maximum size of an object of the type specified by the corresponding entry in ObjectTypes[] that can be supported by the LocalFileSystem configured by this FileSystemSetting element.
FilenameStreamFormats		Optional	This is an array that specifies the stream formats (e.g., UTF-8) supported for filenames by the LocalFileSystem configured by this FileSystemSetting element.
FilenameFormats		Optional	This is an array that specifies the formats (e.g. DOS 8.3 names) supported for filenames by the LocalFileSystem configured by this FileSystemSetting element.
FilenameLengthMax		Optional	This specifies the maximum length of a filename that will be supported by the FileSystem configured by this FileSystemSetting element.

Table 127 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Attached to File-System)

Properties	Flags	Requirement	Description & Notes
FilenameReservedCharacterSet		Optional	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames that will be required by the FileSystem configured by this FileSystemSetting element.
SupportedLockingSemantics		Optional	This array specifies the set of file access/locking semantics supported by the FileSystem configured by this FileSystemSetting element.
SupportedAuthorizationProtocols		Optional	This array specifies the kind of file authorization protocols supported by the FileSystem configured by this FileSystemSetting element.
SupportedAuthenticationProtocols		Optional	This array specifies the set of file authentication protocols that can be supported by the FileSystem configured by this FileSystemSetting element.

9.8.21 SNIA_FileSystemSetting (Predefined FS Settings)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 128 describes class SNIA_FileSystemSetting (Predefined FS Settings).

Table 128 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Predefined FS Settings)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for this FileSystemSetting element.
ElementName		Mandatory	A provider supplied user-friendly name for this FileSystemSetting element.
ActualFileSystemType		Mandatory	This identifies the type of filesystem that this FileSystemSetting represents. It shall match the corresponding property of FileSystemCapabilities.
DataExtentsSharing		Optional	This allows the creation of data blocks (or storage extents) that are shared between files.
CopyTarget		Optional	This specifies if support should be provided for using the created filesystem as a target of a Copy operation.
FilenameCaseAttributes		Mandatory	This specifies the support provided for using upper and lower case characters in a filename.

Table 128 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Predefined FS Settings)

Properties	Flags	Requirement	Description & Notes
ObjectTypes		Mandatory	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
NumberOfObjectsMin		Optional	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[] that will be supportable by a LocalFileSystem configured by this FileSystemSetting element.
NumberOfObjectsMax		Optional	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[] that can be supported by a LocalFileSystem configured by this FileSystemSetting element.
NumberOfObjects		Optional	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		Optional	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		Optional	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[] that will be supportable by a LocalFileSystem configured by this FileSystemSetting element.
ObjectSizeMax		Optional	This is an array that specifies the maximum size of an object of the type specified by the corresponding entry in ObjectTypes[] that can be supported by a LocalFileSystem configured by this FileSystemSetting element.
FilenameStreamFormats		Optional	This is an array that specifies the stream formats (e.g., UTF-8) supported for filenames by a filesystem with this setting.
FilenameFormats		Optional	This is an array that specifies the formats (e.g. DOS 8.3 names) supported for filenames by a filesystem with this setting.
FilenameLengthMax		Optional	This specifies the maximum length of a filename supported by a filesystem with this setting.
FilenameReservedCharacterSet		Optional	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames that will be required by a filesystem with this setting.
SupportedLockingSemantics		Optional	This array specifies the set of file access/locking semantics supported by a filesystem with this setting.

Table 128 - SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Predefined FS Settings)

Properties	Flags	Requirement	Description & Notes
SupportedAuthorizationProtocols		Optional	This array specifies the kind of file authorization protocols supported by a filesystem with this setting.
SupportedAuthenticationProtocols		Optional	This array specifies the kind of file authentication protocols supported by a filesystem with this setting.

9.8.22 SNIA_LocalAccessAvailable

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..

Table 129 describes class SNIA_LocalAccessAvailable.

Table 129 - SMI Referenced Properties/Methods for SNIA_LocalAccessAvailable

Properties	Flags	Requirement	Description & Notes
LocalAccessPoint		Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..The name used by the file server to identify the file system. Sometimes referred to as a mount-point. For many UNIX-based systems, this will be a qualified full pathname. For Windows systems this could also be the drive letter used for the LogicalDisk that the filesystem is resident on.
FileSystem		Mandatory	The LocalFileSystem that is being made available to the file server ComputerSystem.
FileServer		Mandatory	The file server ComputerSystem that will be able to export shares from this LocalFileSystem.

9.8.23 SNIA_LocalFileSystem

The following properties of LocalFileSystem are defined by the MOF, but the way we model LocalFileSystem has changed significantly. The setting/configuration properties are not supported using these properties, and so all of these are "Not Supported". The run-time properties will be supported by a statistics/performance profile and that has yet to be defined.

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Requirement: Mandatory

Table 130 describes class SNIA_LocalFileSystem.

Table 130 - SMI Referenced Properties/Methods for SNIA_LocalFileSystem

Properties	Flags	Requirement	Description & Notes
LocalAccessDefinitionRequired		Mandatory	This boolean property indicates whether or not a LocalFileSystem with this FileSystemSetting must be made locally accessible ("mounted") from a file server ComputerSystem before it can be shared or otherwise made available to operational clients.
PathNameSeparatorString		Mandatory	This indicates the string of characters used to separate directory components of a canonically formatted path to a file from the root of the filesystem. This string is expected to be specific to the ActualFileSystemType and so is vendor/ implementation dependent. However, by surfacing it we make it possible for a client to parse a pathname into the hierarchical sequence of directories that compose it.
DirectoryServiceUsage		Optional	This enumeration indicates whether the filesystem supports security principal information and therefore requires support from a file server that uses one or more directory services. If the filesystem requires such support, there must be a concrete subclass of Dependency between the LocalFileSystem element and the specified file server ComputerSystem. The values supported by this property are: 'Not Used' indicates that the filesystem will not support security principal information and so will not require support from a directory service. 'Optional' indicates that the filesystem may support security principal information. If it does, it will require support from a directory service and the Dependency association described above must exist. 'Required' indicates that the filesystem supports security principal information and will require support from a directory service. The Dependency association described above must exist.
CSCreationClassName		Mandatory	The CIM class name of the hosting ComputerSystem.
CSName		Mandatory	The Name property of the hosting ComputerSystem.
CreationClassName		Mandatory	The CIM class name of the this element.
Name		Mandatory	A unique name for this LocalFileSystem in the context of the hosting ComputerSystem.
EnabledState		Optional	Current state of enablement of the LocalFileSystem.
OtherEnabledState		Optional	Vendor-specific state of the LocalFileSystem indicated by EnabledState = 1("Other").

Table 130 - SMI Referenced Properties/Methods for SNIA_LocalFileSystem

Properties	Flags	Requirement	Description & Notes
TimeOfLastStateChange		Optional	A timestamp indicating when the state was last changed.
RequestedState		Optional	Not supported.
OperationalStatus		Mandatory	The current operational status of the LocalFileSystem.
Root		Optional	A path that specifies the "mount point" of the filesystem in an unitary computer system that is both the host of the file system and is the file server that makes it available.
BlockSize		Mandatory	The size of a block in bytes that the implementation used as a fixed block size when creating this filesystem.
FileSystemSize		Mandatory	The total current size of the filesystem in blocks.
AvailableSpace		Mandatory	The space available currently in the filesystem in blocks.
ReadOnly		Optional	Indicates that this is a read-only filesystem that does not allow modifications.
EncryptionMethod		Optional	Indicates if files are encrypted and the method of encryption.
CompressionMethod		Optional	Indicates if files are compressed before being stored, and the methods of compression..
CaseSensitive		Optional	Whether this filesystem is sensitive to the case of characters in filenames.
CasePreserved		Optional	Whether this filesystem preserves the case of characters in filenames when saving and restoring.
CodeSet		Optional	The codeset used in filenames.
MaxFileNameLength		Optional	The length of the longest filename supported by the implementation.
ClusterSize		Optional	Not supported.
FileSystemType		Optional	This is a string that matches FileSystemSetting.ActualFileSystemType property used to create the filesystem. Pragmatically, this property should be ignored.
NumberOfFiles		Optional	The actual current number of files in the filesystem. NOTE: This value is an approximation as it can vary continuously when the filesystem is in use.
IsFixedSize		Optional	Indicates that the filesystem cannot be expanded or shrunk.
ResizeIncrement		Optional	The size by which to increase the size of the filesystem when requested.
RequestStateChange()		Optional	Not supported.

9.8.24 SNIA_LocallyAccessibleFileSystemCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either 'Local Access Required, Defaulted' or 'Local Access Required, Not Defaulted'.

Table 131 describes class SNIA_LocallyAccessibleFileSystemCapabilities.

Table 131 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the SNIA_LocallyAccessibleFileSystemCapabilities associated to a Filesystem Configuration Service.
ElementName		Mandatory	A user-friendly name for this SNIA_LocallyAccessibleFileSystemCapabilities element.

Table 131 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedProperties		Mandatory	An array of property names of the LocallyAccessibleFileSystemSetting that this SNIA_LocallyAccessibleFileSystemCapabilities element supports. 2 'FailurePolicy' 3 'RetriesMax' 4 'InitialEnabledState' 5 'RequestRetryPolicy' 6 'TransmissionRetriesMax' 7 'RetransmissionTimeout' 8 'CachingOptions' 9 'ReadBufferSize' 10 'WriteBufferSize' 11 'AttributeCaching' 12 'ReadWritePolicy' 13 'LockPolicy' 14 'EnableOnSystemStart' 15 'ReadWritePref' 16 'ExecutePref' 17 'RootAccessPref'

Table 131 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedObjectsFor AttributeCaching		Optional	If AttributeCaching is supported, this specifies the array of objects that can be set up for caching. A subset of these entries will become the entries of the AttributeCachingObjects property in the Setting. These classes represent types of objects stored in a filesystem implementation -- files and directories as well as others that may be defined in the future. The corresponding Setting properties, AttributeCaching, AttributeCachingTimeMin, and AttributeCachingTimeMax provide the supported features for the type of object. 'None' and 'All' cannot both be specified; if either one is specified, it must be the first entry in the array and the entry is interpreted as the default setting for all objects. If neither 'None' or 'All' are specified, the caching settings for other objects are defaulted by the implementation. If 'Rest' is specified, the entry applies to all known object types other than the named ones. If 'Unknown' is specified it applies to object types not known to this application (this can happen when foreign file systems are mounted. 0 'Unknown' 1 'None' 2 'All' 3 'Rest' 4 'File' 5 'Directory'

9.8.25 SNIA_LocallyAccessibleFileSystemSetting

Created By: Extrinsic: CreateFileSystem
Modified By: Extrinsic: ModifyFileSystem
Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem
Requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true..

Table 132 describes class SNIA_LocallyAccessibleFileSystemSetting.

Table 132 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for a LocallyAccessibleFileSystemSetting.
ElementName		Mandatory	A user-friendly name for this LocallyAccessibleFileSystemSetting element.
InitialEnabledState		Optional	InitialEnabledState is an integer enumeration that indicates the enabled/disabled states initially set for a locally accessible file system (LAFS). The element functions by passing commands onto the underlying filesystem, and so cannot indicate transitions between requested states because those states cannot be requested. The following text briefly summarizes the various enabled/disabled initial states: 'Enabled' (2) indicates that the element will execute commands, will process any queued commands, and will queue new requests. 'Disabled' (3) indicates that the element will not execute commands and will drop any new requests. 'In Test' (7) indicates that the element will be in a test state. 'Deferred' (8) indicates that the element will not process any commands but will queue new requests. 'Quiesce' (9) indicates that the element is enabled but in a restricted mode. The element's behavior is similar to the Enabled state, but it only processes a restricted set of commands. All other requests are queued.
OtherEnabledState		Optional	A string describing the element's initial enabled/disabled state when the InitialEnabledState property is set to 1 ("Other"). This property MUST be set to NULL when InitialEnabledState is any value other than 1.
FailurePolicy		Optional	An enumerated value that specifies if the operation to make a FileSystem locally accessible to a scoping ComputerSystem should be attempted one or more times in the foreground or tried repeatedly in the background until it succeeds. The number of attempts would be limited by the corresponding RetriesMax property of the setting.
RetriesMax		Optional	An integer specifying the maximum number of attempts that should be made by the scoping ComputerSystem to make a Filesystem locally accessible. A value of "0" specifies an implementation-specific default.

Table 132 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
RequestRetryPolicy		Optional	An enumerated value representing the policy that is supported by the operational file server on a request to the operational file system that either failed or left the file server hanging. If the request is being performed in the foreground, the options are to try once and fail if a timeout happens, or, to try repeatedly. If the request can be performed in the background, the request will be tried repeatedly until stopped.
TransmissionRetries Max		Optional	An integer specifying the maximum number of retransmission attempts to be made from the operational file server to the operational file system when the transmission of a request fails or makes the file server hang. A value of "0" specifies an implementation-specific default. This is only relevant if there is a transmission channel between the file server and the underlying file system.
RetransmissionTime outMin		Optional	An integer specifying the minimum number of milliseconds that the operational file server must wait before assuming that a request to the operational file system has failed. "0" indicates an implementation-specific default. This is only relevant if there is a transmission channel between the operational file server and the operational file system.
CachingOptions		Optional	An enumerated value that specifies if a local cache is supported by the operational file server when accessing the underlying operational file system.
BuffersSupport		Optional	An array or enumerated values that specifies the buffering mechanisms supported by the operational file server for accessing the underlying operational file system." If supported, other properties will establish the level of support. If the property is NULL or the empty array, buffering is not supported.
ReadBufferSizeMin		Optional	An integer specifying the minimum number of bytes that must be allocated to each buffer used for reading. A value of "0" specifies an implementation-specific default.
ReadBufferSizeMax		Optional	An integer specifying the maximum number of bytes that may be allocated to each buffer used for reading. A value of "0" specifies an implementation-specific default.
WriteBufferSizeMin		Optional	An integer specifying the minimum number of bytes that must be allocated to each buffer used for writing. A value of "0" specifies an implementation-specific default.
WriteBufferSizeMax		Optional	An integer specifying the maximum number of bytes that may be allocated to each buffer used for writing. A value of "0" specifies an implementation-specific default.

Table 132 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
AttributeCaching		Optional	An array of enumerated values that specify whether attribute caching is (or is not) supported by the operational file server when accessing specific types of objects from the underlying operational file system. The object type and the support parameters are specified in the corresponding AttributeCachingObjects, AttributeCachingTimeMin, and AttributeCachingTimeMax array properties. Filesystem object types that can be accessed locally are represented by an entry in these arrays. The entry in the AttributeCaching array can be "On", "Off", or "Unknown". Implementation of this feature requires support from other system components, so it is quite possible that specifying "On" may still not result in caching behavior. "Unknown" indicates that the access operation will try to work with whatever options the operational file server and file system can support. In all cases, AttributeCachingTimeMin and AttributeCachingTimeMax provide the minimum and maximum time for which the attributes can be cached. When this Setting is used as a Goal, the client may specify "Unknown", but the Setting in the created object should contain the supported setting, whether "On" or "Off".
AttributeCachingObjects		Optional	An array of enumerated values that specify the attribute caching support provided to various object types by the operational file server when accessing the underlying operational file system. These", types represent the types of objects stored in a FileSystem -- files and directories as well as others that may be defined in the future. The corresponding properties, AttributeCaching, AttributeCachingTimeMin, and AttributeCachingTimeMax provide the supported features for the type of object. "None" and "All" cannot both be specified; if either one is specified, it must be the first entry in the array and the entry is interpreted as the default setting for all objects. If neither "None" or "All" are specified, the caching settings for other objects are defaulted by the implementation. If "Rest" is specified, the entry applies to all known object types other than the named ones. If "Unknown" is specified it applies to object types not known to this application (this can happen when foreign file systems are mounted.
AttributeCachingTimeMin		Optional	An array of integers specifying, in milliseconds, the minimum time for which an object of the type specified by the corresponding AttributeCaching property must be retained in the attribute cache. When used as a Goal, a value of "0" indicates an implementation-specific default.
AttributeCachingTimeMax		Optional	An array of integers specifying, in milliseconds, the maximum time for which an object of the type specified by the corresponding AttributeCaching property must be retained in the attribute cache. When used as a Goal, a value of "0" indicates an implementation-specific default.

Table 132 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
ReadWritePolicy		Optional	An enumerated value that specifies the Read-Write policy set on the operational file system and supported by the operational file server when accessing it. 'Read Only' specifies that the access to the operational file system by the operational file server is set up solely for reading. 'Read/Write' specifies that the access to the operational file system by the operational file server is set up for both reading and writing. 'Force Read/Write' specifies that 'Read-Only' has been overridden by a client with write access to the operational file system. This option is intended for use when the associated FileSystem has been made 'Read Only' by default, as might happen if it were created to be the target of a Synchronization or Mirror operation.
LockPolicy		Optional	An enumerated value that specifies the Locking that will be enforced on the operational file system by the operational file server when accessing it. 'Enforce None' does not enforce locks. 'Enforce Write' does not allow writes to locked files. 'Enforce Read/Write' does not allow reads or writes to locked files.
EnableOnSystemStart		Optional	An enumerated value that specifies if local access from the operational file server to the operational file system should be enabled when the file server is started.
ReadWritePref		Optional	An instance of a CIM_Privilege, encoded as a string, that expresses the client's expectations about access to elements contained in the operational file system. The provider is expected to surface this access using the CIM privilege model.

Table 132 - SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting

Properties	Flags	Requirement	Description & Notes
ExecutePref		Optional	An enumerated value that specifies if support should be provided on the operational file server for executing elements contained in the operational file system accessed through this local access point. This may require setting up specialized paging or execution buffers either on the operational file server or on the operational file system side (as appropriate for the implementation). Note that this does not provide any rights to actually execute any element but only specifies support for such execution, if permitted.
RootAccessPref		Optional	An instance of a CIM_Privilege, encoded as a string, that expresses the client's expectations about privileged access by appropriately privileged System Administrative users on the operational file server ("root" or "superuser") to the operational file system and its elements. The provider is expected to surface this access using the CIM privilege model. Support for the privileged access might require setup at both the operational file server as well as the operational file system, so there is no guarantee that the request can be satisfied.

9.8.26 SNIA_SettingsDefineCapabilities (Predefined FS Settings)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 133 describes class SNIA_SettingsDefineCapabilities (Predefined FS Settings).

Table 133 - SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Predefined FS Settings)

Properties	Flags	Requirement	Description & Notes
PropertyPolicy		Mandatory	PropertyPolicy defines whether or not the non-null, non-key properties of the associated FileSystemSetting element are treated independently or as a correlated set.
ValueRole		Mandatory	ValueRole is an array that further specifies the semantics of the non-null, non-key properties of the associated FileSystemSetting element, such as whether they are supported or unsupported, and if supported, whether they are a default and/or an optimal value or an average of some kind.

Table 133 - SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Predefined FS Settings)

Properties	Flags	Requirement	Description & Notes
ValueRange		Mandatory	ValueRange is an array that further specifies the semantics of the non-null, non-key properties of the associated FileSystemSetting element, such as whether they are point properties, or whether they represent maximum or minimum values for the properties. If some properties already have maximums and/or minimums specified by another FileSystemSetting instance, this could specify increments of the property value that are supported.
GroupComponent		Mandatory	A Filesystem Capabilities element that is defined by a collection of Filesystem Settings.
PartComponent		Mandatory	A Filesystem Setting that provides a point or a partial definition for a Filesystem Capabilities element.

9.8.27 SNIA_SettingsDefineCapabilities (Predefined Local Access Settings)

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either 'Local Access Required, Defaulted' or 'Local Access Required, Not Defaulted'.

Table 134 describes class SNIA_SettingsDefineCapabilities (Predefined Local Access Settings).

Table 134 - SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Predefined Local Access Settings)

Properties	Flags	Requirement	Description & Notes
PropertyPolicy		Mandatory	PropertyPolicy defines whether or not the non-null, non-key properties of the associated SNIA_LocallyAccessibleFileSystemSetting instance are treated independently or as a correlated set.
ValueRole		Mandatory	ValueRole is an array that further specifies the semantics of the non-null, non-key properties of the associated SNIA_LocallyAccessibleFileSystemSetting instance, such as whether they are supported or unsupported, and if supported, whether they are a default and/or an optimal value or an average of some kind.

Table 134 - SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Predefined Local Access Settings)

Properties	Flags	Requirement	Description & Notes
ValueRange		Mandatory	ValueRange is an array that further specifies the semantics of the non-null, non-key properties of the associated SNIA_LocallyAccessibleFileSystemSetting instance, such as whether they are point properties, or whether they represent maximum or minimum values for the properties. If some properties already have maximums and/or minimums specified by another SNIA_LocallyAccessibleFileSystemSetting instance, this could specify increments of the property value that are supported.
GroupComponent		Mandatory	A Capabilities element of the filesystem that is defined by a collection of SNIA_LocallyAccessibleFileSystemSetting elements, each being scoped to the File Server ComputerSystem with which it can be used.
PartComponent		Mandatory	A SNIA_LocallyAccessibleFileSystemSetting that provides a point or a partial definition for a SNIA_LocallyAccessibleFileSystemCapabilities element.

EXPERIMENTAL

EXPERIMENTAL

Clause 10: Filesystem Performance Profile

10.1 Synopsis

Profile Name: Filesystem Performance

Version: 1.3.0

Organization: SNIA

CIM Schema Version: 2.15

Table 135 describes the related profiles for Filesystem Performance.

Table 135 - Related Profiles for Filesystem Performance

Profile Name	Organization	Version	Requirement	Description
Filesystem	SNIA	1.3.0	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "102" (Local Filesystem statistics support).
File Export	SNIA	1.3.0	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "103" (Exported File Share statistics support).
Generic Target Ports	SNIA	1.3.0	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "104" (Exporting Port statistics support).

Note: Each of these subprofiles is mandatory if the element in question is to be metered. For example, in order to keep statistics on exported file shares, it will be necessary for File Shares to be modeled through the use of the File Export Subprofile.

Central Class: FileSystemStatisticsService

Scoping Class: ComputerSystem

10.2 Description

10.2.1 Overview

The Filesystem Performance Subprofile defines classes and methods for managing filesystem-related performance information. It is a subprofile for use with autonomous profiles that directly support filesystems, which in this release of SMI-S specifically includes the NAS Head and the Self-Contained NAS profiles.

One of the key SRM disciplines for managing storage is Performance Management. In order to manage performance, a number of processes need to be in place, including the ability to measure the performance and saturation points of components within the storage network.

There are currently no common statistics defined that can be used to manage multiple vendor filesystem-related entities (such as File Servers) from a performance perspective. This subprofile defines specific measurements and methods to make common statistics available to SRM applications regarding filesystem-related entities. Examples of such statistics include:

- The read, write and other I/O operation counts for a filesystem or a file share,
- The cumulative elapsed time required for the I/O operations to complete,
- The number of bytes transferred per unit of time.

Particular areas related to Performance Management that can make use of the statistics provided by the Filesystem Performance Subprofile include:

- Filesystem utilization (e.g., "hot-spot" and trend analyses; tracking usage efficiency by monitoring response times and IOPS/throughput rates; identifying over-utilization and contention that is leading to performance degradation).
- Diagnostics and problem determination (e.g., identifying bottlenecks, "point(s) of pain", etc., especially at an upper level within the overall "I/O operation stack").
- Tuning (e.g., determining allocation/reallocation of particular filesystems and/or file placements in the efforts to meet overall performance goals and/or other Service Level Agreements; determining the impact of the underlying storage and applicable network provisioning upon filesystem performance and utilization).
- Workload characterization (e.g., characterizing particular filesystem usage with possible correlation to associated applications).
- Modeling and planning (e.g., enabling the use of empirical metrics as the input/basis for various modeling and planning exercises related to filesystem and overall storage concerns).

Performance Measurement within the context of filesystems is the key deliverable that is the focus of this subprofile. Of particular importance, the statistics provided by the Filesystem Performance Subprofile can help facilitate a "top-down" approach within the areas noted above (i.e., by reflecting performance information that is directly related to and seen by/at a "top-most" component within the overall I/O operation processing stack).

Note: Performance analysis is broader than simply filesystems and related entities such as File Servers. Complete analysis requires performance information from hosts, fabric and the underlying storage systems. Theses are (or will be) addressed separately as part of the appropriate profiles (e.g., the Block Server Performance Subprofile, which includes further discussion regarding Performance Management).

The Filesystem Performance Subprofile provides statistics, which are associated with fundamental elements that can comprise a filesystem-related entity (such as a NAS Head or a Self-Contained NAS). These elements include:

- Filesystems,
- Exported file shares,

- Network-interface ports used to export file shares.

In order to monitor and manage the aforementioned elements, it is necessary to identify performance counters for each of these elements and to externalize an interface so that SRM applications can retrieve the counter values when they so desire. The function of this subprofile is to support such SRM applications.

The Filesystem Performance Subprofile augments the profiles and subprofiles for those autonomous profiles within this release of SMI-S that directly support filesystems. Instead of being an isolated subprofile, this subprofile adds modeling constructs to existing profiles and subprofiles. Together these enhancements make up the Filesystem Performance Subprofile (as would be registered in the Server Profile as a RegisteredSubprofile).

10.3 Implementation

10.3.1 Performance Additions Overview

Figure 15 provides an overview of the model. The shaded grey boxes show the new classes added by the Filesystem Performance Subprofile.

Note: Not all properties defined for the statistics classes are shown within Figure 15. That is, there are additional properties (both mandatory and optional) that are included within the statistical classes. These properties can be found in .

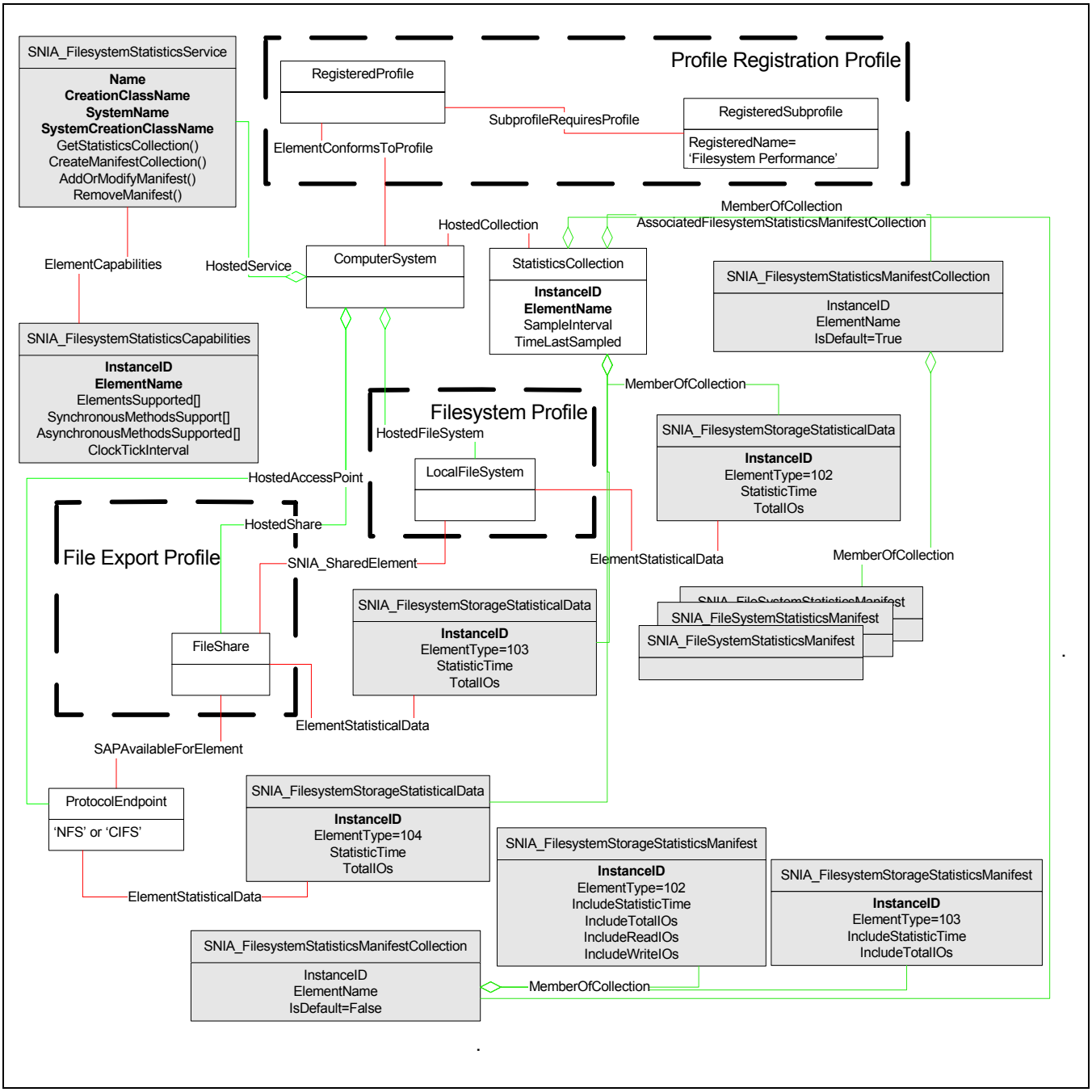

Figure 15 - Filesystem Performance Subprofile Summary Instance Diagram

Figure 15 shows a single instance of StatisticsCollection for the entire profile. The ComputerSystem (i.e., the "top level" computer system depicted within the figure) is that of the autonomous profile (e.g., a NAS Head or a Self-Contained NAS) which utilizes the Filesystem Performance Subprofile.

The StatisticsCollection is the anchor point from which all statistics being kept by the profile can be found. Statistics are defined as a FileSystemStatisticalData class, instances of which hold the statistics for particular metered elements (e.g., filesystems and file shares). The particular type of metered element is recorded in the instance of FileSystemStatisticalData within the ElementType property.

All of the statistics instances are related to the elements that they meter via the ElementStatisticalData association (e.g., FileSystemStatisticalData for a File Share can be found from the File Share by traversing the ElementStatisticalData association).

All of the statistics instances kept within the profile are associated to the one StatisticsCollection instance. Access to all of the statistics for the profile is through the StatisticsCollection. The StatisticsCollection has a HostedCollection association to the "top level" computer system of the profile.

Note that statistics may be kept for a number of elements within the profile, including elements within subprofiles. The particular elements that are metered are:

- **Local Filesystem**. This provides a summary of all statistics for a particular filesystem (i.e., an instance of LocalFileSystem). For example, all file read I/O operations (ReadIOs) directed to a particular filesystem. These statistics are kept within the FileSystemStatisticalData instances, with one for each filesystem within the system.
- **Exported File Share**. This provides a summary of all statistics for a particular file share that is exported (i.e., an instance of FileShare as described within the File Export Profile). For example, all file read I/O operations (ReadIOs) directed to a particular file share that is exported to the network. These statistics are kept within the FileSystemStatisticalData instances, with one for each FileShare within the system.
- **Exporting Port**. This provides a summary of all statistics for a particular port through which a file share being exported can be accessed (i.e., an instance of ProtocolEndpoint through which a FileShare can be accessed as described within the File Export Profile). For example, all file read I/O operations (ReadIOs) directed to a particular file share exporting port. These statistics are kept within the FileSystemStatisticalData instances, with one for each file share exporting port within the system.

Finally, Figure 15 illustrates the FileSystemStatisticsService for Bulk retrieval of all the statistics data and the creation of manifest collections. These methods (which are provided in a manner akin to that provided by the Block Server Performance Subprofile) will be discussed later. They are shown here for completeness. Associated with the FileSystemStatisticsService is a FileSystemStatisticsCapabilities instance that identifies the specific capabilities implemented by the filesystem performance statistics support. Specifically, it includes an "ElementsSupported" property that identifies the elements for which statistics are kept; the FileSystemStatisticsCapabilities instance also identifies the various retrieval mechanisms (e.g., Extrinsic, Association Traversal, Indications and/or Query) that are implemented (i.e., supported) by the filesystem statistics support.

10.3.2 Summary of FileSystemStatisticsData support by Profile

Table 2 defines the Element Types (for FileSystemStatisticsData instances) that may be supported by profile.

Table 136 - Summary of Element Types by Profile

ElementType	NAS Head	Self-Contained NAS
Local Filesystem	YES	YES
Exported File Share	YES	YES
Exporting Port	YES	YES

YES means that this specification defines the element type for the profile, but actual support by any given implementation would be implementation dependent. NO means that this specification does not specify this element type for the profile.

10.3.3 Profile Registration Profile Support for the Filesystem Performance Subprofile

At the top of Figure 15 there is a dashed box that illustrates a part of the Profile Registration Profile for the autonomous profile (e.g., a NAS Head or a Self-Contained NAS) that utilizes the Filesystem Performance Subprofile. The part illustrated represents the particulars for the Filesystem Performance Subprofile. If performance support has been implemented, then there shall be a RegisteredSubprofile instance for the Filesystem Performance Subprofile.

10.3.4 Default Manifest Collection

Associated with the instances of the StatisticsCollection shall be a provider-supplied (Default) SNIA_FileSystemManifestCollection that represents the statistics properties that are kept by the profile. The default manifest collection is indicated by the IsDefault property (=True) of the SNIA_FileSystemManifestCollection. For each metered object (element) of the profile implementation, the default manifest collection will have exactly one manifest that will identify which properties are included for that metered object. If an object is not metered, then there shall not be a manifest for that element type. If an element type (e.g., Local filesystem) is metered, then there shall be a manifest for that element type.

10.3.5 Client Defined Manifest Collection

Manifest collections are either provider-supplied (SNIA_FileSystemManifestCollection.IsDefault=True) for the profile implementation or client-defined collections (SNIA_FileSystemManifestCollection.IsDefault=False). Client-defined collections are used to indicate the specific statistics properties that the client would like to retrieve using the GetStatisticsCollection method. For a discussion of provider-supplied manifest collections, see 10.3.4.

Client-defined manifest collections are a mechanism for restricting the amount of data returned on a GetStatisticsCollection request. A client-defined manifest collection is identified by the IsDefault property of the collection set to False. For each element type of the filesystem statistics class (e.g., Local Filesystem, Exported File Share, etc.), a manifest can be defined that identifies which specific properties of the particular statistics class element type are to be returned on a GetStatisticsCollection request. Each of the element types of the filesystem statistics class may have no or one manifest in any given manifest collection. This is illustrated in Figure 15.

In Figure 15, manifest classes are defined for filesystems (LocalFileSystem) and exported file shares (FileShare). Each property of the manifest is a Boolean that indicates whether the property is to be returned (true) or omitted (false).

Multiple client-defined manifest collections can be defined in the profile. Consequently, different clients or different client applications can define different manifests for different application needs. A manifest collection can completely omit a whole set of statistics pertaining to a particular element type; for example, no ProtocolEndPoint statistics (i.e., filesystem performance statistics associated with the element type of "Exporting Port", which represents a port through which a File Share can be accessed from the network) are included within the client-defined manifest collection shown in Figure 15. Since manifest collections are "client objects", they are named (ElementName) by the client for the client's convenience. The CIM server will generate an instance ID to uniquely identify the manifest collection in the CIM Server.

Client-defined manifest collections are created using the CreateManifestCollection method. Manifests are added or modified using the AddOrModifyManifest method. A manifest may be removed from the manifest collection by using the RemoveManifests method.

Note: Use of manifest collections is optional with the GetStatisticsCollection method. If NULL for the manifest collection is passed on input, then all statistics instances are assumed (i.e., all available statistics will be returned).

10.3.6 Capabilities Support for Filesystem Performance Subprofile

There are two dimensions to determining what is supported with a Filesystem Performance Subprofile implementation. First, there are the RegisteredSubprofiles supported by the autonomous profile (e.g., a NAS Head or a Self-Contained NAS Profile) that utilizes the Filesystem Performance Subprofile. In order to support statistics

for a particular class of metered element, the corresponding object shall be modeled. So, if a NAS Head (for example) has not implemented the File Export Subprofile, then it shall not implement the FileSystemStatisticalData for "Exported File Share" in the Filesystem Performance Subprofile (and implementation of the File Export Subprofile does not guarantee implementation of the FileSystemStatisticalData for exported file shares).

Both of these dimensions are captured in the FileSystemStatisticsCapabilities class instance. This class instance is not created nor modified by Clients; rather, it is populated by the provider and has three properties of interest (as discussed within the following sections). The second dimension is techniques supported for retrieving statistics and manipulating manifest collections.

For the methods-supported properties described below (namely, SynchronousMethodsSupported and AsynchronousMethodsSupported), any or all of the respective values can be missing (e.g., the arrays can be NULL). If all of the methods supported are NULL, then manifest collections are not supported and neither GetStatisticsCollection nor Query are supported for the retrieval of statistics. This leaves enumerations or association traversals as the only methods for retrieving the statistics.

10.3.6.1 ElementsSupported

This property within the FileSystemStatisticsCapabilities class defines a list of element types for which statistical data is available. For this release of SMI-S, the values of interest are "Local Filesystem", "Exported File Share", and "Exporting Port".

To be a valid implementation of the Filesystem Performance Subprofile, at least one of the values listed for ElementsSupported shall be supported. ElementsSupported is an array, such that all of the values can be identified.

10.3.6.2 SynchronousMethodsSupported

This property within the FileSystemStatisticsCapabilities class defines the synchronous mechanisms that are supported for retrieving statistics and for defining and modifying filters for statistics retrieval. For this release of SMI-S, the values of interest are "Exec Query", "Indications", "Query Collection", "GetStatisticsCollection", "Manifest Creation", "Manifest Modification", and "Manifest Removal".

10.3.6.3 AsynchronousMethodsSupported

This property within the FileSystemStatisticsCapabilities class defines the asynchronous mechanisms that are supported for retrieving statistics. For this release of SMI-S, this should be NULL.

10.3.6.4 ClockTickInterval

An internal clocking interval for all timer counters kept in the system implementation, measured in microseconds (i.e., the unit of measure in the timers, measured in microseconds). Time counters are considered to be monotonically increasing counters that contain "ticks". Each tick represents one clock tick interval.

For example, if ClockTickInterval contained a value of 32, then each time counter tick would represent 32 microseconds.

10.3.7 Health and Fault Management Consideration

Not defined in this version of the specification.

10.3.8 Cascading Considerations

Not applicable

10.4 Methods of the Profile

10.4.1 Extrinsic Methods of the Profile

10.4.1.1 Overview

The methods supported by this subprofile are summarized in Table 3 and detailed within the sections that follow it.

Table 137 - Creation, Deletion and Modification Methods in the Filesystem Performance Subprofile

Method	Created Instances	Deleted Instances	Modified Instances
GetStatisticsCollection	None	None	None
CreateManifestCollection	FileSystemStatisticsManifestCollection AssociatedFileSystemStatisticsManifestCollection	None	None
AddOrModifyManifest	FileSystemStatisticsManifest(subclass) MemberOfCollection	None	FileSystemStatisticsManifest(subclass)
RemoveManifest	None	FileSystemStatisticsManifest(subclass) MemberOfCollection	None

10.4.1.2 GetStatisticsCollection

This extrinsic method retrieves statistics in a well-defined bulk format. The set of statistics returned by this method is determined by the list of element types passed into the method and the manifests for those types contained in the supplied manifest collection. The statistics are returned through a well-defined array of strings that can be parsed to retrieve the desired statistics as well as limited information about the elements that those metrics describe.

```
GetStatisticsCollection(
        [IN (false), OUT, Description(Reference to the job(shall be null in this
                        version of SMI-S.)]
CIM_ConcreteJob  REF  Job,
[IN, Description(Element types for which statistics should be returned)
ValueMap { "1", "102", "103", "104", "..", "0x8000.." },
Values { "Other", "Local Filesystem", "Exported File Share", "Exporting Port",
                        "DMTF Reserved", "Vendor Specific" }]
uint16  ElementTypes[],
[IN, Description ( "An array of strings that specify the particular "Other"
                        element(s) when the ElementType property above includes
                        the ElementType value of 1 (i.e., "Other").  Each
                        string within this array identifies a separate "Other"
                        element and duplicate string values are NOT allowed.
                        This property should be set to NULL when the
                        ElementType property does not  include the value of
                        1.")]
      string OtherElementTypeDescriptions[],
```

```
[IN, Description(The manifest collection that contains the manifests which list
                          the metrics that should be returned for each element
                          type)]
SNIA_FileSystemStatisticsManifestCollection  REF  ManifestCollection,
[IN, Description("Specifies the format of the Statistics output parameter")
ValueMap { "2" } ,
Values ( "CSV" )]
uint16 StatisticsFormat,
[OUT, Description(The statistics for all the elements as determined by the
                          Elements and ManifestCollection parameters)]
string Statistics[] );
```

Error returns are:

{ "Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Method Reserved", "Method Parameters Checked - Job Started", "Element Not Supported", "Statistics Format Not Supported", "Method Reserved", "Vendor Specific"}

Note: In this version of the standard, Job Control is not supported for the GetStatisticsCollection method. This method should always return NULL for the Job parameter.

If the ElementTypes[] array is empty, then no data is returned. If the ElementTypes[] array is NULL, then the ElementTypes[] parameter is ignored and all data specified in the manifest collection is returned.

If the manifest collection is empty, then no data is returned. If the manifest collection parameter is NULL, then the default manifest collection is used. (Note: In SMI-S, a default manifest collection shall exist if the GetStatisticalCollection method is supported).

Note: The ElementTypes[] and ManifestCollection parameters may identify different sets of element types. The effect of this will be for the implementation to return statistics for the element types that are in both lists (that is, the intersection of the two lists). This intersection could be empty. In this case, no data will be returned.

For the current version of SMI-S, the only recognized value for StatisticsFormat is "CSV". The method may support other values, but they are not specified by SMI-S (i.e., they would be vendor specific).

Given a client has an inventory of the metered objects with Statistics InstanceIDs that may be used to correlate with the FileSystemStatisticalData instances, a simple CSV format is sufficient and the most efficient human-readable format for transferring bulk statistics. More specifically, the following rules constrain that format and define the content of the String[] Statistics output parameter to the Get Statistics Collection() method:

- The Statistics[] array may contain multiple statistics records per array entry. In such cases, the total length of the concatenated record strings will not exceed 64K bytes. And a single statistics record will not span Array entries.
- There shall be exactly one statistics record per line in the bulk Statistics parameter. A line is terminated by:
  - a line-feed character
  - the end of a String Array Element (i.e., a statistics record cannot overlap elements of the String[] Statistics output parameter).
- Each statistics record shall contain the InstanceID of the FileSystemStatisticalData instance, the value map (number) of the ElementType of the metered object, and one value for each property that the relevant FileSystemStatisticsManifest specifies as "true".

- Each value in a record shall be separated from the next value by a Semi-colon (";"). This is to support internationalization of the CSV format. A provider creating a record in this format should not include white space between values in a record. A client reading a record it has received would ignore white-space between values.
- The InstanceID value is an opaque string that shall correspond to the InstanceID property from FileSystemStatisticalData instance.
  - For the convenience of client software that needs to be able to correlate InstanceIDs between different GetStatisticsCollection method invocations, the InstanceID for FileSystemStatisticalData instance shall be unique across all instances of the FileSystemStatisticalData class. It is not sufficient that InstanceID is unique across subclasses of FileSystemStatisticalData.
- The ElementType value shall be a decimal string representation of the Element Type number (e.g., "102" for Local Filesystem). The StatisticTime shall be a string representation of DateTime. All other values shall be decimal string representations of their statistical values.
- Null values shall be included in records for which a statistic is returned (specified by the manifest or by a lack of manifest for a particular element type) but there is no meaningful value available for the statistic. A NULL statistic is represented by placing a semi-colon (;) in the record without a value at the position where the value would have otherwise been included. A record in which the last statistic has a NULL value shall end in a semi-colon (;).
- The first three values in a record shall be the InstanceID, ElementType and StatisticTime values from the FileSystemStatisticalData instance. The remaining values shall be returned in the order in which they are defined by the MOF for the FileSystemStatisticsManifest class or subclass the record describes.

As an additional convention, a provider should return all the records for a particular element type in consecutive String elements, and the order of the element types should be the same as the order in which the element types were specified in the input parameter to GetStatisticsCollection().

Example output as it might be transmitted in CIM-XML. It shows records for 5 local filesystems and 5 exported file shares, assuming that 6 statistics were specified in the FileSystemStatisticsManifest instance for both local filesystems and exported file shares. The sixth statistic is unavailable for local filesystems, and the fourth statistic is unavailable for exported file shares:

```
<METHODRESPONSE NAME="GetStatisticsCollection">
<RETURNVALUE PARAMTYPE="uint32">
<VALUE>
0
</VALUE>
</RETURNVALUE>
<PARAMVALUE NAME="Statistics" PARAMTYPE="string">
<VALUE.ARRAY>
<VALUE>
LOCALFILESYSTEMSTATS1;102;20060811133015.0000010-
                          300;11111;22222;33333;44444;55555;
LOCALFILESYSTEMSTATS2;102;20060811133015.0000020-
                          300;11111;22222;33333;44444;55555;
LOCALFILESYSTEMSTATS3;102;20060811133015.0000030-
                          300;11111;22222;33333;44444;55555;
LOCALFILESYSTEMSTATS4;102;20060811133015.0000040-
                          300;11111;22222;33333;44444;55555;
LOCALFILESYSTEMSTATS5;102;20060811133015.0000050-
                          300;11111;22222;33333;44444;55555;
```

```
</VALUE>
<VALUE>
EXPORTFILESHARESTATS1;103;20060811133015.0000100-
                          300;11111;22222;33333;;55555;66666
EXPORTFILESHARESTATS2;103;20060811133015.0000110-
                          300;11111;22222;33333;;55555;66666
EXPORTFILESHARESTATS3;103;20060811133015.0000120-
                          300;11111;22222;33333;;55555;66666
EXPORTFILESHARESTATS4;103;20060811133015.0000130-
                          300;11111;22222;33333;;55555;66666
EXPORTFILESHARESTATS5;103;20060811133015.0000140-
                          300;11111;22222;33333;;55555;66666
</VALUE>
</VALUE.ARRAY>
</PARAMVALUE>
</METHODRESPONSE>
```

10.4.1.3 CreateManifestCollection

This extrinsic method creates a new manifest collection whose members serve as a filter for metrics retrieved through the GetStatisticsCollection method.

```
CreateManifestCollection(
[IN, Description(The collection of statistics that will be filtered using the new
                          manifest collection)]
CIM_StatisticsCollection  REF  Statistics,
[IN, Description(Client-defined name for the new manifest collection)
string  ElementName,
[OUT, Description(Reference to the new manifest collection)]
SNIA_FileSystemManifestCollection  REF  ManifestCollection );
```

Error returns are:

```
{  "Ok", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter",
                          "Method Reserved", "Vendor Specific" }
```

10.4.1.4 AddOrModifyManifest

This is an extrinsic method that either creates or modifies a statistics manifest for this statistics service. A client supplies a manifest collection within which the new manifest collection will be placed or an existing manifest will be modified, the element type of the statistics that the manifest will filter, and a list of statistics that should be returned for that element type using the GetStatisticsCollection method.

```
AddOrModifyManifest(
[IN, Description(Manifest collection that the manifest is or should be a member
                          of)]
SNIA_FileSystemStatisticsManifestCollection  REF  ManifestCollection,
[IN, Description(The element type whose statistics the manifest will filter)
ValueMap { "1", "102", "103", "104", "..", "0x8000.." },
Values { "Other", "Local Filesystem", "Exported File Share", "Exporting Port",
                          "DMTF Reserved", "Vendor Specific" }]
uint16  ElementType,
```

```
 [IN, Description ( "A string describing the type of element when the ElementType
                          property above is set to 1 (i.e., "Other").  This
                          property should be set to NULL when the ElementType
                          property is any value other than 1.")]
      string OtherElementTypeDescription,

[IN, Description(The client-defined string that identifies the manifest created or
                          modified by this method)
string  ElementName,
[IN, Description(The statistics that will be included by the manifest filter; that
                          is, the statistics that will be supplied through the
                          GetStatisticsCollection method)
string  StatisticsList[],
 [OUT, Description(The Manifest that is created or modified on the successful
                          execution of this method)]
SNIA_FileSystemManifest  REF  Manifest );
```

Error returns are:

```
{ "Success", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter",
                          "Method Reserved", "Element Not Supported", "Metric not
                          supported", "ElementType Parameter Missing", "Method
                          Reserved", "Vendor Specific" }
```

If the StatisticsList[] array is empty, then only InstanceID and ElementType will be returned when the manifest is referenced. If the StatisticsList[] array parameter is NULL, then all supported properties is assumed (i.e., all supported properties will be included).

Note: This would be the FileSystemStatisticsManifest from the default manifest collection.

10.4.1.5 RemoveManifests

This is an extrinsic method that removes manifests from the manifest collection.

```
RemoveManifests(
[IN, Description(Manifest collection from which the manifests will be removed)]
SNIA_FileSystemStatisticsManifestCollection  REF  ManifestCollection,
[IN, Description(List of manifests to be removed from the manifest collection)
SNIA_FileSystemStatisticsManifest  REF  Manifest[] );
```

Error returns are:

```
{  "Success", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid
                          Parameter", "Method Reserved", "Manifest not found",
                          "Method Reserved", "Vendor Specific" }
```

10.4.2 Intrinsic Methods of this Profile

Note: Basic Write intrinsic methods are not specified for StatisticsCollection, HostedCollection, FileSystemStatisticalData, MemberOfCollection or ElementStatisticalData.

10.4.2.1 DeleteInstance (of a FileSystemStatisticsManifestCollection)

This will delete the FileSystemStatisticsManifestCollection where IsDefault=False, the AssociatedFileSystemStatisticsManifestCollection association to the StatisticsCollection and all manifests collected

by the manifest collection (and the MemberOfCollection associations to the FileSystemStatisticsManifestCollection).

10.4.2.2 Association Traversal

One of the ways of retrieving statistics is through association traversal from the StatisticsCollection to the individual Statistics following the MemberOfCollection association. This shall be supported by all implementations of the Filesystem Performance Subprofile and would be available to clients if the provider does not support the EXEC QUERY or GetStatisticsCollection approaches.

10.5 Use Cases

10.5.1 Summary of Statistics Support by Element

Not all statistics properties are kept for all elements. Table 4 illustrates the statistics properties that are kept for each of the metered elements.

Table 138 - Summary of Statistics Support by Element

Statistic Property	Local Filesystem	Exported File Share	Exporting Port	Other
StatisticTime	R	R	R	R
TotalIOs	R	R	R	R
TotalBytesTransferred	R	R	R	N
ReadIOs	R	R	N	N
WriteIOs	R	R	N	N
OtherIOs	R	R	N	N
MetadataReadIOs	O	O	N	N
MetadataWriteIOs	O	O	N	N
TotalIOTimeCounter	O	O	O	N
TotalIdleTimeCounter	O	O	O	N
ReadIOTimeCounter	O	O	N	N
BytesRead	O	O	N	N
WriteIOTimeCounter	O	O	N	N
BytesWritten	O	O	N	N
MetadataBytesRead	O	O	N	N
MetadataBytesWritten	O	O	N	N

The legend is:

R - Required

O - Optional

N - Not specified

A complete list of definitions of the metered elements as defined by the ElementType property of FileSystemStatisticalData is below:

- ElementType = 1 (Other) - This is used by the provider to specify a filesystem-related metered element other than one explicitly declared (e.g., "Local Filesystem" below) within the list of element types supported by the Filesystem Performance Subprofile in this release of SMI-S. If the ElementType is "Other", then information describing the metered element should be provided in the "OtherElementTypeDescription" string property.
- ElementType = 102 (Local Filesystem) - This is a filesystem that would be a LocalFileSystem in the Filesystem Profile. It is a target for I/O operations that would include file I/O operations for storing and retrieving the contents of a file maintained by the filesystem, I/O operations directed to directories maintained by the filesystem, and other I/O operations performed to manage the filesystem and its contents.
- ElementType = 103 (Exported File Share) - This is a FileShare in the File Export Subprofile; it is a file share that is exported to a network.
- ElementType = 104 (Exporting Port) - This is a port through which a file share being exported can be accessed. It is a ProtocolEndPoint through which a FileShare can be accessed as described within the File Export Profile.

10.5.2 Formulas and Calculations

Table 4 identifies the set of statistics that are recommended for various elements associated with filesystems. Once collected, these metrics can be further enhanced through the definition of formulas and calculations that create additional "derived" statistics.

TTable 139 defines a set of such derived statistics as pertain to a calculated time interval. These calculated statistics are by no means the only possible derivations but serve as examples of commonly requested statistics.

Table 139 - Formulas and Calculations - Calculated Statistics for a Time Interval

New statistic	Formula
TimeInterval	delta StatisticTime
I/O rate	delta TotalIOs / TimeInterval
I/O average response time	delta TotalIOTimeCounter / delta TotalIOs
Read average response time	delta ReadIOTimeCounter / delta ReadIOs
Write average response time	delta WriteIOTimeCounter / delta WriteIOs
Average Read Size	delta BytesRead / delta ReadIOs
Average Write Size	delta BytesWritten / delta WriteIOs
% Read	100 * (delta ReadIOs / delta TotalIOs)
% Write	100 * (delta WriteIOs / delta TotalIOs)

10.5.3 Filesystem Performance Supported Capabilities Patterns

The Filesystem Performance Subprofile in this release of SMI-S formally recognizes the Capabilities patterns summarized in Table 5.

Table 140 - Filesystem Performance Subprofile Supported Capabilities Patterns

Element Supported	SynchronousMethods Supported	AsynchronousMethods Supported
Any (at least one)	NULL	NULL
Any (at least one)	Neither GetStatisticsCollection nor Exec Query	NULL
Any (at least one)	GetStatisticsCollection	NULL
Any (at least one)	Any	NULL
Any (at least one)	Exec Query	NULL
Any (at least one)	GetStatisticsCollection, Exec Query	NULL
Any (at least one)	"Manifest Creation", "Manifest Modification", and "Manifest Removal”	NULL
Any (at least one)	"Indications", "Query Collection”	NULL

An implementation will support GetStatisticsCollection, Query, GetStatisticsCollection and Query or neither. But if the implementation supports GetStatisticsCollection, it shall support Synchronous execution.

If manifest collections are supported, then ALL three methods shall be supported (creation, modification and removal).

10.5.4 Client Considerations and Recipes

Not defined in this version of the specification.

10.6 CIM Elements

Table 135 describes the CIM elements for Filesystem Performance.

Table 141 - CIM Elements for Filesystem Performance

Element Name	Requirement	Description
10.6.1 CIM_ElementCapabilities	Mandatory	This associates the FileSystemStatisticsCapabilities to the FileSystemStatisticsService.
10.6.2 CIM_ElementStatisticalData (Exported File Share Stats)	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "103" (Exported File Share statistics support). This associates a FileSystemStatisticalData instance to the exported File Share for which the statistics are collected.
10.6.3 CIM_ElementStatisticalData (Exporting Port Stats)	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "104" (Exporting Port statistics support). This associates a FileSystemStatisticalData instance to the exporting Port for which the statistics are collected.
10.6.4 CIM_ElementStatisticalData (Local Filesystem Stats)	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "102" (Local Filesystem statistics support). This associates a FileSystemStatisticalData instance to the local Filesystem for which the statistics are collected.
10.6.5 CIM_ElementStatisticalData (OTHER Element Type Stats)	Conditional	Conditional requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "1" (OTHER element type statistics support). This associates a FileSystemStatisticalData instance to a provider-specified other element for which the statistics are collected.
10.6.6 CIM_HostedCollection (Client Defined)	Conditional	Conditional requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported.This would associate a client defined FileSystemStatisticsManifestCollection to the top level system for the profile (e.g., a NAS Head).

Table 141 - CIM Elements for Filesystem Performance

Element Name	Requirement	Description
10.6.7 CIM_HostedCollection (Default)	Mandatory	This would associate a default FileSystemStatisticsManifestCollection to the top level system for the profile (e.g., a NAS Head).
10.6.8 CIM_HostedCollection (Provider Supplied)	Mandatory	This would associate the StatisticsCollection to the top level system for the profile (e.g., NAS Head).
10.6.9 CIM_HostedService	Mandatory	This associates the FileSystemStatisticsService to the ComputerSystem that hosts it.
10.6.10 CIM_MemberOfCollection (Member of client defined collection)	Conditional	Conditional requirement: Clients can modify manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.This would associate Manifests to client-defined manifest collections.
10.6.11 CIM_MemberOfCollection (Member of predefined collection)	Mandatory	This would associate predefined Manifests to the default manifest collection.
10.6.12 CIM_MemberOfCollection (Member of statistics collection)	Mandatory	This would associate all filesystem statistics instances to the StatisticsCollection.
10.6.13 CIM_StatisticsCollection	Mandatory	This would be a collection point for all filesystem statistics that are kept for metered elements of a system that provides filesystem support (such as a NAS Head or a Self-Contained NAS).
10.6.14 SNIA_AssociatedFileSystemStatisticsManifestCollection (Client defined collection)	Conditional	Conditional requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.This is an association between the StatisticsCollection and a client defined manifest collection.
10.6.15 SNIA_AssociatedFileSystemStatisticsManifestCollection (Provider defined collection)	Mandatory	This is an association between the StatisticsCollection and a provider supplied (predefined) manifest collection that defines the filesystem statistics properties supported by the profile implementation.
10.6.16 SNIA_FileSystemStatisticalData	Mandatory	The SNIA_FileSystemStatisticalData class defines the filesystem statistics properties that may be kept for a metered element of a system that provides filesystem support (such as a NAS Head or a Self-Contained NAS). Examples of such metered elements include LocalFileSystem (Local Filesystem) and FileShare (Exported File Share).

Table 141 - CIM Elements for Filesystem Performance

Element Name	Requirement	Description
10.6.17 SNIA_FileSystemStatisticsCapabilities	Mandatory	This defines the statistics capabilities supported by the implementation of the profile.
10.6.18 SNIA_FileSystemStatisticsManifest (Client Defined)	Conditional	Conditional requirement: Clients can modify manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.An instance of this class defines the filesystem statistics properties of interest to the client for one element type.
10.6.19 SNIA_FileSystemStatisticsManifest (Provider Support)	Mandatory	An instance of this class defines the filesystem statistics properties supported by the profile implementation for one element type.
10.6.20 SNIA_FileSystemStatisticsManifestCollection (Client Defined)	Conditional	Conditional requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.An instance of this class defines one client defined collection of filesystem statistics manifests (one manifest for each element type).
10.6.21 SNIA_FileSystemStatisticsManifestCollection (Provider Defined)	Mandatory	An instance of this class defines the predefined collection of default filesystem statistics manifests (one manifest for each element type).
10.6.22 SNIA_FileSystemStatisticsService	Mandatory	This is a Service that provides (optional) services of bulk statistics retrieval and manifest set manipulation methods.

10.6.1 CIM_ElementCapabilities

CIM_ElementCapabilities represents the association between ManagedElements (i.e., SNIA_FileSystemStatisticsService) and their Capabilities (e.g., SNIA_FileSystemStatisticsCapabilities). Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementCapabilities association for the referenced instance of Capabilities. ElementCapabilities describes the existence requirements and context for the referenced instance of ManagedElement. Specifically, the ManagedElement shall exist and provides the context for the Capabilities.

CIM_ElementCapabilities is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 142 describes class CIM_ElementCapabilities.

Table 142 - SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Requirement	Description & Notes
ManagedElement	Mandatory	The managed element (FileSystemStatisticsService)
Capabilities	Mandatory	The Capabilities instance associated with the FileSystemStatisticsService.

10.6.2 CIM_ElementStatisticalData (Exported File Share Stats)

CIM_ElementStatisticalData is an association that relates an exported File Share to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of FileSystemStatistics. ElementStatisticalData describes the existence requirements and context for the FileSystemStatistics, relative to a specific File Share that is being exported.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "103" (Exported File Share statistics support).

Table 143 describes class CIM_ElementStatisticalData (Exported File Share Stats).

Table 143 - SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Exported File Share Stats)

Properties	Requirement	Description & Notes
ManagedElement	Mandatory	A reference to an exported FileShare for which the Statistics apply.
Stats	Mandatory	A reference to the FileSystemStatisticalData that hold the statistics for the exported FileShare.

10.6.3 CIM_ElementStatisticalData (Exporting Port Stats)

CIM_ElementStatisticalData is an association that relates an exporting Port to its statistics. This exporting Port is a ProtoEndPoint through which a file share that is being exported can be accessed. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of FileSystemStatistics. ElementStatisticalData describes the existence requirements and context for the FileSystemStatistics, relative to a specific exporting Port.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "104" (Exporting Port statistics support).

Table 144 describes class CIM_ElementStatisticalData (Exporting Port Stats).

Table 144 - SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Exporting Port Stats)

Properties	Requirement	Description & Notes
ManagedElement	Mandatory	A reference to a ProtocolEndPoint port for which the Statistics apply.
Stats	Mandatory	A reference to the FileSystemStatisticalData that hold the statistics for the exporting Port.

10.6.4 CIM_ElementStatisticalData (Local Filesystem Stats)

CIM_ElementStatisticalData is an association that relates a local filesystem to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of FileSystemStatistics. ElementStatisticalData describes the existence requirements and context for the FileSystemStatistics, relative to a specific local Filesystem.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "102" (Local Filesystem statistics support).

Table 145 describes class CIM_ElementStatisticalData (Local Filesystem Stats).

Table 145 - SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Local Filesystem Stats)

Properties	Requirement	Description & Notes
ManagedElement	Mandatory	A reference to a LocalFileSystem for which the Statistics apply.
Stats	Mandatory	A reference to the FileSystemStatisticalData that hold the statistics for the local Filesystem.

10.6.5 CIM_ElementStatisticalData (OTHER Element Type Stats)

CIM_ElementStatisticalData is an association that relates a provider-specified other element to its statistics. This other element is a filesystem-related managed element whose type is not explicitly declared within the list of ElementTypesSupported values defined within SNIA_FileSystemStatisticsCapabilities. Information describing the metered element in this case should also be provided in the SNIA_FileSystemStatisticalData.OtherElementTypeDescription property for the referenced instance of the FileSystemStatistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of FileSystemStatistics. ElementStatisticalData describes the existence requirements and context for the FileSystemStatistics, relative to the specific metered element.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: This is mandatory if SNIA_FileSystemStatisticsCapabilities.ElementTypesSupported = "1" (OTHER element type statistics support).

Table 146 describes class CIM_ElementStatisticalData (OTHER Element Type Stats).

Table 146 - SMI Referenced Properties/Methods for CIM_ElementStatisticalData (OTHER Element Type Stats)

Properties	Requirement	Description & Notes
ManagedElement	Mandatory	A reference to the provider-specified managed element for which the Statistics apply.
Stats	Mandatory	A reference to the FileSystemStatisticalData that hold the statistics for the provider-specified managed element.

10.6.6 CIM_HostedCollection (Client Defined)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Filesystem Performance Subprofile, it is used to associate a client-defined FileSystemStatisticsManifestCollections to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported.

Table 147 describes class CIM_HostedCollection (Client Defined).

Table 147 - SMI Referenced Properties/Methods for CIM_HostedCollection (Client Defined)

Properties	Requirement	Description & Notes
Antecedent	Mandatory	The top level ComputerSystem of the profile.
Dependent	Mandatory	A client defined FileSystemStatisticsManifestCollection.

10.6.7 CIM_HostedCollection (Default)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Filesystem Performance Subprofile, it is used to associate the default (provider-defined) FileSystemStatisticsManifestCollection to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 148 describes class CIM_HostedCollection (Default).

Table 148 - SMI Referenced Properties/Methods for CIM_HostedCollection (Default)

Properties	Requirement	Description & Notes
Antecedent	Mandatory	The top level ComputerSystem of the profile.
Dependent	Mandatory	The provider defined FileSystemStatisticsManifestCollection.

10.6.8 CIM_HostedCollection (Provider Supplied)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Filesystem Performance Subprofile, it is used to associate the StatisticsCollection to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 149 describes class CIM_HostedCollection (Provider Supplied).

Table 149 - SMI Referenced Properties/Methods for CIM_HostedCollection (Provider Supplied)

Properties	Requirement	Description & Notes
Antecedent	Mandatory	The top level ComputerSystem of the profile.
Dependent	Mandatory	The StatisticsCollection.

10.6.9 CIM_HostedService

CIM_HostedService is an association between a Service (SNIA_FileSystemStatisticsService) and the System (ComputerSystem) on which the functionality resides. Services are weak with respect to their hosting System. Heuristic: A Service is hosted on the System where the Filesystems or SoftwareFeatures that implement the Service are located.

CIM_HostedService is subclassed from CIM_HostedDependency.

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 150 describes class CIM_HostedService.

Table 150 - SMI Referenced Properties/Methods for CIM_HostedService

Properties	Requirement	Description & Notes
Antecedent	Mandatory	The hosting System.
Dependent	Mandatory	The Service hosted on the System.

10.6.10 CIM_MemberOfCollection (Member of client defined collection)

This use of MemberOfCollection is to Collect all Manifests instances in a client-defined manifest collection.

Created By: Extrinsic: AddOrModifyManifest

Modified By: Static

Deleted By: Extrinsic: RemoveManifests

Requirement: Clients can modify manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.

Table 151 describes class CIM_MemberOfCollection (Member of client defined collection).

Table 151 - SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of client defined collection)

Properties	Requirement	Description & Notes
Collection	Mandatory	A client defined manifest collection
Member	Mandatory	The individual Manifest Instance that is part of the set.

10.6.11 CIM_MemberOfCollection (Member of predefined collection)

This use of MemberOfCollection is to Collect all Manifests instances in the default manifest collection

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 152 describes class CIM_MemberOfCollection (Member of predefined collection).

Table 152 - SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of pre-defined collection)

Properties	Requirement	Description & Notes
Collection	Mandatory	The provider defined default manifest collection
Member	Mandatory	The individual Manifest Instance that is part of the set.

10.6.12 CIM_MemberOfCollection (Member of statistics collection)

This use of MemberOfCollection is to collect all FileSystemStorageStatisticalData instances (in the StatisticsCollection). Each association is created as a side effect of the metered object getting created.

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 153 describes class CIM_MemberOfCollection (Member of statistics collection).

Table 153 - SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of statistics collection)

Properties	Requirement	Description & Notes
Collection	Mandatory	The collection of all filesystem statistics data instances
Member	Mandatory	The individual filesystem statistics data Instance that is part of the set.

10.6.13 CIM_StatisticsCollection

The CIM_StatisticsCollection collects all filesystem statistics kept by the profile. There is one instance of the CIM_StatisticsCollection class and all individual metered element statistics can be accessed by using association traversal (using MemberOfCollection) from the StatisticsCollection.

CIM_StatisticsCollection is subclassed from CIM_SystemSpecificCollection.

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 154 describes class CIM_StatisticsCollection.

Table 154 - SMI Referenced Properties/Methods for CIM_StatisticsCollection

Properties	Requirement	Description & Notes
InstanceID	Mandatory	
ElementName	Mandatory	
SampleInterval	Mandatory	Minimum recommended polling/sampling interval for a system that provides filesystem support (e.g., NAS Head or Self-Contained NAS). It is set by the provider and cannot be modified.
TimeLastSampled	Mandatory	Time statistics table by object was last updated (Time Stamp in SMI 2.2 specification format)
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile

10.6.14 SNIA_AssociatedFileSystemStatisticsManifestCollection (Client defined collection)

The SNIA_AssociatedFileSystemStatisticsManifestCollection associates an instance of a SNIA_FileSystemStatisticsManifestCollection to the instance of CIM_StatisticsCollection to which it applies. Client defined manifest collections identify the Manifests (statistic properties) for retrieval of filesystem statistics.

SNIA_AssociatedFileSystemStatisticsManifestCollection is not subclassed from anything.

There will be one instance of the SNIA_AssociatedFileSystemStatisticsManifestCollection class, for each client defined manifest collection that has been created.

Created By: Extrinsic: CreateManifestCollection

Modified By: Static

Deleted By: Static

Requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.

Table 155 describes class SNIA_AssociatedFileSystemStatisticsManifestCollection (Client defined collection).

Table 155 - SMI Referenced Properties/Methods for SNIA_AssociatedFileSystemStatisticsManifestCollection (Client defined collection)

Properties	Requirement	Description & Notes
Statistics	Mandatory	The StatisticsCollection to which the manifest collection applies
ManifestCollection	Mandatory	A client defined manifest collection.

10.6.15 SNIA_AssociatedFileSystemStatisticsManifestCollection (Provider defined collection)

The SNIA_AssociatedFileSystemStatisticsManifestCollection associates an instance of a SNIA_FileSystemStatisticsManifestCollection to the instance of CIM_StatisticsCollection to which it applies. The default manifest collection defines the SNIA_FileSystemStatisticalData properties that are supported by the profile implementation.

SNIA_AssociatedFileSystemStatisticsManifestCollection is not subclassed from anything.

One instance of the SNIA_AssociatedFileSystemStatisticsManifestCollection shall exist for the default manifest collection if the Filesystem Performance Subprofile is implemented.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 156 describes class SNIA_AssociatedFileSystemStatisticsManifestCollection (Provider defined collection).

Table 156 - SMI Referenced Properties/Methods for SNIA_AssociatedFileSystemStatisticsManifestCollection (Provider defined collection)

Properties	Requirement	Description & Notes
Statistics	Mandatory	The StatisticsCollection to which the manifest collection applies
ManifestCollection	Mandatory	The default manifest collection.

10.6.16 SNIA_FileSystemStatisticalData

SNIA_FileSystemStorageStatisticalData is subclassed from CIM_StatisticalData.

Instances of this class will exist for each of the metered elements if the "ElementTypesSupported" property of the SNIA_FileSystemStatisticsCapabilities indicates that the metered element is supported. For example, if "Local Filesystem" is identified in the "ElementTypesSupported" property, then this indicates support for metering of the local filesystem.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 157 describes class SNIA_FileSystemStatisticalData.

Table 157 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticalData

Properties	Requirement	Description & Notes
InstanceID	Mandatory	The InstanceID for a FileSystemStatisticalData instance shall be unique across all instances of the FileSystemStatisticalData class.
StatisticTime	Mandatory	The time that the most recent measurement was taken, relative to the object (managed element) where the statistics were collected. (Time stamp in CIM 2.2 specification format).
ElementType	Mandatory	Defines the role that the metered element (object) played for which this statistics record was collected. This value is required AND the current version of SMI-S specifies the following values: ValueMap {"1", "102", "103", "104"} Values { "Other", "Local Filesystem", "Exported File Share", "Exporting Port"}
OtherElementTypeDescription	Mandatory	A string describing the type of element when the ElementType property of this class (or any of its subclasses) is set to 1 (i.e., "Other"). This property should be set to NULL when the ElementType property is any value other than 1.
TotalIOs	Mandatory	The cumulative count of file I/O operations for the object, including metadata I/O operations.

Table 157 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticalData

Properties	Requirement	Description & Notes
TotalBytesTransferred	Conditional	Conditional requirement: This property is required if the ElementType is 102, 103, or 104. The cumulative count of bytes transferred for all of the file I/O operations as defined in "TotalIOs" above. Note: This is not specified for the "Other" ElementType.
ReadIOs	Conditional	Conditional requirement: This property is required if the ElementType is 102 or 103.The cumulative count of file I/O operations that were directed to the object and that performed a transfer of data from the file contents. Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
WriteIOs	Conditional	Conditional requirement: This property is required if the ElementType is 102 or 103. The cumulative count of file I/O operations that were directed to the object and that performed a transfer of data to the file contents. Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
OtherIOs	Conditional	Conditional requirement: This property is required if the ElementType is 102 or 103. The cumulative count of file I/O operations that were directed to the object and that did not perform a transfer of data either to or from the file contents. This count excludes metadata I/ O operations (both read and write). File "open", "close", and "lock" I/O operations are examples of an "OtherIO" I/O operation. Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
MetadataReadIOs	Optional	The cumulative count of file I/O operations that were directed to the object and that performed a read transfer of metadata. "Get Attributes" and "Read Directory" I/O operations are examples of a Metadata read I/O operation. Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
MetadataWriteIOs	Optional	The cumulative count of file I/O operations that were directed to the object and that performed a write transfer of metadata. "Set Attributes" I/O operations are an example of a Metadata write I/O operation. Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.

Table 157 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticalData

Properties	Requirement	Description & Notes
TotalIOTimeCounter	Optional	The cumulative elapsed I/O operation time (number of ClockTickIntervals) for all file I/O operations as defined in "TotalIOs" above. The I/O operation response time is added to this counter at the completion of each measured I/O operation using ClockTickInterval units. The TotalIOTimeCounter value can be divided by the total number of I/O operations (TotalIOs) to obtain an I/O operation average response time. Note: This is not specified for the "Other" ElementType.
TotalIdleTimeCounter	Optional	The cumulative elapsed idle time using ClockTickInterval units. That is, the cumulative number of ClockTickIntervals for all idle time within the object, with "idle time" being that time during which no I/O operations were being processed by the object. Note: This is not specified for the "Other" ElementType.
ReadIOTimeCounter	Optional	The cumulative elapsed I/O operation time for all Read I/O operations (that is, the cumulative elapsed time for all Read I/O operations as defined in "ReadIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
BytesRead	Optional	The cumulative count of bytes read (that is, the cumulative count of bytes transferred by all Read I/O operations as defined in "ReadIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
WriteIOTimeCounter	Optional	The cumulative elapsed I/O operation time for all Write I/O operations (that is, the cumulative elapsed time for all Write I/O operations as defined in "WriteIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
BytesWritten	Optional	The cumulative count of bytes written (that is, the cumulative count of bytes transferred by all Write I/O operations as defined in "WriteIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
MetadataBytesRead	Optional	The cumulative count of metadata bytes read (that is, the cumulative count of bytes transferred by all Metadata read I/O operations as defined in "MetadataReadIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.

Table 157 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticalData

Properties	Requirement	Description & Notes
MetadataBytesWritten	Optional	The cumulative count of metadata bytes written (that is, the cumulative count of bytes transferred by all Metadata write I/O operations as defined in "MetadataWriteIOs" above). Note: This is not specified for the "Exporting Port" and the "Other" ElementTypes.
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile
ElementName	Optional	Not Specified in this version of the Profile
SampleInterval	Optional	Not Specified in this version of the Profile
StartStatisticTime	Optional	Not Specified in this version of the Profile
ResetSelectedStats()	Optional	Not Specified in this version of the Profile

10.6.17 SNIA_FileSystemStatisticsCapabilities

An instance of the SNIA_FileSystemStatisticsCapabilities class defines the specific support provided with the filesystem statistics implementation. Note: There would be zero or one instance of this class in a profile. There would be none if the profile did not support the Filesystem Performance Subprofile. There would be exactly one instance if the profile did support the Filesystem Performance Subprofile.

SNIA_FileSystemStatisticsCapabilities class is subclassed from CIM_Capabilities.

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 158 describes class SNIA_FileSystemStatisticsCapabilities.

Table 158 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsCapabilities

Properties	Requirement	Description & Notes
InstanceID	Mandatory	
ElementName	Mandatory	
ElementTypesSupported	Mandatory	ValueMap { "1", "102", "103", "104" }, Values {"Other", "Local Filesystem", "Exported File Share", "Exporting Port"}
SynchronousMethodsSupported	Mandatory	This property is mandatory, but the array may be empty. ValueMap { "2", "3", "4", "5", "6", "7", "8"}, Values {"Exec Query", "Indications", "QueryCollection", "GetStatisticsCollection", "Manifest Creation", "Manifest Modification", "Manifest Removal" }

Table 158 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsCapabilities

Properties	Requirement	Description & Notes
AsynchronousMethodsSupported	Optional	Not supported in current version of SMI-S.
ClockTickInterval	Mandatory	An internal clocking interval for all timers in the subsystem, measured in microseconds (Unit of measure in the timers, measured in microseconds). Time counters are monotonically increasing counters that contain "ticks". Each tick represents one ClockTickInterval. If ClockTickInterval contained a value of 32 then each time counter tick would represent 32 microseconds.
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile
CreateGoalSettings()	Optional	Not Specified in this version of the Profile

10.6.18 SNIA_FileSystemStatisticsManifest (Client Defined)

The SNIA_FileSystemStatisticsManifest class is a Concrete class that defines the SNIA_FileSystemStorageStatisticalData properties that should be returned on a GetStatisticsCollection request.

SNIA_FileSystemStatisticsManifest is subclassed from CIM_ManagedElement.

In order for a client defined instance of the SNIA_FileSystemStatisticsManifest class to exist, all of the manifest collection manipulation functions shall be identified in the "SynchronousMethodsSupported" property of the SNIA_FileSystemStatisticsCapabilities (FileSystemStatisticsCapabilities.SynchronousMethodsSupported = "6") instance, AND a client must have created at least ONE instance of SNIA_FileSystemStatisticsManifestCollection.

Created By: Extrinsic: AddOrModifyManifest

Modified By: Extrinsic: AddOrModifyManifest

Deleted By: Extrinsic: RemoveManifests

Requirement: Clients can modify manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.

Table 159 describes class SNIA_FileSystemStatisticsManifest (Client Defined).

Table 159 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifest (Client Defined)

Properties	Requirement	Description & Notes
ElementName	Mandatory	A Client defined string that identifies the manifest.
InstanceID	Mandatory	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.

Table 159 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifest (Client Defined)

Properties	Requirement	Description & Notes
ElementType	Mandatory	This value is required AND the current version of SMI-S specifies the following values: ValueMap {"1", "102", "103", "104"} Values { "Other", "Local Filesystem", "Exported File Share", "Exporting Port"}
OtherElementTypeDescription	Mandatory	A string describing the type of element when the ElementType property of this class (or any of its subclasses) is set to 1 (i.e., "Other"). This property should be set to NULL when the ElementType property is any value other than 1.
IncludeStatisticTime	Mandatory	
IncludeTotalIOs	Mandatory	
IncludeTotalBytesTransferred	Mandatory	
IncludeReadIOs	Mandatory	
IncludeWriteIOs	Mandatory	
IncludeOtherIOs	Mandatory	
IncludeMetadataReadIOs	Mandatory	
IncludeMetadataWriteIOs	Mandatory	
IncludeTotalIOTimeCounter	Mandatory	
IncludeTotalIdleTimeCounter	Mandatory	
IncludeReadIOTimeCounter	Mandatory	
IncludeBytesRead	Mandatory	
IncludeWriteIOTimeCounter	Mandatory	
IncludeBytesWritten	Mandatory	
IncludeMetadataBytesRead	Mandatory	
IncludeMetadataBytesWritten	Mandatory	
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile
IncludeStartStatisticTime	Optional	Not Specified in this version of the Profile

10.6.19 SNIA_FileSystemStatisticsManifest (Provider Support)

The SNIA_FileSystemStatisticsManifest class is a Concrete class that defines the SNIA_FileSystemStatisticalData properties that are supported by the Provider. These Manifests are established by the Provider for the default manifest collection.

SNIA_FileSystemStatisticsManifest is subclassed from CIM_ManagedElement.

At least one Provider supplied instance of the SNIA_FileSystemStatisticsManifest class shall exist, if the Filesystem Performance Subprofile is supported.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 160 describes class SNIA_FileSystemStatisticsManifest (Provider Support).

Table 160 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifest (Provider Support)

Properties	Requirement	Description & Notes
ElementName	Mandatory	A Provider defined string that identifies the manifest in the context of the Default Manifest Collection.
InstanceID	Mandatory	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.
ElementType	Mandatory	This value is required AND the current version of SMI-S specifies the following values: ValueMap {"1", "102", "103", "104"} Values { "Other", "Local Filesystem", "Exported File Share", "Exporting Port"}
OtherElementTypeDescription	Mandatory	A string describing the type of element when the ElementType property of this class (or any of its subclasses) is set to 1 (i.e., "Other"). This property should be set to NULL when the ElementType property is any value other than 1.
IncludeStatisticTime	Mandatory	
IncludeTotalIOs	Mandatory	
IncludeTotalBytesTransferred	Mandatory	
IncludeReadIOs	Mandatory	
IncludeWriteIOs	Mandatory	
IncludeOtherIOs	Mandatory	
IncludeMetadataReadIOs	Mandatory	
IncludeMetadataWriteIOs	Mandatory	
IncludeTotalIOTimeCounter	Mandatory	
IncludeTotalIdleTimeCounter	Mandatory	
IncludeReadIOTimeCounter	Mandatory	
IncludeBytesRead	Mandatory	
IncludeWriteIOTimeCounter	Mandatory	

Table 160 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifest (Provider Support)

Properties	Requirement	Description & Notes
IncludeBytesWritten	Mandatory	
IncludeMetadataBytesRead	Mandatory	
IncludeMetadataBytesWritten	Mandatory	
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile
IncludeStartStatisticTime	Optional	Not Specified in this version of the Profile

10.6.20 SNIA_FileSystemStatisticsManifestCollection (Client Defined)

An instance of a client defined SNIA_FileSystemStatisticsManifestCollection defines the set of Manifests to be used in the retrieval of Filesystem statistics by the GetStatisticsCollection method.

SNIA_FileSystemStatisticsManifestCollection is subclassed from CIM_SystemSpecificCollection.

In order for a client defined instance of the SNIA_FileSystemStatisticsManifestCollection class to exist, then all the manifest collection manipulation functions shall be identified in the "SynchronousMethodsSupported" property of the SNIA_FileSystemStatisticsCapabilities instance and a client must have created a Manifest Collection..

Created By: Extrinsic: CreateManifestCollection

Modified By: Static

Deleted By: Static

Requirement: Clients can create manifests as identified by SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported.

Table 161 describes class SNIA_FileSystemStatisticsManifestCollection (Client Defined).

Table 161 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifestCollection (Client Defined)

Properties	Requirement	Description & Notes
InstanceID	Mandatory	
ElementName	Mandatory	A client defined user-friendly name for the manifest collection. It is set during creation of the Manifest Collection through the ElementName parameter of the CreateManifestCollection method.
IsDefault	Mandatory	Denotes whether or not this manifest collection is a provider defined default manifest collection. For the client defined manifest collections this is set to "false".
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile

10.6.21 SNIA_FileSystemStatisticsManifestCollection (Provider Defined)

An instance of a default SNIA_FileSystemStatisticsManifestCollection defines the set of Manifests that define the properties supported for each ElementType supported for the implementation. It can also be used by clients in retrieval of Filesystem statistics by the GetStatisticsCollection method.

SNIA_FileSystemStatisticsManifestCollection is subclassed from CIM_SystemSpecificCollection.

At least ONE SNIA_FileSystemStatisticsManifestCollection shall exist if the Filesystem Performance Subprofile is implemented. This would be the default manifest collection that defines the properties supported by the implementation.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 162 describes class SNIA_FileSystemStatisticsManifestCollection (Provider Defined).

Table 162 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsManifestCollection (Provider Defined)

Properties	Requirement	Description & Notes
InstanceID	Mandatory	
ElementName	Mandatory	For the default manifest collection, this should be set to "DEFAULT".
IsDefault	Mandatory	Denotes whether or not this manifest collection is a provider defined default manifest collection. For the default manifest collection this is set to "true".
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile

10.6.22 SNIA_FileSystemStatisticsService

The SNIA_FileSystemStatisticsService class provides methods for statistics retrieval and Manifest Collection manipulation.

The SNIA_FileSystemStatisticsService class is subclassed from CIM_Service.

There shall be an instance of the SNIA_FileSystemStatisticsService, if the Filesystem Performance Subprofile is implemented. It is not necessary to support any methods of the service, but the service shall be populated.

The methods that are supported can be determined from the SynchronousMethodsSupported and AsynchronousMethodsSupported properties of the SNIA_FileSystemStatisticsCapabilities.

Created By: Static

Modified By: Static

Deleted By: Static

Requirement: Mandatory

Table 163 describes class SNIA_FileSystemStatisticsService.

Table 163 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsService

Properties	Requirement	Description & Notes
SystemCreationClassName	Mandatory	
SystemName	Mandatory	
CreationClassName	Mandatory	
Name	Mandatory	
Caption	Optional	Not Specified in this version of the Profile
Description	Optional	Not Specified in this version of the Profile
ElementName	Optional	Not Specified in this version of the Profile
OperationalStatus	Optional	Not Specified in this version of the Profile
StatusDescriptions	Optional	Not Specified in this version of the Profile
InstallDate	Optional	Not Specified in this version of the Profile
HealthState	Optional	Not Specified in this version of the Profile
TimeOfLastStateChange	Optional	Not Specified in this version of the Profile
OtherEnabledState	Optional	Not Specified in this version of the Profile
EnabledDefault	Optional	Not Specified in this version of the Profile
RequestedState	Optional	Not Specified in this version of the Profile
EnabledState	Optional	Not Specified in this version of the Profile
Started	Optional	Not Specified in this version of the Profile
PrimaryOwnerName	Optional	Not Specified in this version of the Profile
PrimaryOwnerContact	Optional	Not Specified in this version of the Profile
GetStatisticsCollection()	SynchGSC|AsynchGSC	Support for this method is conditional on SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported containing '5' (GetStatisticsCollection). This method retrieves all statistics kept for the profile as directed by a manifest collection.
CreateManifestCollection()	SynchMC|AsynchMC	Support for this method is conditional on SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported containing '6' (Manifest Creation). This method is used to create client defined manifest collections.

Table 163 - SMI Referenced Properties/Methods for SNIA_FileSystemStatisticsService

Properties	Requirement	Description & Notes
AddOrModifyManifest()	SynchMM|AsynchMM	Support for this method is conditional on SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported containing '7' (Manifest Modification). This method is used to add or modify filesystem statistics manifests in a client defined manifest collection.
RemoveManifests()	SynchRM|AsynchRM	Support for this method is conditional on SNIA_FileSystemStatisticsCapabilities.SynchronousMethodsSupported or SNIA_FileSystemStatisticsCapabilities.AsynchronousMethodsSupported containing '8' (Manifest Removal). This method is used to remove a filesystem statistics manifest from a client defined manifest collection.
RequestStateChange()	Optional	Not Specified in this version of the Profile
StopService()	Optional	Not Specified in this version of the Profile
StartService()	Optional	Not Specified in this version of the Profile

EXPERIMENTAL

EXPERIMENTAL

Clause 11: Filesystem Quotas Profile

- Profile Name: Filesystem
- Version: 1.2.0
- Organization: SNIA
- CIM schema version: 2.13
- Central Class: LocalFileSystem
- Scoping Class: ComputerSystem

11.1 Description

The Filesystem Quotas Profile is intended to provide management of quotas on the use of filesystem resources--raw space and inodes especially--by the common filesystem principals. User, group and tree quotas are modeled. *Trees* means *directories* (rooted directory hierarchy structures) within filesystems. Some systems allow quotas only on directories that have some special distinguishing feature, others allow them on arbitrary directories.

Quota systems work by keeping statistics-in real time-on the space used by each monitored principal/container pair e.g. a user and her home share. They then trigger events when filesystem writes cause the space used by the principal to exceed some threshold. There are four common varieties of quota thresholds:

1. Hard. Hitting a hard quota threshold causes subsequent writes by the principal to the affected container to fail. In POSIX, the error returned to the client is ENOSPC (no space). An indication is also thrown.
2. Soft. A soft threshold causes the system to issue a warning. Systems variously issue warnings via CLI, SNMP traps, pop-up windows at the user station, audit log entries, email and other proprietary methods. This profile provides required indications for this purpose.
3. Soft, with grace period. This type of quota is really a refinement of soft quotas. It functions like a soft quota until the grace period expires, at which point it begins behaving like a hard quota.
4. Monitoring. When a monitoring quota is in effect, there are no thresholds, and no warnings are issued. This type of quota is useful because the underlying system keeps track of the relevant usage statistics in real time for all quotas. A monitoring quota permits an administrator to simply issue a command to print a quota report instead of having to do a complete filesystem scan to get the usage information of interest.

In general, it is not possible to have a quota system that meets the above semantics without some level of access to the data path. More loosely coupled systems may need to relax the semantics of the hard limit, for example, and may not actually trigger an event until a file is closed, for example. This profile allows these semantic variations.

Some systems allow "default" quotas for users, groups and/or trees. A default user quota, by way of example, is used for every user of the system who does not have a quota entry specific to them.

11.1.1 Tree Quotas

Tree quotas are quotas on directory trees with an identifiable root. Mounts and symlinks are not followed. In other words, a directory which contains nothing but mount points and symbolic links may satisfy a very small quota, even though the amount of data addressable by entries in the container is large.

Hard links are another matter. The policy is system-dependent, but a common behavior is that if a file or directory is hard-linked in two separate trees with separate tree quotas, the space used is charged against both quotas.

Pure tree quotas apply no matter which principal does the writing. There are some caveats however.

- Root on some systems is not constrained by quotas.
- An entity with sufficient privilege may not be constrained by quotas on some systems (e.g. a Windows user with BackupOperator privilege).

Some systems may support tree quotas only on directories with certain special characteristics. Directories may be constrained to being top-level, for example. This profile does not specify a means for determining whether a given directory may have a tree quota set on it.

11.1.2 User Quotas

User quotas are limits on how much space a principal with a given User ID can use. They may be either global or restricted by namespace tree, as well as by filesystem.

11.1.3 Group Quotas

Group quotas set limits on how much all the members of a group with a given Group ID can use in the aggregate. They are not, therefore, quotas which apply to each member of a group. This follows Unix usage. Group quotas only work on systems which have the concept of a primary group id (PGID), as the system needs to know which group to charge writes against. As NTFS does not have the concept of a primary group, it does not do group quotas. (Note: There is a primary group field that can be discovered on a file in NTFS. This is for POSIX support, however, and does not always contain anything meaningful)

Group quotas may be global or applicable to a given namespace tree and/or filesystem.

11.1.4 Container Boundaries

Quotas may be system-wide, or scoped to an individual filesystem. Not all systems support both of these, however, so provision is made for both. The SupportedPrincipalTypes array in the FSQuotaCapabilities class distinguishes between User, Group, Tree, User-Tree and Group-Tree quotas as follows:

- User Quotas and Group quotas are described in 11.1.2 and 11.1.3.
- A Tree Quota is a limit on the storage (or other limit such as number of inodes) used by a directory tree. This quota is not specific to a user or a group but applies to all users and groups creating files and directories within the tree.
- A User-Tree quota is a limit on the storage used by a user within a directory tree and is applied in parallel with the Tree Quota (both must be satisfied).
- A Group-Tree quota is a limit on the aggregate storage used by a group of users within a directory tree and is applied in parallel with the Tree Quota (both must be satisfied).

If a Tree is configured with a Tree Quota as well as User-Tree and Group-Tree quotas, they must all be satisfied.

11.1.5 Quota types

Quotas are usually limits on disk space. Some systems, however, also support limits on the number of files and/or directories.

11.1.6 Class design considerations

11.1.6.1 New Classes

This profile uses several new classes—FSDomainIdentity, FSQuotaCapabilities, FSQuotaReportRecord, FSQuotaConfigEntry, FSQuotaManagementService, and FSQuotaIndication

11.1.6.1.1 FSDomainIdentity

Due to the in-band nature of quota tracking, the identifier of the principal being managed needs to be small and easily optimized. Traditional systems use Unix UIDs and GIDs, which are 32-bit quantities, or SIDs which are short strings. To tie these into CIM, this new class is specified. Each instance contains a string with the UID, GID or SID, respectively, in it, and enums for the type of domain and principal.

11.1.6.1.2 FSQuotaCapabilities

This Capabilities subclass lists the properties in a FSQuotaConfigEntry that are supported by the underlying system. The client shall not attempt to set any properties which are not listed as supported in the instance of this class associated to the service. It shall instead always populate unsupported properties with null.

There shall exist exactly one instance of this class per FSQuotaManagementService instance.

11.1.6.1.3 FSQuotaReportRecord

When running a quota report, the underlying system generally issues a text file, each line or group of lines representing the status of a filesystem principal with respect to one quota configuration entry. There may be hundreds of thousands of these records, and they are not keyed, meaning that there is no way to go back and fetch any given one of them. Therefore FSQuotaReportRecord is derived from a new proposed abstract root class called ReportRecord, which carries the Indication qualifier. Note that this qualifier does not mean that these classes are subclasses of CIM_Indication. It's used because it's the only way, currently, to construct a class in CIM which does not require a key.

11.1.6.1.4 FSQuotaConfigEntry

An FSQuotaConfigEntry instance represents a single quota entry supported by the system. For example, one FSFQuotaConfigEntry instance may specify that on filesystem “foo,” in directory “/bar,” the user “joe” is restricted to 1GB of space, and should receive a warning at 0.97GB.

A quota entry has been defined as a complex of several classes and associations. If implementation experience turns up performance considerations related to this, this design may need to be revisited.

Many systems do not support any method of identifying individual FSQuotaConfigEntry instances, as they simply represent lines in a text file, and the underlying system may not care about duplicates or conflicts. However, FSQuotaConfigEntry instances need to be modified; this corresponds to editing the corresponding line in the file. Therefore, if the underlying system does not expose a key, one may be created by composing the PrincipalID property, a unique reference to the FileSystem or ComputerSystem to which the entry applies (from the association FSQuotaAppliesToElement), the TreeName property (if a tree quota), the measured quantity type (the ResourceType property), the quota type (QuotaType property), and its default status (the Default property). An implementation may expose the algorithm used to compose the key so that the client may decompose it, but this is not required by this version of the profile. Upon creation of a new quota instance, clients shall verify that no quota with the same key already exists. Upon modification of an instance, clients shall modify all instances whose keys match that instance key.

- PrincipalID: This indicates a user by the user’s UID or SID, or a group by the group’s GID or SID, or it can be the empty string. It is interpreted in the context of a directory service specified for the file server ComputerSystem that is associated to the FileSystem by a Dependency association.
- InstanceID. This property is a unique identifier for this FSQuotaConfigEntry. This property shall be unique in the presence of multiple instances of ComputerSystem and FileSystem, and multiple properties of QuotaType, Default, ResourceType and PrincipalID. It may be constructible by the client, but this profile does not specify this format.

11.1.6.1.5 FSQuotaManagementService

The FSQuotaManagementService provides the interface to the underlying system for most operations which are overtly related to quotas. There shall be at most one instance of a FSQuotaManagementService for each underlying ComputerSystem.

11.1.6.1.6 FSQuotaIndication

The FSQuotaIndication class provides information about threshold crossing events, meaning that a quota has just been exceeded.

11.1.7 Instance Diagram

Figure 16 shows the Filesystem Quotas instance diagram.

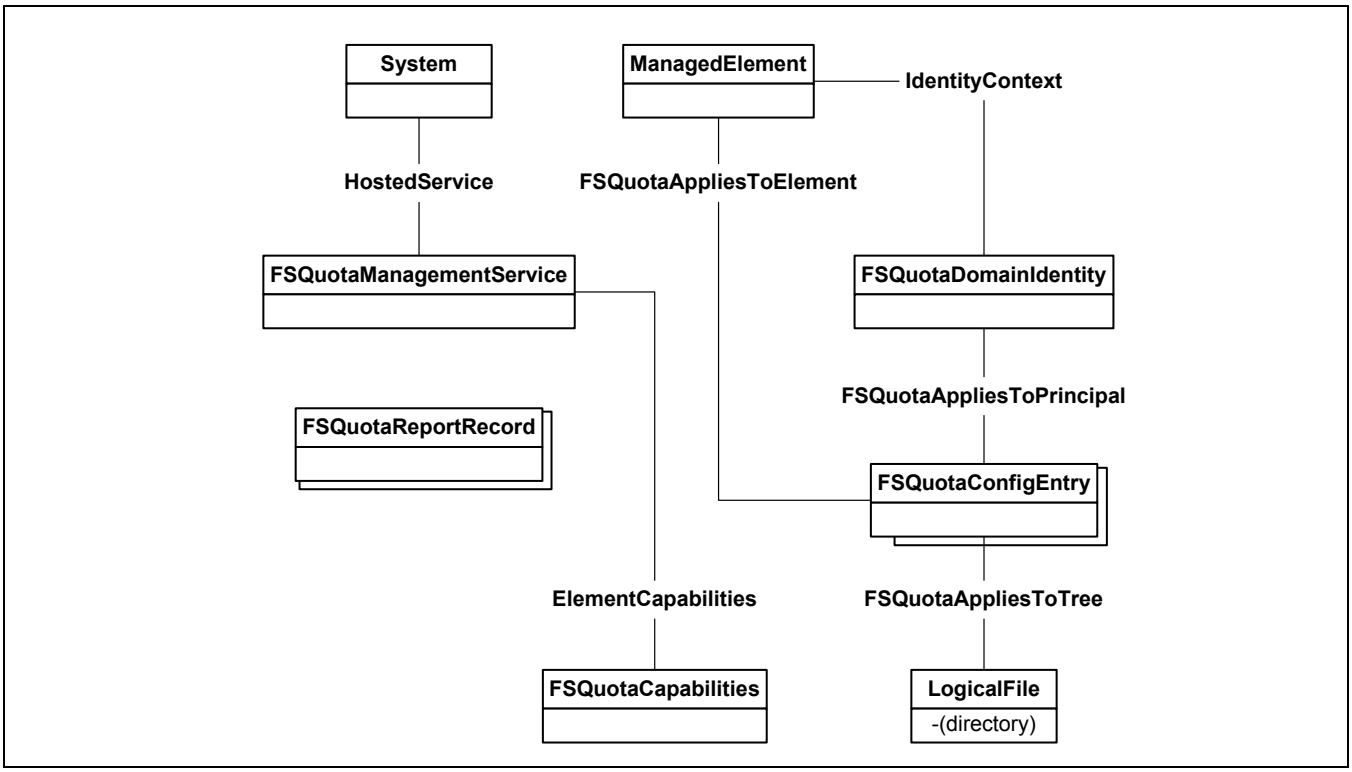

Figure 16 - Filesystem Quotas Instance Diagram

11.2 Health and Fault Management Considerations

None currently applicable.

11.3 Supported Profiles, Subprofiles, and Packages

The Filesystem and Indications Profiles are required by this profile.

Table 164 describes the supported profiles for FileSystem Quotas.

Table 164 - Supported Profiles for FileSystem Quotas

Registered Profile Names	Mandatory	Version
Filesystem	Yes	1.3.0
Indication	Yes	1.3.0
Job	No	1.3.0

11.4 Methods of the Profile

All profile methods are contained in the FSQuotaManagementService.

11.4.1 FindQuotaEntries

```
uint32 FindQuotaEntries(
        IN string IdentityId,
        IN ManagedElement REF Element,
        IN string Tree,
        IN  uint16 QuotaType,
        OUT EmbeddedInstance("SNIA_FSQuotaConfigEntry")string QuotaEntries[],
```

Given a combination of inputs, FindQuotaEntries( ) searches the quota entry database on the managed device for quota entries that match, and returns a list. On systems that support it, long-running queries may return a job.

Possible quota entries are:

1) IdentityId

IdentityId is an optional string that can specify the UID, GID, or SID or can specify a pattern. The following rules apply to IdentityId:

a) If IdentityId is NULL or the empty string, no identity-based quotas should be returned.

b) If IdentityID is NULL, default quotas will be returned.

c) If IdentityId is “*”, this matches all identity-based quotas entries.

d) IdentityId may also be a specific ID, or the prefix of an ID followed by "*". Standard prefix string matching is used to determine which entries match in this case.

2) Element

Element is an optional reference to a ManagedElement (declared as CIM_ManagedElement REF Element). The following rules apply to Element:

a) When a ComputerSystem is passed for Element, the returned entries may be default or other entries for both the ComputerSystem and its contained FileSystems.

b) When a FileSystem is passed in for Element, only entries pertaining to that FileSystem shall be returned. This may include default entries applicable to that FileSystem.

c) If NULL is passed in for Element, the FSQuotaManagementService assumes that the ComputerSystem it is hosted on is the Element.

Element is an optional reference to a ComputerSystem or a LocalFileSystem, but is declared as a reference to a ManagedElement, which is the only common parent class.

3) Tree

Tree is an optional string that identifies a tree (or trees) contained within a filesystem. The following rules apply to Tree:

a) A null or empty string indicates that no tree quota entries should be returned.

b) A “*” tree parameter matches all tree quota entries defined within the filesystem(s) indicated by Element, if any.

c) If Tree contains a path, it matches that path only. On some systems, this may result in multiple matches, one for the same-named tree in each of several filesystems.

d) Prefix string matching may be used for the Tree parameter. E.g. the path "/x/y/*" will match tree quotas on both "/x/y/m" and "/x/y/p".

4) QuotaType

QuotaType is an enumerated optional parameter that describes what type of quota entries should be returned. The following rules apply to QuotaType:

a) NULL or Unknown matches any entry (i.e., it is a wildcard).

b) If Tree is specified, then entries which are pure tree quotas (not user-tree or group-tree) match.

c) If User-Tree or Group-Tree is specified, then only user-tree or group-tree quotas match.

11.4.2 DeleteQuotaEntry

```
uint32 DeleteQuotaEntry(IN string EntryID);
```

This routine deletes a given quota entry from the managed device's quota entry database. Recall that the ManagedElement's name is specified as part of a QuotaEntry's InstanceID, above. A CIMOM managing multiple devices may use that to find which device to address when deleting the actual entry.

11.4.3 ModifyQuotaEntry

```
uint32 ModifyQuotaEntry(
         IN string EntryId,
         IN EmbeddedInstance("SNIA_FSQuotaConfigEntry") string QuotaEntry,
         OUT CIM_Job REF Job;
      );
```

Given the InstanceID of an existing quota entry, ModifyQuotaEntry( ) replaces it with a new entry specified as an EmbeddedInstance.

11.4.4 AddQuotaEntry

```
uint32 AddQuotaEntry(
         IN EmbeddedInstance("SNIA_FSQuotaConfigEntry") string QuotaEntry,
         OUT CIM_Job REF Job;
      );
```

This routine adds a new quota entry to the quota entry database on the appropriate managed element.

The ConflictingEntriesUsage property in FSQuotaCapabilities (see 11.7) will govern what happens if an entry already exists with the same combination of PrincipalID, ManagedElement, TreeName, ResourceType, QuotaType, and Default.

11.4.5 GetQuotaReport

```
uint32 GetQuotaReport(
         IN CIM_ManagedElement REF Element,
         IN string Tree,
         IN string User,
         IN EmbeddedInstance("FSQuotaDomainIdentity") string Group,
         IN, OUT string Cursor,
```

```
            IN, OUT uint64 NQuotas,
            OUT  CIM_Job REF Job,
            OUT EmbeddedInstance("SNIA_FSQuotaReportRecord") string ReportRecs[];
        );
```

This routine gets a quota report from a managed element. As there may be millions of records in this report, a chunking mechanism is provided so that the client does not become overwhelmed by the quantity of data furnished by the server.

The initial call to GetQuotaReport shall pass in NULL as a Cursor. Subsequent calls shall pass back the cursor exactly as received from the server, without modification, as an indication of where to continue the report from.

Clients which do not wish to use the chunking mechanism may pass a number such as 2^{63} - 1 in NQuotas.

On many systems, the initial phase of preparing a quota report may take some time. A job is returned in this case.

11.4.6 EnableQuotas

```
uint32 EnableQuotas(
        IN Boolean OnOff,
        IN CIM_ManagedElement element,
        OUT CIM_Job REF Job
    );
```

This routine turns quota management on or off on a given managed element. This routine shall always be supported when the element is a CIM_ComputerSystem. On systems that support it, the ManagedElement may alternatively be a filesystem. If an attempt is made to change the state on an unsupported ManagedElement, the routine shall return an appropriate error (“Operation unsupported for individual MEs of this type”).

11.4.7 InitializeQuotas

```
uint32 InitializeQuotas(
  IN CIM_ComputerSystem REF Server,
  OUT CIM_Job REF Job);
```

Some systems require an explicit initialization step before quotas may be used. If this step takes some time, a job shall be returned. Systems which do not require this step shall return “Success”.

11.5 Client Considerations and sample code

Because quota management capabilities vary so widely from device to device, clients must be prepared to receive "unsupported" errors. These can be kept to a minimum by inspecting the QuotaCapabilities of the managed device. See the QuotaGetCapabilities routine in 11.5.1.

There are five fundamental operations on quotas:

1. Initialize the quota management system
2. Turn quota tracking on or off
3. Add or modify a quota table entry
4. Read the quota table
5. Get a report on quota usage for one or all entries in the quota table

The QuotaManagementService class contains extrinsics for all of these things, so each task reduces to getting the service instance and invoking the desired method.

The following example code is advisory, not normative.

EXPERIMENTAL

11.5.1 Common subroutines

In the Filesystem Quotas sample routines, the following subroutines are used (and declared inline here):

```
sub CIM_QuotaManagementService QuotaGetQMService(
                  IN REF CIM_System system);
{
     services = Associators(system,
        "CIM_HostedService",
        "CIM_QuotaManagementService",
        "Antecedent",
        "Dependent",
        false, false, NULL);

     return services[0];
}

sub CIM_QuotaCapabilities QuotaGetCapabilities(
               IN REF CIM_System system)
{
     service = QuotaGetQMService(system);

     caps = Associators(service,
        "CIM_ElementCapabilities",
        "CIM_QuotaCapabilities",
        "CIM_ManagedElement",
        "ManagedElement",
        "Capabilities",
        false, false, NULL);

     return caps[0];
}

sub boolean QuotaSupportsPrincipalType(
               IN REF CIM_System system,
               IN uint16 type)
{
     capabilities = QuotaGetCapabilities(system);

     for(i = 0; capabilities.SupportedPrincipalTypes[i] != NULL; ++) {
        if (capabilities.SupportedPrincipalTypes[i] == type) {
           return TRUE;
        }
     }
```

```
     return FALSE;
}
```

All of the following routines may return errors indicating that the supplied managed element is not supported. In most cases this will be because the operation (e.g. initializing quotas) is a system-wide operation, and cannot be done on a per-filesystem basis.

EXPERIMENTAL

EXPERIMENTAL

11.5.2 Initialize quotas

```
sub uint_16 InitializeQuotas(
            IN REF CIM_System system)
{
     qms = QuotaGetQMService(system);

     result = qms->InitializeQuotas(system, job);

     //
     // See the Job Control profile for information on
     // handling the job if one is returned.
     //
     return result;
}
```

EXPERIMENTAL

EXPERIMENTAL

11.5.3 Enable or disable quota tracking

```
//
// enable or disable quotas
//
// See the mof for the EnableQuotas extrinsic for possible
// return values
//
sub uint16 EnableQuotas(IN REF CIM_System system,
                        IN REF CIM_ManagedElement me,
                        IN boolean onoff)
{
     qms = QuotaGetQMService(system);
     result = qms->EnableQuotas(onoff, me, job);

     //
     // See the Job Control profile for information on
```

```
        // handling the job if one is returned.
        //
        return result;
    }
```

EXPERIMENTAL

EXPERIMENTAL

11.5.4 Add a quota entry

```
    sub uint16 AddQuotaEntry(IN REF CIM_System system,
                  IN REF CIM_ManagedElement me,
                  IN String tree,
                  IN REF CIM_DomainIdentity principal,
                  IN uint64 hardlimit,
                  IN uint64 softlimit,
                  IN uint64 graceperiod,
                  IN boolean active,
                  IN string restype,
                  IN uint16 quotatype,
                  IN REF logicalfile,
                  IN REF me,
                  IN boolean default)
    {
        service = QuotaGetQMService(system);
        entry = CreateInstance("SNIA_FSQuotaConfigEntry");
        entry->HardLimit = hardlimit;
        entry->SoftLimit = softlimit;
        entry->SoftLimitGracePeriod = graceperiod;
        entry->Active = active;
        switch (restype) {
            case "Bytes": entry->ResourceType = 2;
            case "Files": entry->ResourceType = 3;
            case "Directories": entry->ResourceType = 4;
            case "Files+Directories": entry->ResourceType = 5;
            case "Inodes": entry->ResourceType = 6;
            default: entry->ResourceType = 0;
        }
        switch (quotatype) {
            case "User": entry->QuotaType = 2;
            case "Group": entry->QuotaType = 3;
            case "Tree": entry->QuotaType = 4;
            default: entry->QuotaType = 0;
        }
        if (principal != NULL) {
            entry->PrincipalID = principal->PrincipalID;
        else
```

```
        entry->PrincipalID = NULL;
    if (logicalfile != NULL) {
        entry->TreeName = logicalfile->Name;
    else
        entry->TreeName = NULL;
    entry->ManagedElement = me;
    entry->Default = default;
    entry->InstanceID = NULL;

    result = service->AddQuotaEntry(entry, job);

    //
    // analyse error, if there is one: the above code
    // cannot return '1' or '3', so only '2' is left.
    // And that means there's already an identical
    // entry, so declare victory and move on.
    //
    return result;      // could return 0, if you prefer
}
```

EXPERIMENTAL

EXPERIMENTAL

11.5.5 Delete a quota entry

```
//
// See the mof for the DeleteQuotaEntry extrinsic for possible
// return values
//
sub uint16 DeleteQuotaEntry(IN REF CIM_System system,
                            IN string entryid,
                            OUT REF CIM_Job job)
{
    service = QuotaGetQMService(system);
    result = service->DeleteQuotaEntry(entryid);

    return result;
}
```

EXPERIMENTAL

EXPERIMENTAL

11.5.6 Modify a quota entry

```
//
```

```
// There are many ways to modify a quota entry.  Here are
// a couple examples
//
sub uint16 ModifyQuotaHardLimit(IN REF CIM_System system,
                                IN string entryid,
                                IN uint64 newlimit)
{
     service = QuotaGetQMService(system);
     entry = GetInstance(entryid);
     entry->HardLimit = newlimit;
     result = service->ModifyQuotaEntry(entryid, entry, job);
     //
     // See the Job Control profile for information on
     // handling the job if one is returned.
     //
     return result;


sub uint16 SpecificUserToDefault(IN REF CIM_System system,
                                IN string uid)
{
     //
     // change Alice's quota to be the default for
     // all users
     //
     service = QuotaGetQMService(system);

     //
     // Need to search through all the quota entry instances
     // for the given uid.
     //
     qes[] = EnumerateInstances("SNIA_FSQuotaConfigEntry",
               true, false, false, false, "PrincipalID");
     foreach qe (qes[]) {
        if (qe->PrincipalID == uid) {
            qe->PrincipalID = NULL);
            qe->Default = true;
            return 0;
        }
     }
     return 1;        // not found
}
```

EXPERIMENTAL

EXPERIMENTAL

11.5.7 Read the quota entries

```
//
// Warning: on some systems, this may return 10's of
// thousands of entries
//
sub FSQuotaConfigEntry[] ReadQuotaEntries(IN REF CIM_System system)
{
     service = QuotaGetQMService(system);
     service->FindQuotaEntries(NULL, system, NULL, NULL, NULL,
                               qes[], job);

     //
     // See the Job Control profile for information on
     // handling the job if one is returned.
     //
     return qes[];
}
```

EXPERIMENTAL

EXPERIMENTAL

11.5.8 Get a report on quota usage

```
sub FSQuotaReportRecord[] GetQuotaReport(IN REF CIM_System system)
{
     cursor = NULL;
     service = QuotaGetQMService(system);
     nrecs = 1000;
     r = service->GetQuotaReport(system, NULL, NULL, NULL,
                     cursor, nrecs, job, recs[]);
        <manage job>;
        <do something with recs>;
        while (r != "No more data") {
            r = service->GetQuotaReport(system, NULL, NULL, NULL,
                     cursor, nrecs, job, recs[]);
            <manage job>;
            <do something with recs>;
        }
     }
}
```

EXPERIMENTAL

11.6 Registered Name and Version

FileSystem Quotas version 1.3.0

11.7 CIM Elements

Table 164 describes the CIM elements for FileSystem Quotas.

Table 165 - CIM Elements for FileSystem Quotas

Element Name	Requirement	Description
11.7.1 SNIA_FSDomainIdentity	Mandatory	A small class containing the unique ID of a user or group in a Unix or Windows domain
11.7.2 SNIA_FSQuotaAppliesToElement	Mandatory	An association between a quota config entry and a managed element
11.7.3 SNIA_FSQuotaAppliesToPrincipal	Mandatory	An association between a quota config entry and a Filesystem principal entity
11.7.4 SNIA_FSQuotaAppliesToTree	Mandatory	An association between a quota config entry and a directory
11.7.5 SNIA_FSQuotaCapabilities	Mandatory	The supported targets, quota types, resource types and behaviors of the FSQuotaManagementService associated to this class instance.
11.7.6 SNIA_FSQuotaConfigEntry	Mandatory	A single quota entry in the configuration database.
11.7.7 SNIA_FSQuotaIndication	Optional	An indication specially referring to quota events. Note that the threshold and current value are passed in the parent class, in ThresholdValue and ObservedValue
11.7.8 SNIA_FSQuotaManagementService	Mandatory	Quota Management Service class.
11.7.9 SNIA_FSQuotaReportRecord	Mandatory	A class representing a single line in a quota report generated by a call to the QuotaReport() extrinsic of the FSQuotaManagementService
11.7.10 SNIA_ReportRecord	Mandatory	An abstract keyless class proposed as the root of a tree of report record classes
SELECT * FROM SNIA_FSQuotaIndication WHERE WhichLimit = 2	Mandatory	Hard quota threshold crossed
SELECT * FROM SNIA_FSQuotaIndication WHERE WhichLimit = 3	Mandatory	Soft quota threshold crossed

11.7.1 SNIA_FSDomainIdentity

Created By: CreateInstance_or_Static_or_External
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 166 describes class SNIA_FSDomainIdentity.

Table 166 - SMI Referenced Properties/Methods for SNIA_FSDomainIdentity

Properties	Flags	Requirement	Description & Notes
PrincipalID		Mandatory	The unique ID of a principal. This may be a UID, GID or a SID.
DomainType		Mandatory	The type of domain the Principal belongs to. Possible values are "Unknown", "Other", "Unix", and "Active Directory".
PrincipalType		Mandatory	The type of Principal represented by this identity instance. Possible values are "Unknown", "Other", "User" and "Group"

11.7.2 SNIA_FSQuotaAppliesToElement

Created By: CreateInstance
Modified By: Extrinsic_or_External
Deleted By: DeleteInstance
Requirement: Mandatory

Table 167 describes class SNIA_FSQuotaAppliesToElement.

Table 167 - SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToElement

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The managed element
Dependent		Mandatory	The quota config entry

11.7.3 SNIA_FSQuotaAppliesToPrincipal

Created By: CreateInstance
Modified By: Extrinsic_or_External
Deleted By: DeleteInstance
Requirement: Mandatory

Table 168 describes class SNIA_FSQuotaAppliesToPrincipal.

Table 168 - SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToPrincipal

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The filesystem principal
Dependent		Mandatory	The quota config entry

11.7.4 SNIA_FSQuotaAppliesToTree

Created By: CreateInstance
Modified By: Extrinsic_or_External
Deleted By: DeleteInstance
Requirement: Mandatory

Table 169 describes class SNIA_FSQuotaAppliesToTree.

Table 169 - SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToTree

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The filesystem directory tree
Dependent		Mandatory	The quota config entry

11.7.5 SNIA_FSQuotaCapabilities

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 170 describes class SNIA_FSQuotaCapabilities.

Table 170 - SMI Referenced Properties/Methods for SNIA_FSQuotaCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	A unique ID for the capabilities instance.
ElementName		Mandatory	A user-friendly name for the instance in the format SystemName:ManagedElementName:Capabilities.
SupportedTargetTypes		Mandatory	The target types supported by the Service. Possible values are "ComputerSystem" and "FileSystem"
SupportedPrincipalTypes		Mandatory	An array of the types of Principal supported by the Service. Possible values are "User", "Group", "User-tree", "Group-tree" and "Tree".

Table 170 - SMI Referenced Properties/Methods for SNIA_FSQuotaCapabilities

Properties	Flags	Requirement	Description & Notes
ConflictingEntriesUsage		Mandatory	The behavior of the system when it encounters quota entries with duplicate keys
SupportedResourceTypes		Mandatory	An array of resource types that may have quotas placed on them by this Service. Possible values are"Unknown", "Other", "Bytes", "Files", "Directories", "Files+Directories", "Inodes" and "Blocks"
DefaultSupported		Mandatory	An array that indicates which resource types may have default quotas set upon them by this Service. Possible values are the same as for SupportedResourceTypes
IsActiveSettingPerEntrySupported		Mandatory	Indicates whether quotas may be made active or inactive per entry
IsMonitoredSettingPerEntrySupported		Mandatory	Indicates whether quota monitoring may be turned on or off per entry
IsGracePeriodSupported		Mandatory	Indicates whether a grace period may be set on a quota. If it can, then crossing over a soft threshold for more then the period of time specified in the grace period effectively converts the soft threshold to a hard limit, cutting off further allocation of the resource.

11.7.6 SNIA_FSQuotaConfigEntry

Created By: Extrinsic_or_External
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 171 describes class SNIA_FSQuotaConfigEntry.

Table 171 - SMI Referenced Properties/Methods for SNIA_FSQuotaConfigEntry

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	A unique ID for the entry.
HardLimit		Mandatory	The hard limit for this quota
SoftLimit		Mandatory	The soft limit for this quota
SoftLimitGracePeriod		Optional	Length of the grace period, if any exists.
Active		Optional	Whether or not this quota is to be actively monitored. If NULL, the system does not support activation of individual quotas.
Monitor		Mandatory	Whether or not this is a monitor-only quota. If this is TRUE, no enforcement of any kind is done.

Table 171 - SMI Referenced Properties/Methods for SNIA_FSQuotaConfigEntry

Properties	Flags	Requirement	Description & Notes
ResourceType		Mandatory	The type of resource being managed
QuotaType		Mandatory	The type of quota to create (user, group, etc.)
TreeName		Mandatory	Path of the tree over which this quota is applicable, if any.
PrincipalID		Mandatory	The user or group being constrained by this quota, if any.
FileSystem		Mandatory	A The name of the FileSystem, if any, over which this quota is monitored.
Default		Mandatory	Whether or not this is a default quota.

11.7.7 SNIA_FSQuotaIndication

Created By: External
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 172 describes class SNIA_FSQuotaIndication.

Table 172 - SMI Referenced Properties/Methods for SNIA_FSQuotaIndication

Properties	Flags	Requirement	Description & Notes
IdentityID		Mandatory	The InstanceID of the FSDomainIdentity involved in causing the event. If there is none, NULL shall be passed in this property.
EntryID		Mandatory	The InstanceID of the FSQuotaConfigEntry involved in causing the event..
Path		Mandatory	The complete path of the tree involved in causing the event. If there is none, NULL shall be passed in this property.
WhichLimit		Mandatory	Either "hard" or "soft"
ResourceType		Mandatory	"Bytes", "Files", "Directories","Files+Directories" or "Inodes"
QuotaType		Mandatory	Either "user", "group" or "tree".
Limit		Mandatory	The limit set by the quota entry
AmountUsed		Optional	Amount of resource actually used at the time the indication was generated.
FileSystem		Mandatory	The Name of the FileSystem in which the event occurred.

11.7.8 SNIA_FSQuotaManagementService

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 173 describes class SNIA_FSQuotaManagementService.

Table 173 - SMI Referenced Properties/Methods for SNIA_FSQuotaManagementService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
Name		Mandatory	
FindQuotaEntries()		Mandatory	Retrieve one or more quota entries based on a set of input criteria.
DeleteQuotaEntry()		Mandatory	Delete a specified quota entry
ModifyQuotaEntry()		Mandatory	Modify a specified quota entry
AddQuotaEntry()		Mandatory	Add a new quota entry.
GetQuotaReport()		Mandatory	Obtain a report on usage against the quota entries on a system.
EnableQuotas()		Mandatory	Turn quota monitoring on or off.
InitializeQuotas()		Mandatory	Initialize the quota reporting system.

11.7.9 SNIA_FSQuotaReportRecord

Created By: Extrinsic
Modified By: Static
Deleted By: Static
Requirement: Mandatory

Table 174 describes class SNIA_FSQuotaReportRecord.

Table 174 - SMI Referenced Properties/Methods for SNIA_FSQuotaReportRecord

Properties	Flags	Requirement	Description & Notes
HardLimit		Optional	The hard threshold associated with this quota report record, if any
SoftLimit		Optional	The soft threshold associated with this quota report record, if any

Table 174 - SMI Referenced Properties/Methods for SNIA_FSQuotaReportRecord

Properties	Flags	Requirement	Description & Notes
SoftLimitGracePeriod		Optional	The grace period associated with the soft limit associated with this report record, if any
Active		Optional	Whether the quota associated with this report record is being actively enforced. If not, this indicates the quota is being used for tracking purposes only.
Monitored		Optional	Whether or not thresholds on this quota are being monitored. If a system reports quotas that aren't being monitored, this value may be false.
ResourceType		Mandatory	The type of resource whose use is counted in this quota report record
QuotaType		Mandatory	The type of Principal to which this quota applies. Possible values are "Unknown", "Other", "User", "Group" and "Tree".
AmountUsed		Mandatory	The amount of resource used by the combination of Principal, Resource type, Tree, and ManagedElement specified in the quota configuration entry that generated this quota report record (and reported in other fields in the record).
TreeName		Optional	The URI of the filesystem tree upon which the quota was set, if any
PrincipalID		Optional	The FSDomainIdentity for the Principal associated with this quota report record, if any
FileSystem		Optional	The name of the filesystem over which the quota entry that generated the report record was placed, if any

11.7.10 SNIA_ReportRecord

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Mandatory

EXPERIMENTAL

STABLE

Clause 12: NAS Head Profile

12.1 Description

12.1.1 Synopsis

Profile Name: NAS Head

Version: 1.3.0

Organization: SNIA

CIM schema version: 2.15

Central Class: ComputerSystem

Scoping Class: ComputerSystem

12.1.2 Overview

The NAS Head Profile exports File elements (contained in a FileSystem) as FileShares. The storage for the FileSystem is obtained from external SAN storage, for example, a Storage Array that exports Storage Volumes as LUNs. The storage array may also provide storage to other hosts or devices (or other NAS Heads), and the storage on the array might be visible to other external management tools, and may be actively managed independently.

This profile models the necessary Filesystem and NAS concepts and defines how the connections to the underlying storage is managed. A NAS Head looks like a "host system" on a SAN and gains access to the storage through zoning (see the Fabric Book) and LUN provisioning and masking and mapping functions in the arrays it is using (see the *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*).

The NAS Head Profile reuses a significant portion of Clause 23: Storage Virtualizer Profile in *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*.

The NAS Head Profile and its subprofiles and packages are illustrated in Figure 17.

.

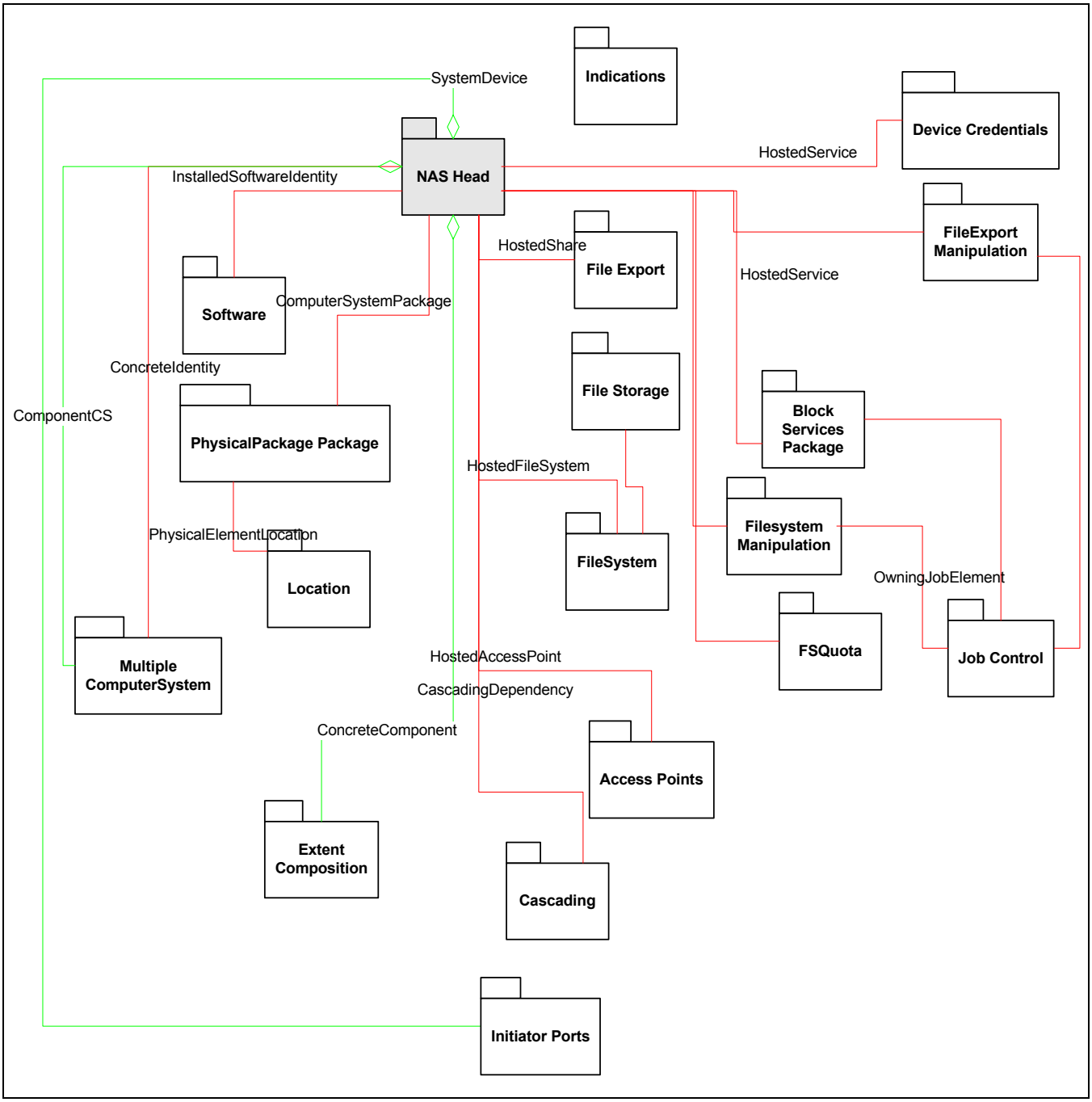

Figure 17 - NAS Head Profiles and Subprofiles

12.1.3 Implementation

12.1.3.1 Summary Instance Diagram

Figure 18 illustrates all the classes that are mandatory for the NAS Head Profile. Later diagrams will review specific sections of this diagram.

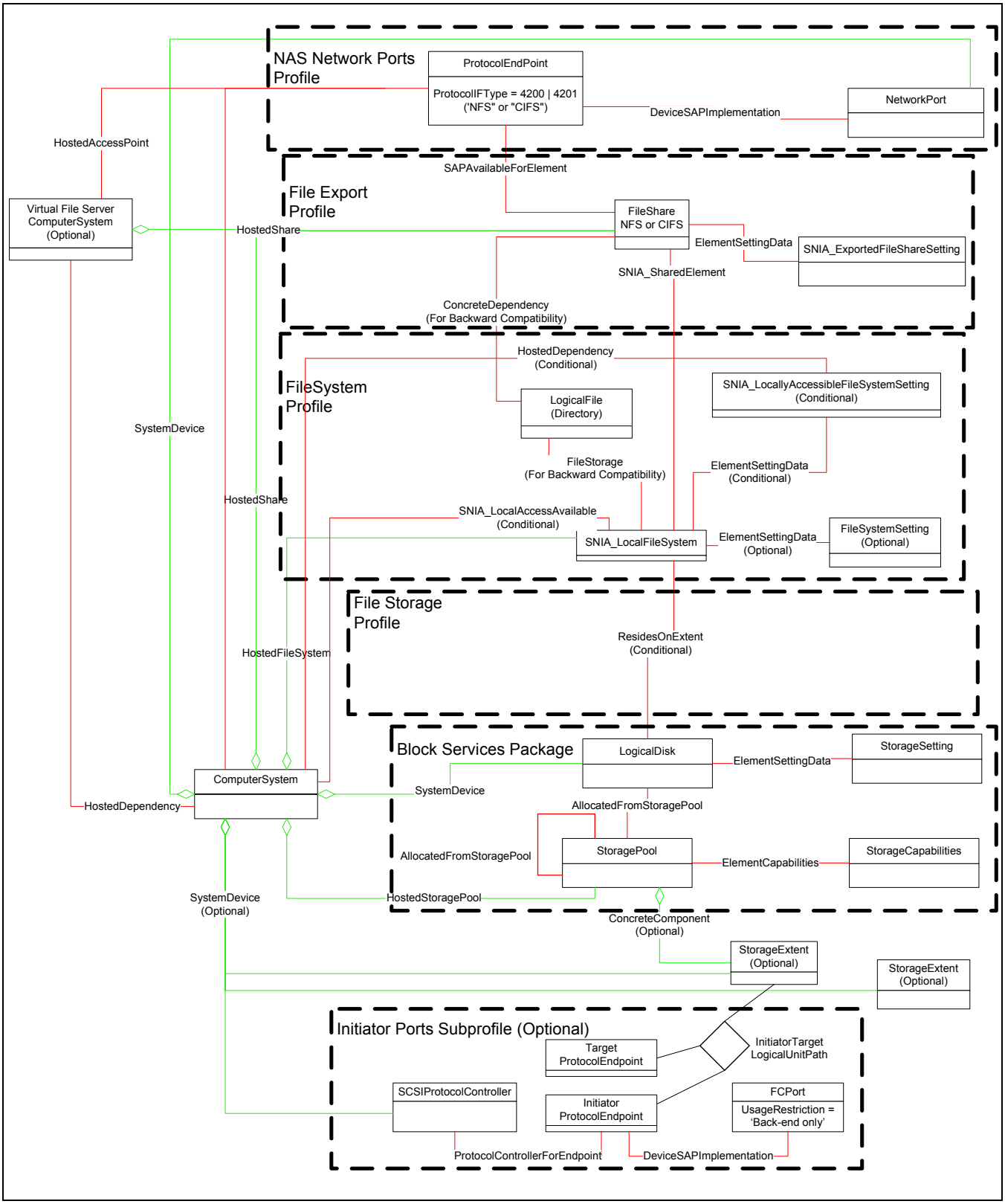

Figure 18 - NAS Head Instance

The NAS Head Profile closely parallels the Storage Virtualizer Profile in how it models storage. Imported storage may be assigned to a StoragePool (the StorageExtent with a ConcreteComponent to the primordial storage pool) and some are available for possible assignment to a primordial StoragePool (Extents without a ConcreteComponent to a storage pool). Storage is assigned to StoragePools and LogicalDisks are allocated from those storage pools for the purpose of holding local filesystems of the NAS.

As with the Storage Virtualizer Profile, the NAS Head StoragePools have StorageCapabilities associated to the StoragePools via ElementCapabilities. Similarly, LogicalDisks that are allocated from those StoragePools have StorageSettings, which are associated to the LogicalDisk via ElementSettingData. StoragePools are hosted by a ComputerSystem that represents the NAS “top level” system, and the StorageExtents have a SystemDevice association to the “top level” ComputerSystem.

Note: As with Self-Contained NAS, the StoragePools may be hosted by a component ComputerSystem if the Profile has implemented the Multiple Computer System Subprofile.

As with the Storage Virtualizer Profile, the “top level” ComputerSystem of the NAS Head does not (and typically isn’t) a real ComputerSystem. It is merely the ManagedElement upon which all aspects of the NAS offering are scoped.

A NAS Head may implement "Virtual File Servers" in addition to, or instead of, implementing File Servers in the Top Level ComputerSystem or one of the Multiple Computer System ComputerSystems. A Virtual File Server shall have a HostedDependency to either the top level NAS ComputerSystem or one of the Multiple Computer System ComputerSystems. NOTE: A Virtual File Server shall not have a ComponentCS association to the top level NAS ComputerSystem.

As with the Storage Virtualizer Profile, the NAS Head draws its storage from an open SAN. That is, the actual disk storage is addressable independent of the NAS Head. However, unlike the Storage Virtualizer, modeling of the imported storage is optional in the NAS Head. The NAS head may model the Initiator ports and the StorageExtents that it acquires from the SAN. The NAS Head may support at least one of the Initiator Ports Subprofiles (the dashed box at the bottom of Figure 18) to effect the support for backend ports. The NAS Head includes the Block Services Package to effect the logical storage management (the dashed box just above the Initiator Ports dashed box in Figure 18).

Everything above the LogicalDisk is specific to NAS (and does not appear in the Storage Virtualizer Profile). LocalFileSystems are created on the LogicalDisks, LogicalFiles within those LocalFileSystems are shared (FileShare) through ProtocolEndpoints associated with NetworkPorts.

Note: The classes and associations in the dashed boxes are from the required packages and subprofiles (as indicated by the labels on the dashed boxes).

The ConcreteDependency association is provided for backward compatibility with SMI-S 1.1.0. It represents a relationship between a FileShare and a Directory.

In the NAS Head a LocalFileSystem shall map to a LogicalDisk using the ResidesOnExtent association.

Also note that FileSystemSetting (and the corresponding ElementSettingData) are also optional. They are only shown here to illustrate where they would show up in the model should they be implemented.

In the base NAS Head profile, the classes and associations shown in Figure 18 are automatically populated based on how the NAS Head is configured. Client modification of the configuration (including configuring storage, creating extents, local filesystems and file shares) are functions found in subprofiles of the NAS Head Profile.

12.1.3.2 NAS Storage Model

Figure 19 illustrates the classes mandatory for modeling of storage for the NAS Head Profile.

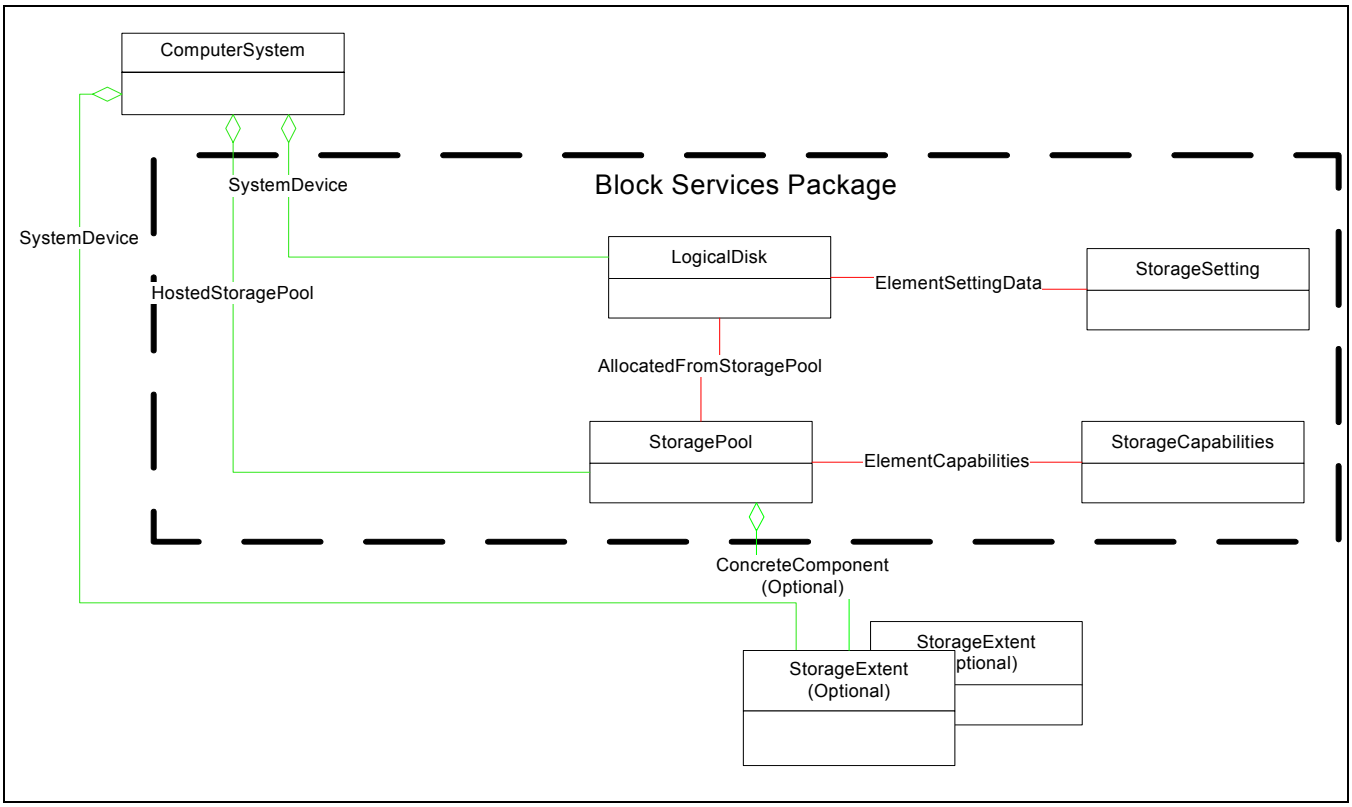

Figure 19 - NAS Storage Instance

The NAS Head Profile uses most of the classes and associations used by the Storage Virtualizer Profile (including those in the Block Services Package). In doing this, it leverages many of the subprofiles that are available for Storage Virtualizer Profiles. The classes and associations shown in Figure 19 are the minimum mandatory for read only access in the base profile.

Storage for the NAS shall be modeled as logical storage. That is, StoragePools shall be modeled, including the HostedStoragePool and ElementCapabilities to the StorageCapabilities supported by the StoragePool. In addition, for NAS Heads, which get their storage from a SAN, the StorageExtents that compose the primordial StoragePools may also be modeled with ConcreteComponent associations to the StoragePool to which they belong and they would be primordial.

In addition, in order for storage to be used it shall be allocated to one or more LogicalDisks. A LogicalDisk shall have an AllocatedFromStoragePool association to the StoragePool from which it is allocated. The LogicalDisk shall have an ElementSettingData association to the settings that were used when the LogicalDisk was created.

For manipulation of Storage, see Clause 5: Block Services Package of *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*. LogicalDisks are the ElementType that is supported for storage allocation functions (e.g., CreateOrModifyElementFromStoragePool and ReturnToStoragePool), but the Block Services methods for managing LogicalDisks are optional for the NAS Head Profile. The NAS Head Profile also supports (optionally) the Pool manipulation functions (e.g., CreateOrModifyStoragePool and DeleteStoragePool) of the Block Services Package.

12.1.3.3 NAS Head Use of Filesystem Profile (Mandatory)

The NAS Head Profile uses the Filesystem Profile for modeling of its filesystem constructs. For the NAS Head, implementation of the Filesystem Profile is mandatory. See Clause 8: Filesystem Profile for details on this modeling.

12.1.3.4 NAS Head Use of File Storage Profile (Mandatory)

The NAS Head Profile uses the File Storage Profile for modeling of its file storage constructs. For the NAS Head, implementation of the Filesystem Profile is mandatory. See Clause 7: File Storage Profile for details on the file storage modeling.

12.1.3.5 NAS Head Use of File Export Profile (Mandatory)

The NAS Head Profile uses the File Export Profile for modeling of its file export constructs. For the NAS Head, implementation of the File Export Profile is mandatory. See Clause 4: File Export Profile for details on this modeling.

12.1.3.6 NAS Head Support for Front-end Network Ports

Figure 20 illustrates the classes for modeling of front end NetworkPorts for the NAS Head Profile.

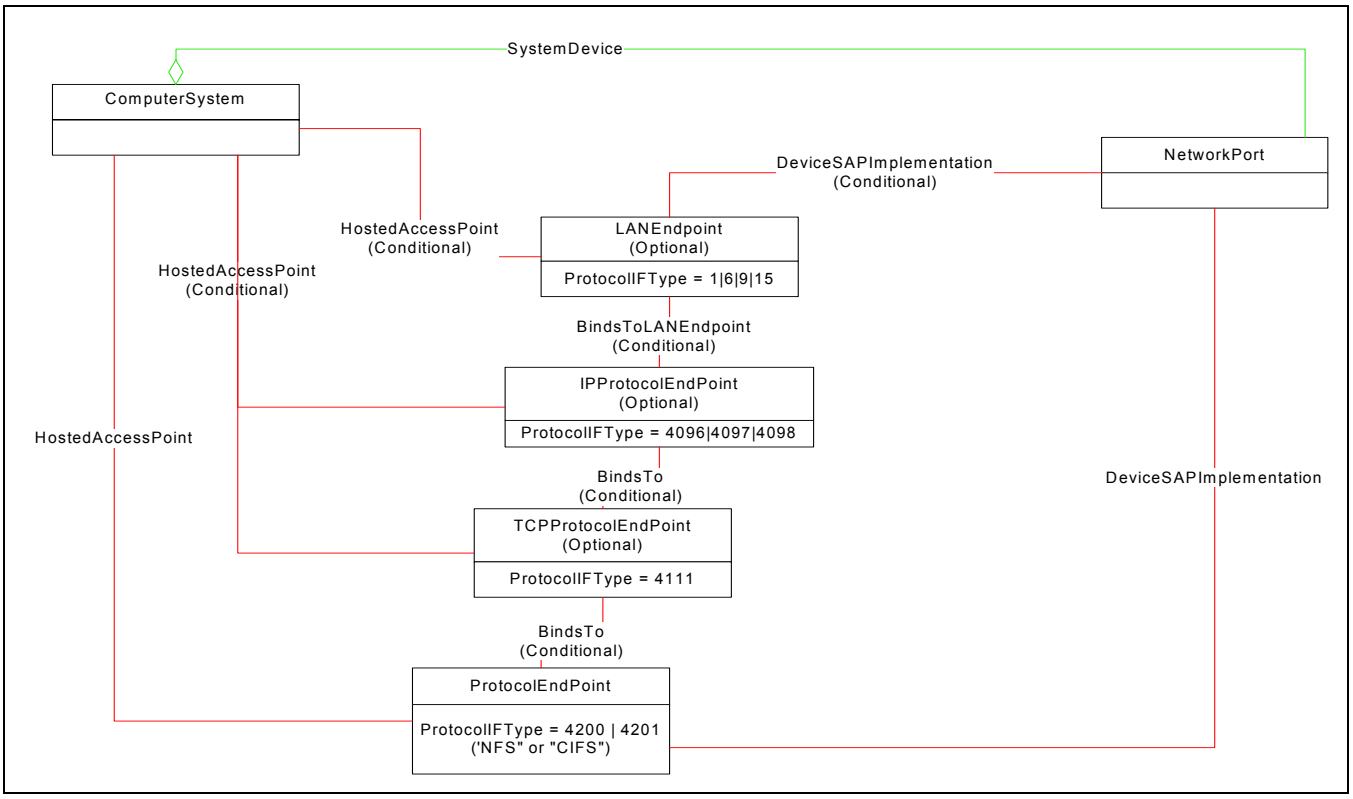

Figure 20 - NAS Head Support for Front-end Network Ports

The ProtocolEndpoint for NFS or CIFS shall be present and shall be associated to a ComputerSystem via a HostedAccessPoint association. It shall also be associated to a NetworkPort via the DeviceSAPImplementation. The NetworkPort shall be modeled and shall have an SystemDevice association to a ComputerSystem. The ComputerSystem in the diagram may be the top level system for the NAS Head or any of its component computer systems. The ComputerSystem that hosts the NFS or CIFS ProtocolEndpont need not be the same ComputerSystem associated to the NetworkPort via its SystemDevice association.

The modeling of the TCPProtocolEndpoint, IPProtocolEndpoint and the LANEndpoint are optional. The associations from (to) those classes are conditional on the existence of the classes. Like the NFS or CIFS

ProtocolEndpoint, the TCPProtocolEndpoint, IPProtocolEndpoint and LANEndpoint shall have HostedAccessPoint associations to some ComputerSystem. Typically, this would be the same ComputerSystem that hosts the NetworkPort. However this is not a requirement.

EXPERIMENTAL

12.1.3.7 NAS Head Support of Cascading

Figure 21 illustrates the NAS Head support for cascading. Support for the Cascading Subprofile is optional (and the Cascading Subprofile is experimental). It is provided here to illustrate stitching between the NAS Head and Array or Storage Virtualizer Profiles.

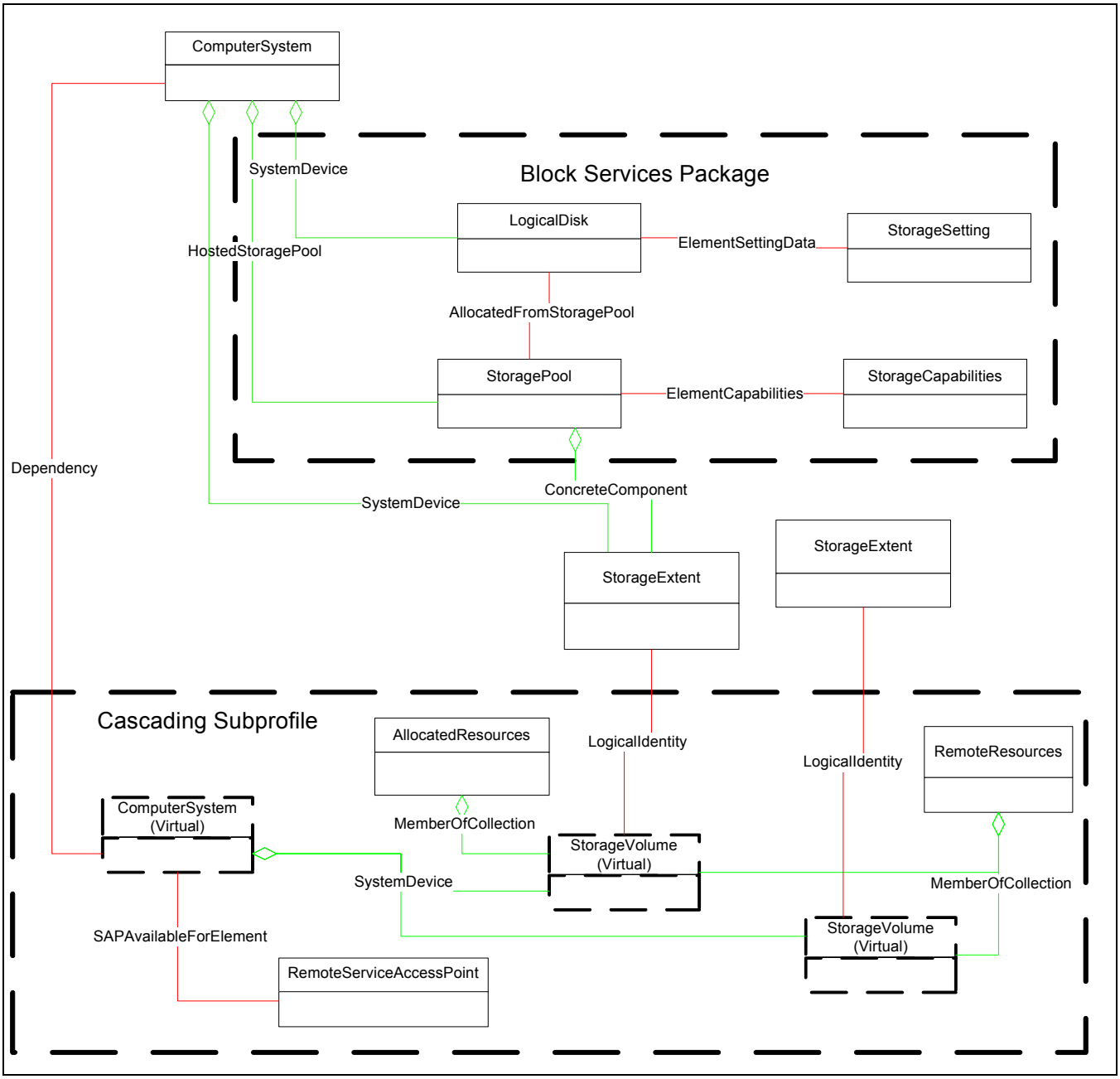

Figure 21 - NAS Head Cascading Support Instance

The lower dashed box in the figure illustrates the classes and associations of the Cascading Subprofile. The dashed classes are virtual instances (copies cached from the Array or Storage Virtualizer Profile). The other classes of the Cascading Subprofile represent NAS Head usage of those classes. For example, the collection AllocatedResources collects all the Array volumes that are used in StoragePools of the NAS Head. The RemoteResources collection collects all volumes that the NAS Head has discovered (whether used or not).

Note: Imported Storage Extents are modeled in the NAS Head for both the allocated resources and the remote resources. StorageExtents for allocated resources would have a ConcreteComponent to a Primordial StoragePool, but StorageExtents that are not members of the AllocatedResources would not have the ConcreteComponent association to any StoragePool.

The RemoteServiceAccessPoint is the URL of the management interface that the NAS Head uses for managing the Array or Storage Virtualizer Profiles.

EXPERIMENTAL

12.2 Health and Fault Management Considerations

The NAS Head supports state information (e.g., OperationalStatus) on the following elements of the model:

- Network Ports (See 12.2.1)
- Back-end Ports (See 17.3.3 Health and Fault Management Considerations in *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*)
- ComputerSystems (See 25.1.5 Computer System Operational Status in *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*)
- FileShares that are exported (See 4.2.1)
- LocalFileSystems (See 8.2.1)
- ProtocolEndpoints (See 12.2.2)

12.2.1 OperationalStatus for Network Ports

Table 175 - NetworkPort OperationalStatus

OperationalStatus	Description
OK	Port is online
Error	Port has a failure
Stopped	Port is disabled
InService	Port is in Self Test
Unknown	

12.2.2 OperationalStatus for ProtocolEndpoints

Table 176 - ProtocolEndpoint OperationalStatus

OperationalStatus	Description
OK	ProtocolEndpoint is online
Error	ProtocolEndpoint has a failure
Stopped	ProtocolEndpoint is disabled
Unknown	

EXPERIMENTAL

12.3 Cascading Considerations

The NAS Head is a cascading Profile, but the Cascading Subprofile is Experimental in this release of SMI-S; see Clause 24: Cascading Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*. As such, the Cascading Subprofile is defined as an optional subprofile. A NAS Head may cascade storage. The cascading considerations for this are discussed in the following sections.

12.3.1 Cascading Resources for the NAS Head Profile

By definition, a NAS Head gets its storage from the network. As such, there is a cascading relationship between the NAS Head Profile and the Profiles (e.g., Array Profiles) that provide the storage for the NAS Head. Figure 21 illustrates the constructs to be used to model this cascading relationship.

- The NAS Head Cascaded Resources are Primordial StorageExtents (used to populate Primordial StoragePools)
  - The NAS Head obtains the storage for these from Array or Storage Virtualizer Profiles
  - Each Primordial StorageExtent maps (via LogicalIdentity) to a StorageVolume (from the Array or Storage Virtualizer Profile).

12.3.2 Ownership Privileges Asserted by NAS Heads

In support of the Cascading NAS Heads may assert ownership over the StorageVolumes that they import. If the Array or Storage Virtualizer implementation supports Ownership, NAS Heads would assert ownership using the following Privilege:

- Activity - Execute
- ActivityQualifier - CreateOrModifyFromStoragePool and ReturnToStoragePool
- FormatQualifier - Method

12.3.3 NAS Head Limitations on use of the Cascading Subprofile

The NAS Head support for Cascading places the following limitations and restrictions on the Cascading Subprofile:

- The AllocationService is not supported. - Allocation is done as a side effect of assigning the extents to the Primordial pool.

- CascadingDependency - The CascadingDependency may exist, even when there are no resources that are imported. This signifies that the NAS Head has discovered the Array or Virtualizer, but has no access to any of their volumes.

EXPERIMENTAL

12.4 Supported Subprofiles and Packages

Table 177 describes the supported profiles for NAS Head.

Table 177 - Supported Profiles for NAS Head

Registered Profile Names	Mandatory	Version
Indication	Yes	1.3.0
Filesystem	Yes	1.3.0
File Storage	Yes	1.3.0
File Export	Yes	1.3.0
Cascading	No	1.3.0
Access Points	No	1.3.0
Multiple Computer System	No	1.2.0
Software	No	1.3.0
Location	No	1.3.0
Extent Composition	No	1.2.0
Filesystem Manipulation	No	1.3.0
File Export Manipulation	No	1.3.0
File Server Manipulation	No	1.3.0
Filesystem Performance	No	1.3.0
FileSystem Quotas	No	1.3.0
Job Control	No	1.3.0
SPI Initiator Ports	No	1.2.0
FC Initiator Ports	No	1.3.0
Device Credentials	No	1.3.0
Physical Package	Yes	1.3.0
Block Services	Yes	1.3.0
Health	Yes	1.2.0

12.5 Methods of the Profile

12.5.1 Extrinsic Methods of the Profile

None.

12.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

Manipulation functions are supported in subprofiles of the profile.

12.6 Client Considerations and Recipes

Not defined in this version of the specification.

12.7 Registered Name and Version

NAS Head version 1.3.0

12.8 CIM Elements

Table 178 describes the CIM elements for NAS Head.

Table 178 - CIM Elements for NAS Head

Element Name	Requirement	Description
12.8.1 CIM_BindsTo (CIFS or NFS)	Conditional	Conditional requirement: This is required if a TCPProtocolEndpoint exists.Associates a CIFS or NFS ProtocolEndpoint to an underlying TCPProtocolEndpoint. This is used in the NAS Head to support the TCP/IP Network protocol stack.
12.8.2 CIM_BindsTo (TCP)	Conditional	Conditional requirement: This is required if an IPProtocolEndpoint exists.Associates a TCPProtocolEndpoint to an underlying IPProtocolEndpoint. This is used in the NAS Head to support the TCP/IP Network protocol stack.
12.8.3 CIM_BindsToLANEndpoint	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Associates an IPProtocolEndpoint to an underlying LANEndpoint in the NAS Head (to support the TCP/IP Network protocol stack).
12.8.4 CIM_ComputerSystem (Top Level)	Mandatory	This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the NAS Head.
12.8.5 CIM_ComputerSystem (Virtual File Server)	Optional	This represents a Virtual File Server, if one exists.
12.8.6 CIM_ConcreteComponent	Optional	Represents the association between a Primordial StoragePool and the underlying StorageExtents that compose it.
12.8.7 CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)	Mandatory	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
12.8.8 CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
12.8.9 CIM_HostedAccessPoint (CIFS or NFS)	Mandatory	(CIFS or NFS) Represents the association between a CIFS or NFS front end ProtocolEndpoint and the Computer System that hosts it.

Table 178 - CIM Elements for NAS Head

Element Name	Requirement	Description
12.8.10 CIM_HostedAccessPoint (IP)	Conditional	Conditional requirement: This is required if an IPProtocolEndpoint exists.Represents the association between a front end IPProtocolEndpoint and the Computer System that hosts it.
12.8.11 CIM_HostedAccessPoint (LAN)	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Represents the association between a front end LANEndpoint and the Computer System that hosts it.
12.8.12 CIM_HostedAccessPoint (TCP)	Conditional	Conditional requirement: This is required if a TCPProtocolEndpoint exists.Represents the association between a front end TCPProtocolEndpoint and the Computer System that hosts it.
12.8.13 CIM_HostedDependency	Optional	Associates a Virtual File Server to the Computer System hosting it. This is required if a Virtual File Server exists.
12.8.14 CIM_IPProtocolEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
12.8.15 CIM_LANEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
12.8.16 CIM_LogicalDisk (LD for FS)	Mandatory	Represents the single Storage Extent on which the NAS Head will build a LocalFileSystem.
12.8.17 CIM_NetworkPort	Mandatory	Represents the front end logical port that supports access to a local area network.
12.8.18 CIM_ProtocolEndpoint (CIFS or NFS)	Mandatory	(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
12.8.19 CIM_StorageExtent (Primordial Imported Extent)	Optional	This StorageExtent represents the LUNs (StorageVolumes) imported from a storage device to the NAS Head.
12.8.20 CIM_SystemDevice (Logical Disks)	Mandatory	This association links all LogicalDisks to the scoping system.
12.8.21 CIM_SystemDevice (Network Ports)	Mandatory	This association links all NetworkPorts to the scoping system. This is used to represent both front end and back end ports.
12.8.22 CIM_SystemDevice (Storage Extents)	Conditional	Conditional requirement: This is required if primordial StorageExtents exist.This association links all StorageExtents to the scoping system.

Table 178 - CIM Elements for NAS Head

Element Name	Requirement	Description
12.8.23 CIM_TCPProtocolEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support TCP services.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	Optional	CQL -Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="SNIA" and MessageID="FSM001"	Optional	CQL -This is a bellwether indication of a change of Status of a NAS ComputerSystem (controller) and related classes (LogicalDisks, Services, ProtocolEndpoints, StoragePools, FileShares and FileSystems).
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.CIM_NetworkPort::OperationalStatus <> PreviousInstance.CIM_NetworkPort::OperationalStatus	Optional	CQL -Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="SNIA" and MessageID="FSM002"	Optional	CQL -This is a bellwether indication of a change of Status of a Port and related classes (ProtocolEndpoints and FileShares).
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.CIM_ProtocolEndpoint::OperationalStatus <> PreviousInstance.CIM_ProtocolEndpoint::OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Optional	CQL -Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.

Table 178 - CIM Elements for NAS Head

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	CQL -Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="SNIA" and MessageID="FSM003"	Optional	CQL -This is a bellwether indication of a change of status of a LogicalDisk.

12.8.1 CIM_BindsTo (CIFS or NFS)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a TCPProtocolEndpoint exists.

Table 179 describes class CIM_BindsTo (CIFS or NFS).

Table 179 - SMI Referenced Properties/Methods for CIM_BindsTo (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ProtocolEndpoint that uses a lower level ProtocolEndpoint for connectivity.
Antecedent		Mandatory	The TCPProtocolEndpoint that supports a CIFS or NFS ProtocolEndpoint.

12.8.2 CIM_BindsTo (TCP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if an IPProtocolEndpoint exists.

Table 180 describes class CIM_BindsTo (TCP).

Table 180 - SMI Referenced Properties/Methods for CIM_BindsTo (TCP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The TCPProtocolEndpoint that uses an IPProtocolEndpoint for connectivity.
Antecedent		Mandatory	The IPProtocolEndpoint that supports a TCPProtocolEndpoint.

12.8.3 CIM_BindsToLANEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 181 describes class CIM_BindsToLANEndpoint.

Table 181 - SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint

Properties	Flags	Requirement	Description & Notes
FrameType		Mandatory	Only supports 1="Ethernet" at this point.
Dependent		Mandatory	An IPProtocolEndpoint.
Antecedent		Mandatory	A LANEndpoint.

12.8.4 CIM_ComputerSystem (Top Level)

Created By: Static
Modified By: External
Deleted By: Static
Requirement: Mandatory

Table 182 describes class CIM_ComputerSystem (Top Level).

Table 182 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	The actual class of this object, e.g., Vendor_NASComputerSystem.
ElementName		Mandatory	User friendly name
Name		Mandatory	Unique identifier for the NAS Head in a format specified by NameFormat. For example, IP address or Vendor/Model/SerialNo.

Table 182 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Properties	Flags	Requirement	Description & Notes
OperationalStatus		Mandatory	Overall status of the NAS Head
NameFormat		Mandatory	Format for Name property.
PrimaryOwnerContact	M	Optional	Owner of the NAS Head
PrimaryOwnerName	M	Optional	Contact details for owner
Dedicated		Mandatory	This shall be a NAS Head (24).
OtherIdentifyingInfo	C	Mandatory	An array of know identifiers for the NAS Head.
IdentifyingDescriptions	C	Mandatory	An array of descriptions of the OtherIdentifyingInfo. Some of the descriptions would be "Ipv4 Address", "Ipv6 Address" or "Fully Qualified Domain Name".
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
Roles	N	Optional	Not Specified in this version of the Profile.
OtherDedicatedDescriptions	N	Optional	Not Specified in this version of the Profile.
ResetCapability	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.

12.8.5 CIM_ComputerSystem (Virtual File Server)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 183 describes class CIM_ComputerSystem (Virtual File Server).

Table 183 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Virtual File Server)

Properties	Flags	Requirement	Description & Notes
Dedicated		Mandatory	A Virtual File Server is a File Server (16).
NameFormat		Mandatory	Format for Name property. This shall be "Other".
Name	C	Mandatory	Unique identifier for the NAS Head's Virtual File Servers (Eg Vendor/Model/SerialNo+FS+Number).
OperationalStatus		Mandatory	Overall status of the Virtual File Server.
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
ElementName		Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
Roles	N	Optional	Not Specified in this version of the Profile.
OtherDedicatedDescriptions	N	Optional	Not Specified in this version of the Profile.
ResetCapability	N	Optional	Not Specified in this version of the Profile.
PrimaryOwnerContact	N	Optional	Not Specified in this version of the Profile.
PrimaryOwnerName	N	Optional	Not Specified in this version of the Profile.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.

12.8.6 CIM_ConcreteComponent

Created By: External

Modified By: Static
Deleted By: External
Requirement: Optional

Table 184 describes class CIM_ConcreteComponent.

Table 184 - SMI Referenced Properties/Methods for CIM_ConcreteComponent

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Primordial StoragePool that is built from the StorageExtent.
PartComponent		Mandatory	A StorageExtent that is part of a Primordial StoragePool.

12.8.7 CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 185 describes class CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort).

Table 185 - SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ProtocolEndpont that supports on the NetworkPort. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		Mandatory	The NetworkPort supported by the Access Point.

12.8.8 CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 186 describes class CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort).

Table 186 - SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	A LANEndpoint that depends on a NetworkPort for connecting to its LAN segment.
Antecedent		Mandatory	The Logical network adapter device that connects to a LAN.

12.8.9 CIM_HostedAccessPoint (CIFS or NFS)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 187 describes class CIM_HostedAccessPoint (CIFS or NFS).

Table 187 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ServiceAccessPoint hosted on the file server. These include ProtocolEndpoints for NFS or CIFS.
Antecedent		Mandatory	The Computer System hosting this Access Point. In the context of the NAS Head, these are always file servers (Dedicated=16).

12.8.10 CIM_HostedAccessPoint (IP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if an IPProtocolEndpoint exists.

Table 188 describes class CIM_HostedAccessPoint (IP).

Table 188 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (IP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The IPProtocolEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

12.8.11 CIM_HostedAccessPoint (LAN)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 189 describes class CIM_HostedAccessPoint (LAN).

Table 189 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (LAN)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The LANEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

12.8.12 CIM_HostedAccessPoint (TCP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a TCPProtocolEndpoint exists.

Table 190 describes class CIM_HostedAccessPoint (TCP).

Table 190 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The TCPProtocolEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

12.8.13 CIM_HostedDependency

Created By: External
Modified By: Static
Deleted By: External
Requirement: Optional

Table 191 describes class CIM_HostedDependency.

Table 191 - SMI Referenced Properties/Methods for CIM_HostedDependency

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The Virtual File Server ComputerSystem.
Antecedent		Mandatory	The hosting ComputerSystem.

12.8.14 CIM_IPProtocolEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 192 describes class CIM_IPProtocolEndpoint.

Table 192 - SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the Protocol Endpoint.
Name		Mandatory	The unique name of the Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name.
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	
Description		Mandatory	This shall be the IP protocol endpoints supported by the NAS Head.
ProtocolIFType		Mandatory	4096="IP v4", 4097="IP v6", and 4098 is both. (Note that 1="Other" is not supported)
IPv4Address		Conditional	Conditional requirement: This is required if ProtocolIFType = 4096 or 4098.An IP v4 address in the format "A.B.C.D".
IPv6Address		Conditional	Conditional requirement: This is required if ProtocolIFType = 4097 or 4098.An IP v6 address.

Table 192 - SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
SubnetMask		Conditional	Conditional requirement: This is required if ProtocolIFType = 4096 or 4098.An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		Conditional	Conditional requirement: This is required if ProtocolIFType = 4097 or 4098.For an IPv6 address.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
AddressOrigin	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

12.8.15 CIM_LANEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 193 describes class CIM_LANEndpoint.

Table 193 - SMI Referenced Properties/Methods for CIM_LANEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the Protocol Endpoint.
Name		Mandatory	The unique name of the Protocol Endpoint.

Table 193 - SMI Referenced Properties/Methods for CIM_LANEndpoint

Properties	Flags	Requirement	Description & Notes
NameFormat		Mandatory	The Format of the Name.
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	
Description		Mandatory	This shall be the LAN protocol endpoints supported by the NAS Head.
ProtocolIFType		Mandatory	LAN endpoints supported are: 1="Other",6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".
OtherTypeDescription		Optional	If the LAN endpoint is a vendor-extension specified by "Other" and a description.
LANID	N	Optional	A unique id for the LAN segment to which this device is connected. The value will be NULL if the LAN is not connected.
MACAddress		Mandatory	Primary Unicast address for this LAN device.
AliasAddresses		Mandatory	Other unicast addresses supported by this device.
GroupAddresses		Mandatory	Multicast addresses supported by this device.
MaxDataSize		Mandatory	The max size of packet supported by this LAN device.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

12.8.16 CIM_LogicalDisk (LD for FS)

Created By: Extrinsic_or_External

Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 194 describes class CIM_LogicalDisk (LD for FS).

Table 194 - SMI Referenced Properties/Methods for CIM_LogicalDisk (LD for FS)

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	CIM Class of the NAS Head Computer System that is the host of this LogicalDisk.
SystemName		Mandatory	Name of the NAS Head Computer System that hosts this LogicalDisk.
CreationClassName		Mandatory	CIM Class of this instance of LogicalDisk.
DeviceID		Mandatory	Opaque identifier for the LogicalDisk.
OperationalStatus		Mandatory	A subset of operational status that is applicable for LogicalDisks in a NAS Head.
ExtentStatus		Mandatory	This LogicalDisk is neither imported (16) nor exported (17).
Primordial		Mandatory	This represents a Concrete Logical Disk that is not primordial.
Name		Mandatory	Identifier for a local LogicalDisk that will be used for a filesystem; since this logical disk will be referenced by a client, it must have a unique name. We cannot constrain the format here, but the OS-specific format described in the Block Services specification is not appropriate, so "Other" is used.
NameFormat		Mandatory	The format of the Name appropriate for LogicalDisks in the NAS Head. This shall be coded as "1" ("other").
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.

Table 194 - SMI Referenced Properties/Methods for CIM_LogicalDisk (LD for FS)

Properties	Flags	Requirement	Description & Notes
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.
LocationIndicator	N	Optional	Not Specified in this version of the Profile.
DataOrganization	N	Optional	Not Specified in this version of the Profile.
Purpose	N	Optional	Not Specified in this version of the Profile.
Access	N	Optional	Not Specified in this version of the Profile.
ErrorMethodology	N	Optional	Not Specified in this version of the Profile.
SequentialAccess	N	Optional	Not Specified in this version of the Profile.
NameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameFormat	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
Reset()		Optional	Not Specified in this version of the Profile.

12.8.17 CIM_NetworkPort

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 195 describes class CIM_NetworkPort.

Table 195 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Network adapter that provides a network port.
OperationalStatus		Mandatory	The operational status of the adapter.
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Network Port.
SystemName		Mandatory	The name of the Computer System hosting the Network Port.
CreationClassName		Mandatory	The CIM Class name of the Network Port.

Table 195 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
DeviceID		Mandatory	A unique ID for the device (in the context of the hosting System).
Speed		Optional	
MaxSpeed		Optional	
RequestedSpeed		Optional	
UsageRestriction		Optional	
PortType		Optional	
PortNumber		Optional	A unique number for the adapter in the context of the hosting System.
PermanentAddress	C	Mandatory	The hard-coded address of this port.
NetworkAddresses		Mandatory	An array of network addresses for this port.
LinkTechnology		Optional	1="Other", 2="Ethernet", 3="IB", 4="FC", 5="FDDI", 6="ATM", 7="Token Ring", 8="Frame Relay", 9="Infrared", 10="BlueTooth", 11="Wireless LAN. The link technology supported by this adapter.
OtherLinkTechnology		Optional	The vendor-specific "Other" link technology supported by this adapter.
FullDuplex		Optional	
AutoSense		Optional	
SupportedMaximumTransmissionUnit		Optional	
ActiveMaximumTransmissionUnit		Optional	
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
Name	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.

Table 195 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.
LocationIndicator	N	Optional	Not Specified in this version of the Profile.
OtherPortType	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
Reset()		Optional	Not Specified in this version of the Profile.

12.8.18 CIM_ProtocolEndpoint (CIFS or NFS)

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 196 describes class CIM_ProtocolEndpoint (CIFS or NFS).

Table 196 - SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the Protocol Endpoint.
Name		Mandatory	The unique name of the Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name.
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	
Description		Mandatory	This shall be one of the NFS or CIFS protocol endpoints supported by the NAS Head.

Table 196 - SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
ProtocolIFType		Mandatory	This represents either NFS=4200 or CIFS=4201. Other protocol types are specified in subclasses of ProtocolEndpoint.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

12.8.19 CIM_StorageExtent (Primordial Imported Extent)

Created By: Static_or_External
Modified By: External
Deleted By: External
Requirement: Optional

Table 197 describes class CIM_StorageExtent (Primordial Imported Extent).

Table 197 - SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial Imported Extent)

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CreationClassName for the scoping system.
SystemName		Mandatory	The System Name of the scoping system
CreationClassName		Mandatory	CreationClassName indicates the name of the class or the subclass.
DeviceID		Mandatory	An ID that uniquely names the StorageExtent in the NAS Head.
BlockSize		Mandatory	The size (in bytes) of blocks.

Table 197 - SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial Imported Extent)

Properties	Flags	Requirement	Description & Notes
NumberOfBlocks		Mandatory	The number of Blocks from the imported StorageVolume.
ExtentStatus		Mandatory	This shall contain ,Äò16,Äô (Imported).
OperationalStatus		Mandatory	Value shall be 2|3|6|8|15 (OK or Degraded or Error or Starting or Dormant).
Name		Mandatory	DEPRECATED: Identifier for a remote LUN on a storage array; possibly, the array ID plus LUN Node WWN.
Primordial		Mandatory	The StorageExtent imported from an Array is considered primordial in the NAS Head.
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.
LocationIndicator	N	Optional	Not Specified in this version of the Profile.
DataOrganization	N	Optional	Not Specified in this version of the Profile.
Purpose	N	Optional	Not Specified in this version of the Profile.
Access	N	Optional	Not Specified in this version of the Profile.
ErrorMethodology	N	Optional	Not Specified in this version of the Profile.
ConsumableBlocks	N	Optional	Not Specified in this version of the Profile.
IsBasedOnUnderlyingRedundancy	N	Optional	Not Specified in this version of the Profile.
SequentialAccess	N	Optional	Not Specified in this version of the Profile.

Table 197 - SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial Imported Extent)

Properties	Flags	Requirement	Description & Notes
NoSinglePointOfFailure	N	Optional	Not Specified in this version of the Profile.
DataRedundancy	N	Optional	Not Specified in this version of the Profile.
PackageRedundancy	N	Optional	Not Specified in this version of the Profile.
DeltaReservation	N	Optional	Not Specified in this version of the Profile.
NameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameFormat	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
Reset()		Optional	Not Specified in this version of the Profile.

12.8.20 CIM_SystemDevice (Logical Disks)

Created By: Extrinsic_or_External_or_Static
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 198 describes class CIM_SystemDevice (Logical Disks).

Table 198 - SMI Referenced Properties/Methods for CIM_SystemDevice (Logical Disks)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this device.
PartComponent		Mandatory	The LogicalDisk that is a part of a computer system.

12.8.21 CIM_SystemDevice (Network Ports)

Created By: Extrinsic_or_External_or_Static
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 199 describes class CIM_SystemDevice (Network Ports).

Table 199 - SMI Referenced Properties/Methods for CIM_SystemDevice (Network Ports)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this device.
PartComponent		Mandatory	The NetworkPort that is a part of a computer system.

12.8.22 CIM_SystemDevice (Storage Extents)

Created By: Extrinsic_or_External_or_Static
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: This is required if primordial StorageExtents exist.

Table 200 describes class CIM_SystemDevice (Storage Extents).

Table 200 - SMI Referenced Properties/Methods for CIM_SystemDevice (Storage Extents)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this device.
PartComponent		Mandatory	The primordial StorageExtent that is imported to a computer system in the NAS Head.

12.8.23 CIM_TCPProtocolEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 201 describes class CIM_TCPProtocolEndpoint.

Table 201 - SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the Protocol Endpoint.
Name		Mandatory	The unique name of the Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name.

Table 201 - SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	
Description		Mandatory	This shall be the TCP protocol endpoints supported by the NAS Head.
ProtocolIFType		Mandatory	4111="TCP". Note that no other protocol type is supported by this endpoint.
PortNumber		Mandatory	The number of the TCP Port that this element represents.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

STABLE

STABLE

Clause 13: Self-Contained NAS Profile

13.1 Description

13.1.1 Synopsis

Profile Name: Self-Contained NAS

Version: 1.3.0

Organization: SNIA

CIM schema version: 2.15

Central Class: ComputerSystem

Scoping Class: ComputerSystem

13.1.2 Overview

The Self-contained NAS profile exports File elements (contained in a filesystem) as FileShares. The storage for the filesystem is obtained from captive storage. In the simplest case, this could be a set of directly connected disks, but it could also be a captive storage array that is not shared with any other hosts or devices (though it could be visible to external management tools and even actively managed independently).

This profile models the necessary filesystem and NAS concepts and defines how the connections to the underlying storage is managed. The details of how a directly attached set of disks is used by the Self-contained NAS profile is covered as part of the Disk Drive or Disk Drive Lite subprofile. The details of how an underlying Storage Array might export storage to the SC NAS is not covered in this profile but is covered by Clause 4: Array Profile in *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*.

The Self-contained NAS profile reuses a significant portion of Clause 4: Array Profile in *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*.

The Self-Contained NAS Profile and its subprofiles and packages are illustrated in Figure 22.

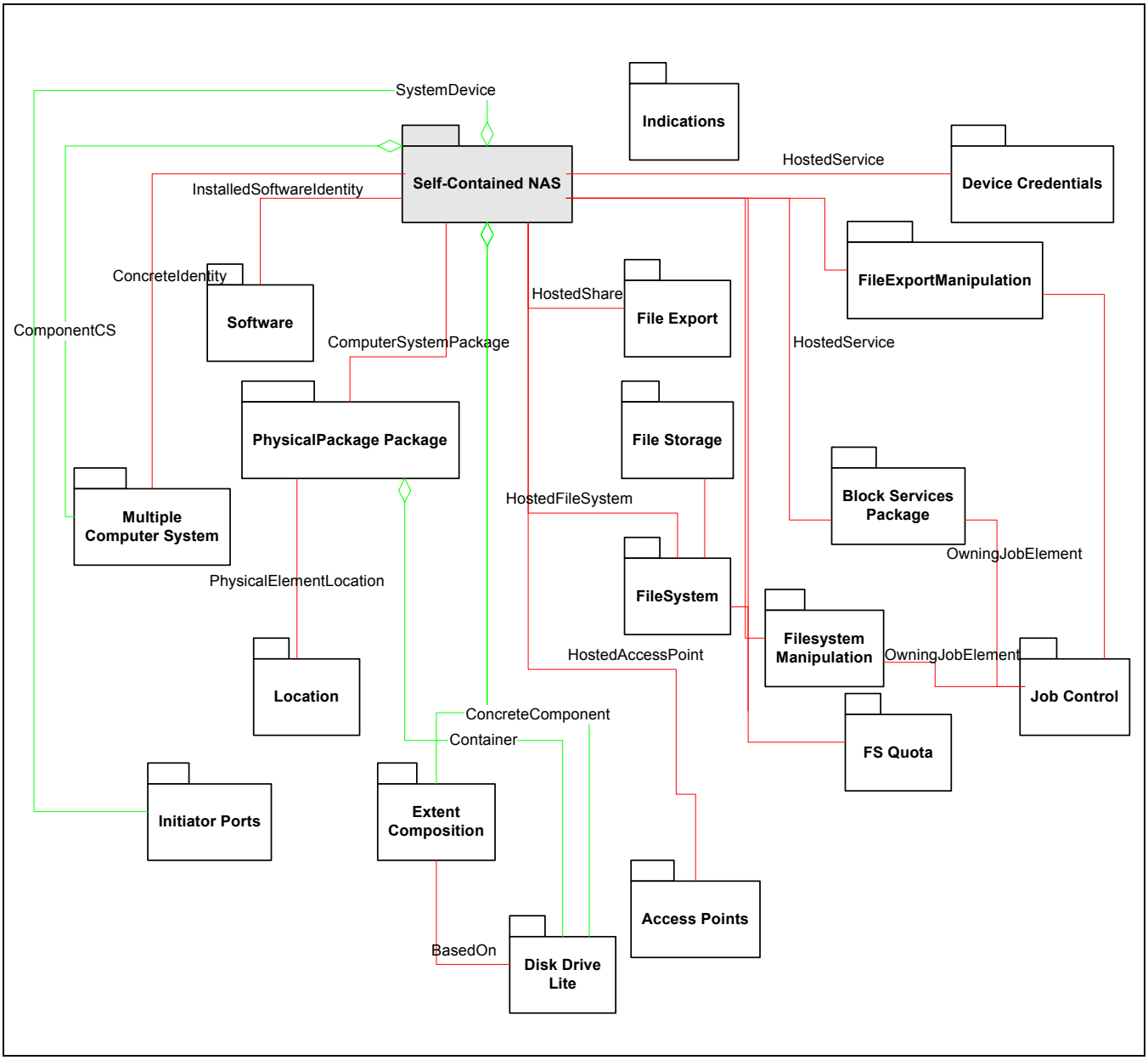

Figure 22 - Self-Contained NAS Profile and Subprofiles

13.1.3 Implementation

13.1.3.1 Summary Instance Diagram

Figure 23 illustrates all the classes that are mandatory for the Self-contained NAS Profile. Later diagrams will review specific sections of this diagram.

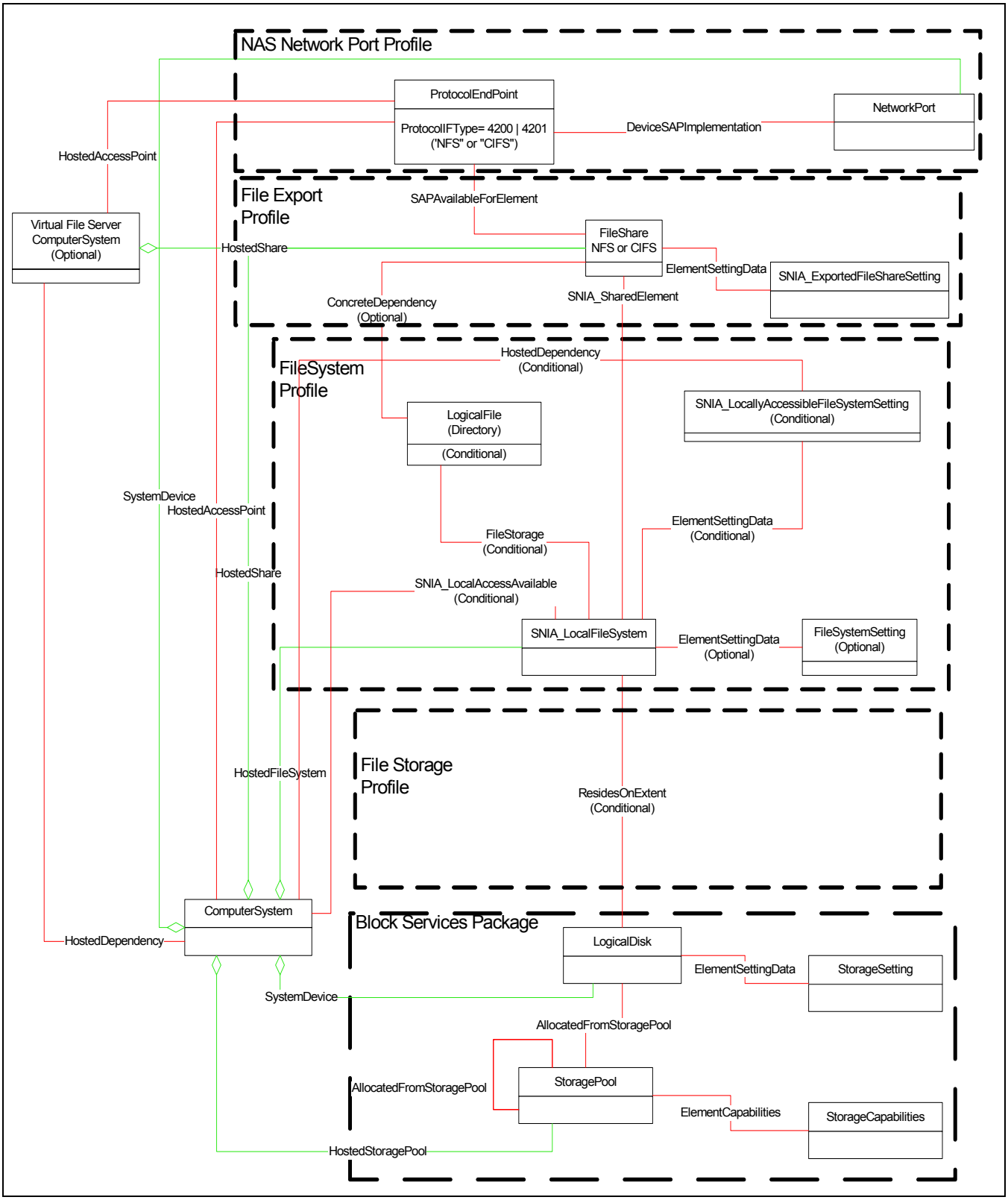

Figure 23 - Self-Contained NAS Instance

The Self-Contained NAS Profile closely parallels the Array Profile in how it models storage. Storage is assigned to StoragePools and LogicalDisks are allocated from those storage pools for the purpose of holding local filesystems of the NAS.

As with the Array profile, the Self-contained NAS StoragePools have StorageCapabilities associated to the StoragePools via ElementCapabilities. Similarly, LogicalDisks have StorageSettings, which are associated to the LogicalDisk via ElementSettingData. StoragePools are hosted by a ComputerSystem that represents the NAS “top level” system, and the LogicalDisks have a SystemDevice association to the “top level” ComputerSystem.

Note: As with Arrays, the StoragePools may be hosted by a component ComputerSystem if the profile has implemented the Multiple Computer System Subprofile.

As with Arrays, the “top level” ComputerSystem of the Self-Contained NAS does not (and typically isn’t) a real ComputerSystem. It is merely the ManagedElement upon which all aspects of the NAS offering are scoped.

A may implement "Virtual File Servers" in addition to, or instead of, implementing File Servers in the Top Level ComputerSystem or one of the Multiple Computer System ComputerSystems. A Virtual File Server shall have a HostedDependency to either the top level NAS ComputerSystem or one of the Multiple Computer System ComputerSystems. NOTE: A Virtual File Server shall not have a ComponentCS association to the top level NAS ComputerSystem.

Everything above the LogicalDisk is specific to NAS (and does not appear in the Array Profile). LocalFileSystems are created on the LogicalDisks, LogicialFiles within those LocalFileSystems are shared (FileShare) through ProtocolEndpoints associated with NetworkPorts.

Note: The classes and associations in the dashed boxes are from the required packages and subprofiles (as indicated by the labels on the dashed boxes).

The ConcreteDependency association is provided for backward compatibility with SMI-S 1.1.0. It represents a relationship between a FileShare and a Directory.

In the Self-contained NAS a LocalFileSystem shall map to a LogicalDisk using the ResidesOnExtent association.

Also note that FileSystemSetting (and the corresponding ElementSettingData) are also optional. They are only shown here to illustrate where they would show up in the model should they be implemented.

In the base Self-Contained NAS profile, the classes and associations shown in Figure 23 are automatically populated based on how the Self-Contained NAS is configured. Client modification of the configuration (including configuring storage, creating extents, local filesystems and file shares) are functions found in subprofiles of the profile.

EXPERIMENTAL

13.1.3.2 Combination Profile Considerations

Some devices combine the function of an array with the function of a Self-contained NAS. There are a number of approaches that may be used to model such a device. One way is to present two seemly independent profiles in the SAN (e.g., Array and SC NAS). In this case, there may be duplication of instances. These duplicates would be recognized by clients via correlated ids.

Another approach would be to use one, shared top level ComputerSystem that reflects both the SC NAS and the Array in its ComputerSystem.Dedicated property. In this case, care must be taken to ensure the sharing of instances between the profiles do not conflict with their respective profile definitions.

For more information on the rules for combination profiles, see section B.5 of Annex B: (Normative) Compliance with the SNIA SMI Specification in *Storage Management Technical Specification, Part 1 Common Architecture, 1.3.0 Rev 6*.

EXPERIMENTAL

13.1.3.3 NAS Storage Model

Figure 24 illustrates the classes mandatory for modeling of storage for the Self-Contained NAS Profile.

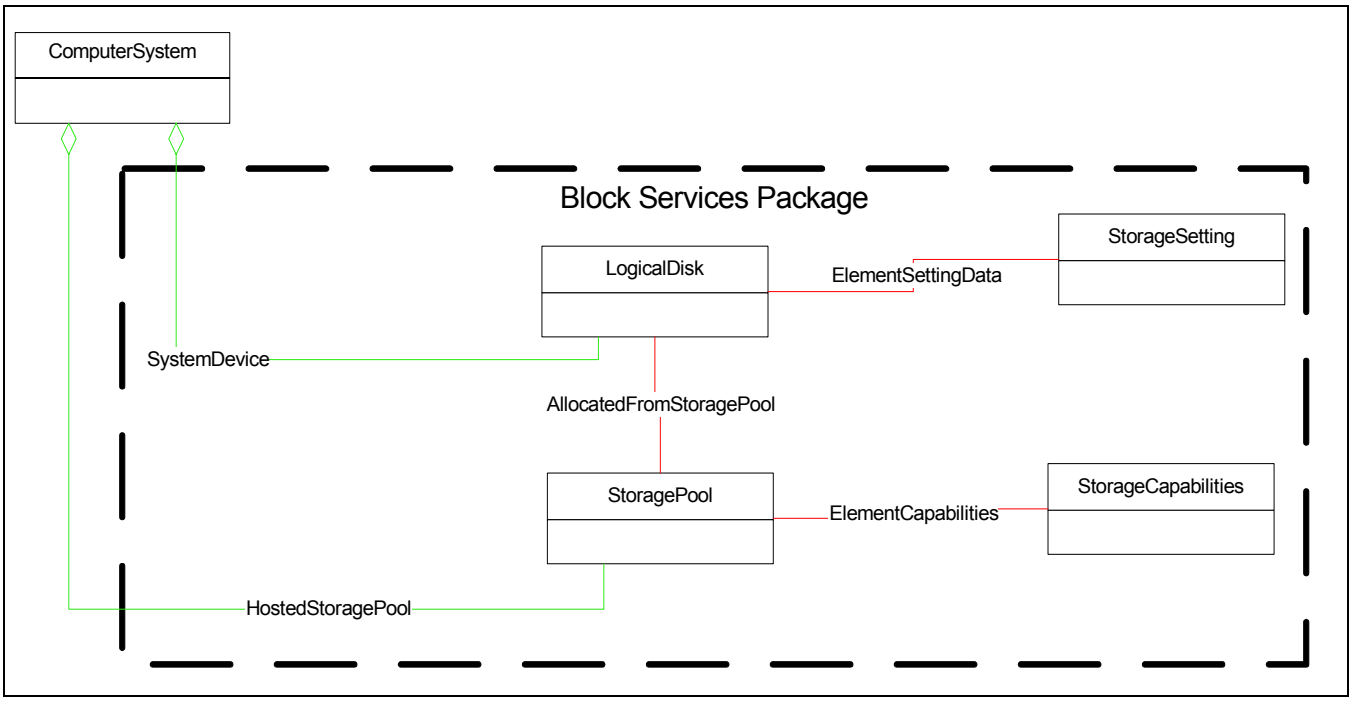

Figure 24 - NAS Storage Instance

The Self-Contained NAS Profile uses most of the classes and associations defined in the Array Profile (including those in the Block Services Package). In doing this, it leverages many of the subprofiles that are available for Array Profiles. The classes and associations shown in Figure 24 are the minimum mandatory classes and associations of the Block Services Package for read only access in the base profile.

Storage for the NAS shall be modeled as logical storage. That is, StoragePools shall be modeled, including the HostedStoragePool and ElementCapabilities to the StorageCapabilities supported by the StoragePool. In addition, in order for storage to be used it shall be allocated to one or more LogicalDisks. A LogicalDisk shall have an AllocatedFromStoragePool association to the StoragePool from which it is allocated. And the LogicalDisk shall have an ElementSettingData association to the settings that were used when the LogicalDisk was created.

Note: At this level, the model for storage is the same for both the Self-contained NAS Profile and the NAS Head Profile. In the case of the Self-contained NAS, storage for the StoragePools is drawn from Disk Drives. Modeling of Disk Drives is Optional (See Clause 11: Disk Drive Lite Subprofile of *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*).

For manipulation of Storage, see Clause 5: Block Services Package in the *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*. For Self-Contained NAS, LogicalDisks are the ElementType that is supported for storage allocation functions (e.g., CreateOrModifyElementFromStoragePool and ReturnToStoragePool), but the Block Services methods for managing LogicalDisks are optional for the Self-

Contained NAS Profile. The Self-Contained NAS Profile also supports (optionally) the Pool manipulation functions (e.g., CreateOrModifyStoragePool and DeleteStoragePool) of the Block Services Package.

13.1.3.4 Self-Contained NAS Use of Filesystem Profile (Mandatory)

The Self-Contained NAS Profile uses the Filesystem Profile for modeling of its filesystem constructs. For the Self-Contained NAS, implementation of the Filesystem Profile is mandatory. See Clause 8: Filesystem Profile for details on this modeling.

13.1.3.5 Self-Contained NAS Use of File Storage Profile (Mandatory)

The Self-Contained NAS Profile uses the File Storage Profile for modeling of its file storage constructs. For the Self-Contained NAS, implementation of the File Storage Profile is mandatory. See Clause 7: File Storage Profile for details on the file storage modeling.

13.1.3.6 Self-Contained NAS Use of File Export Profile (Mandatory)

The Self-Contained NAS Profile uses the File Export Profile for modeling of its file export constructs. For the Self-Contained NAS, implementation of the File Export Profile is mandatory. See Clause 4: File Export Profile for details on this modeling.

13.1.3.7 Self-Contained NAS Support for Front-end Network Ports

Figure 25 illustrates the classes for modeling of front end NetworkPorts for the Self-contained NAS Profile.

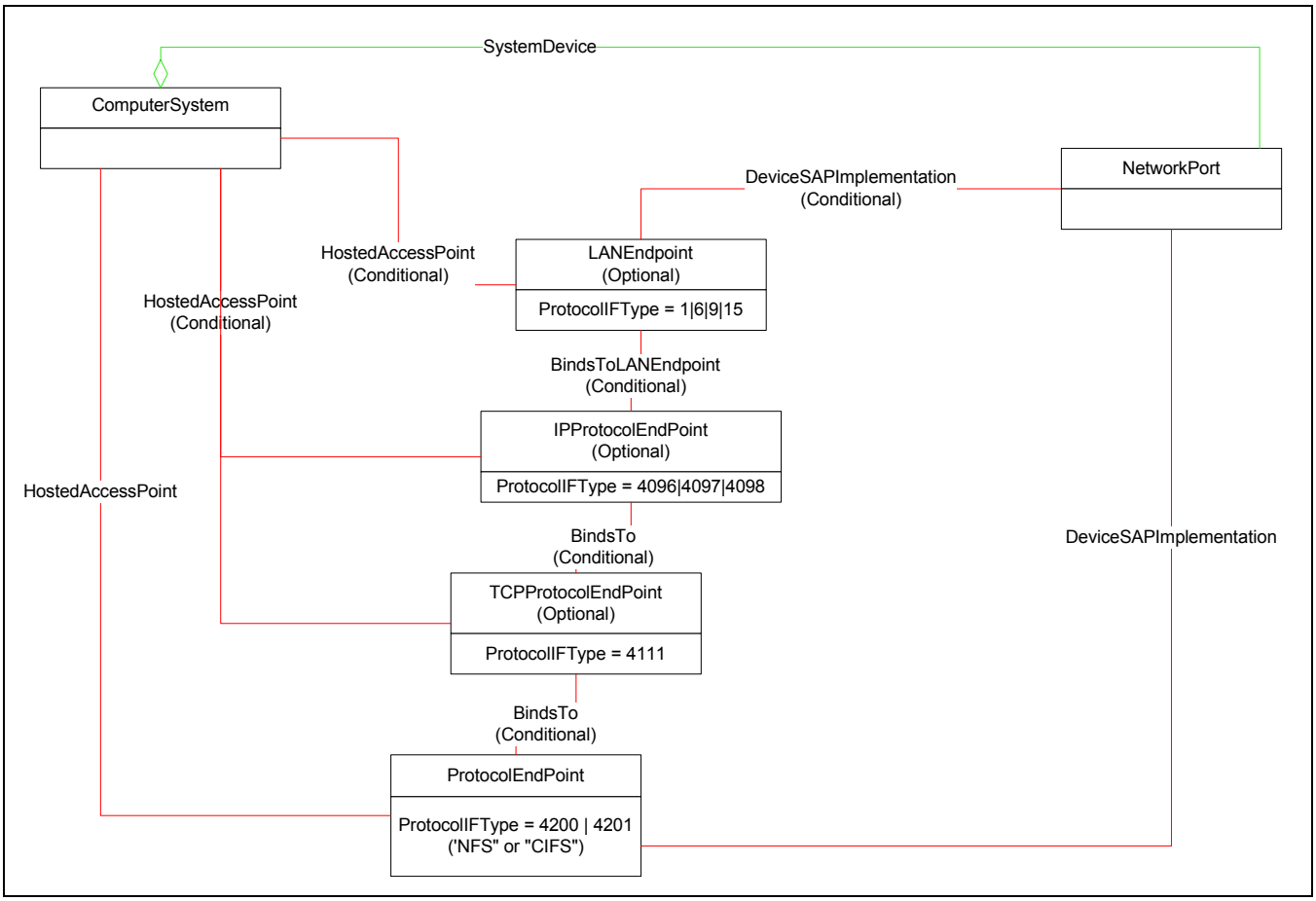

Figure 25 - Self-contained NAS Support for Front-end Network Ports

The ProtocolEndpoint for NFS or CIFS shall be present and shall be associated to a ComputerSystem via a HostedAccessPoint association. It shall also be associated to a NetworkPort via the DeviceSAPImplementation. The NetworkPort shall be modeled and shall have an SystemDevice association to a ComputerSystem. The ComputerSystem in the diagram may be the top level system for the self-contained NAS or any of its component computer systems. The ComputerSystem that hosts the NFS or CIFS ProtocolEndpont need not be the same ComputerSystem associated to the NetworkPort via its SystemDevice association.

The modeling of the TCPProtocolEndpoint, IPProtocolEndpoint and the LANEndpoint are optional. The associations from (to) those classes are conditional on the existence of the classes. Like the NFS or CIFS ProtocolEndpoint, the TCPProtocolEndpoint, IPProtocolEndpoint and LANEndpoint shall have HostedAccessPoint associations to some ComputerSystem. Typically, this would be the same ComputerSystem that hosts the NetworkPort. However this is not a requirement.

13.2 Health and Fault Management Considerations

Self-Contained NAS supports state information (e.g., OperationalStatus) on the following elements of the model:

- Network Ports (See 13.2.1 OperationalStatus for Network Ports)
- Back-end Ports (See 17.3.3 Health and Fault Management Considerations of*Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*)
- ComputerSystems (See 25.1.5 Computer System Operational Status of *Storage Management Technical Specification, Part 2 Common Profiles, 1.3.0 Rev 6*)
- FileShares that are exported (See 4.2.1 OperationalStatus for FileShares)
- LocalFileSystems (See 8.2.1 OperationalStatus for Filesystems)
- ProtocolEndpoints (See 13.2.2 OperationalStatus for ProtocolEndpoints)
- DiskDrive (See 11.2 Health and Fault Management Considerations of *Storage Management Technical Specification, Part 3 Block Devices, 1.3.0 Rev 6*)

13.2.1 OperationalStatus for Network Ports

Table 202 - NetworkPort OperationalStatus

OperationalStatus	Description
OK	Port is online
Error	Port has a failure
Stopped	Port is disabled
InService	Port is in Self Test
Unknown	

13.2.2 OperationalStatus for ProtocolEndpoints

Table 203 - ProtocolEndpoint OperationalStatus

OperationalStatus	Description
OK	ProtocolEndpoint is online
Error	ProtocolEndpoint has a failure
Stopped	ProtocolEndpoint is disabled
Unknown	

13.3 Cascading Considerations

Not Applicable.

13.4 Supported Subprofiles and Packages

Table 204 describes the supported profiles for Self-contained NAS System.

Table 204 - Supported Profiles for Self-contained NAS System

Registered Profile Names	Mandatory	Version
Indication	Yes	1.3.0
Filesystem	Yes	1.3.0
File Storage	Yes	1.3.0
File Export	Yes	1.3.0
Access Points	No	1.3.0
Multiple Computer System	No	1.2.0
Software	No	1.3.0
Location	No	1.3.0

Table 204 - Supported Profiles for Self-contained NAS System

Registered Profile Names	Mandatory	Version
Extent Composition	No	1.2.0
Filesystem Manipulation	No	1.3.0
File Export Manipulation	No	1.3.0
File Server Manipulation	No	1.3.0
Filesystem Performance	No	1.3.0
FileSystem Quotas	No	1.3.0
Job Control	No	1.3.0
Disk Drive Lite	No	1.3.0
SPI Initiator Ports	No	1.2.0
FC Initiator Ports	No	1.3.0
iSCSI Initiator Ports	No	1.2.0
Device Credentials	No	1.3.0
Physical Package	Yes	1.3.0
Block Services	Yes	1.3.0
Health	Yes	1.2.0

13.5 Methods of the Profile

13.5.1 Extrinsic Methods of the Profile

None.

13.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Specifically, the list of intrinsic operations supported are as follows:

- GetInstance
- Associators
- AssociatorNames
- References
- ReferenceNames
- EnumerateInstances
- EnumerateInstanceNames

Manipulation functions are supported in subprofiles of the profile.

13.6 Client Considerations and Recipes

Not defined in this version of the specification.

13.7 Registered Name and Version

Self-contained NAS System version 1.3.0

13.8 CIM Elements

Table 205 describes the CIM elements for Self-contained NAS System.

Table 205 - CIM Elements for Self-contained NAS System

Element Name	Requirement	Description
13.8.1 CIM_BindsTo (CIFS or NFS)	Conditional	Conditional requirement: This is required if a TCPProtocolEndpoint exists.Associates a CIFS or NFS ProtocolEndpoint to an underlying TCPProtocolEndpoint. This is used in the Self-contained NAS System to support the TCP/IP Network protocol stack.
13.8.2 CIM_BindsTo (TCP)	Conditional	Conditional requirement: This is required if an IPProtocolEndpoint exists.Associates a TCPProtocolEndpoint to an underlying IPProtocolEndpoint. This is used in the Self-contained NAS System to support the TCP/IP Network protocol stack.
13.8.3 CIM_BindsToLANEndpoint	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Associates an IPProtocolEndpoint to an underlying LANEndpoint in the Self-contained NAS System (to support the TCP/IP Network protocol stack).
13.8.4 CIM_ComputerSystem (Top Level)	Mandatory	This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the Self-contained NAS System.
13.8.5 CIM_ComputerSystem (Virtual File Server)	Optional	This represents a Virtual File Server, if one exists.
13.8.6 CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)	Mandatory	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.

Table 205 - CIM Elements for Self-contained NAS System

Element Name	Requirement	Description
13.8.7 CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
13.8.8 CIM_HostedAccessPoint (CIFS or NFS)	Mandatory	(CIFS or NFS) Represents the association between a front end ProtocolEndpoint and the Computer System that hosts it.
13.8.9 CIM_HostedAccessPoint (IP)	Conditional	Conditional requirement: This is required if an IPProtocolEndpoint exists.Represents the association between a front end IPProtocolEndpoint and the Computer System that hosts it.
13.8.10 CIM_HostedAccessPoint (LAN)	Conditional	Conditional requirement: This is required if a LANEndpoint exists.Represents the association between a front end LANEndpoint and the Computer System that hosts it.
13.8.11 CIM_HostedAccessPoint (TCP)	Conditional	Conditional requirement: This is required if a TCPProtocolEndpoint exists.Represents the association between a front end TCPProtocolEndpoint and the Computer System that hosts it.
13.8.12 CIM_HostedDependency	Optional	Associates a Virtual File Server to the Computer System hosting it. This is required if a Virtual File Server exists.
13.8.13 CIM_IPProtocolEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
13.8.14 CIM_LANEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
13.8.15 CIM_LogicalDisk (Disk for FS)	Mandatory	Represents LogicalDisks used for building LocalFileSystems.
13.8.16 CIM_NetworkPort	Mandatory	Represents the front end logical port that supports access to a local area network.
13.8.17 CIM_ProtocolEndpoint (CIFS or NFS)	Mandatory	(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
13.8.18 CIM_SystemDevice (Logical Disks)	Mandatory	This association links all LogicalDisks to the scoping system.
13.8.19 CIM_SystemDevice (Network Ports)	Mandatory	This association links all NetworkPorts to the scoping system.
13.8.20 CIM_TCPProtocolEndpoint	Optional	Represents the front-end ProtocolEndpoint used to support TCP services.

Table 205 - CIM Elements for Self-contained NAS System

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	Optional	CQL -Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.CIM_NetworkPort::OperationalStatus <> PreviousInstance.CIM_NetworkPort::OperationalStatus	Optional	CQL -Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.CIM_ProtocolEndpoint::OperationalStatus <> PreviousInstance.CIM_ProtocolEndpoint::OperationalStatus	Optional	CQL -Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Deprecated WQL -Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

Table 205 - CIM Elements for Self-contained NAS System

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	CQL -Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity="SNIA" and MessageID="FSM003"	Optional	CQL -This is a bellwether indication of a change of status of a LogicalDisk.

13.8.1 CIM_BindsTo (CIFS or NFS)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a TCPProtocolEndpoint exists.

Table 206 describes class CIM_BindsTo (CIFS or NFS).

Table 206 - SMI Referenced Properties/Methods for CIM_BindsTo (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ProtocolEndpoint that uses a lower level ProtocolEndpoint for connectivity.
Antecedent		Mandatory	The TCPProtocolEndpoint that supports a CIFS or NFS ProtocolEndpoint.

13.8.2 CIM_BindsTo (TCP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if an IPProtocolEndpoint exists.

Table 207 describes class CIM_BindsTo (TCP).

Table 207 - SMI Referenced Properties/Methods for CIM_BindsTo (TCP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The TCPProtocolEndpoint that uses an IPProtocolEndpoint for connectivity.
Antecedent		Mandatory	The IPProtocolEndpoint that supports a TCPProtocolEndpoint.

13.8.3 CIM_BindsToLANEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 208 describes class CIM_BindsToLANEndpoint.

Table 208 - SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint

Properties	Flags	Requirement	Description & Notes
FrameType		Mandatory	Only supports 1="Ethernet" at this point.
Dependent		Mandatory	A IPProtocolEndpoint.
Antecedent		Mandatory	A LANEndpoint.

13.8.4 CIM_ComputerSystem (Top Level)

Created By: Static
Modified By: External
Deleted By: Static
Requirement: Mandatory

Table 209 describes class CIM_ComputerSystem (Top Level).

Table 209 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	The actual class of this object, e.g., Vendor_NASComputerSystem.
ElementName		Mandatory	User-friendly name
Name		Mandatory	Unique identifier for the Self-contained NAS System in a format specified by NameFormat. For example, IP address or Vendor/Model/SerialNo.

Table 209 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Properties	Flags	Requirement	Description & Notes
OperationalStatus		Mandatory	Overall status of the Self-contained NAS System
NameFormat		Mandatory	Format for Name property.
PrimaryOwnerContact	M	Optional	Owner of the Self-contained NAS System
PrimaryOwnerName	M	Optional	Contact details for owner
Dedicated		Mandatory	This shall indicate that this computer system is dedicated to operation as a Self-contained NAS (25).
OtherIdentifyingInfo	C	Mandatory	An array of know identifiers for the Self-contained NAS.
IdentifyingDescriptions	C	Mandatory	An array of descriptions of the OtherIdentifyingInfo. Some of the descriptions would be "Ipv4 Address", "Ipv6 Address" or "Fully Qualified Domain Name".
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
Roles	N	Optional	Not Specified in this version of the Profile.
OtherDedicatedDescriptions	N	Optional	Not Specified in this version of the Profile.
ResetCapability	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.

13.8.5 CIM_ComputerSystem (Virtual File Server)

Created By: Static
Modified By: Static
Deleted By: Static
Requirement: Optional

Table 210 describes class CIM_ComputerSystem (Virtual File Server).

Table 210 - SMI Referenced Properties/Methods for CIM_ComputerSystem (Virtual File Server)

Properties	Flags	Requirement	Description & Notes
Dedicated		Mandatory	A Virtual File Server is a File Server (16).
NameFormat		Mandatory	Format for Name property. This shall be "Other".
Name	C	Mandatory	Unique identifier for the Self-contained NAS System's Virtual File Servers (Eg Vendor/Model/ SerialNo+FS+Number).
OperationalStatus		Mandatory	Overall status of the Virtual File Server.
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
ElementName		Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
Roles	N	Optional	Not Specified in this version of the Profile.
OtherDedicatedDescriptions	N	Optional	Not Specified in this version of the Profile.
ResetCapability	N	Optional	Not Specified in this version of the Profile.
PrimaryOwnerContact	N	Optional	Not Specified in this version of the Profile.
PrimaryOwnerName	N	Optional	Not Specified in this version of the Profile.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.

13.8.6 CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 211 describes class CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort).

Table 211 - SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ProtocolEndpoint that supports on the NetworkPort. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		Mandatory	The NetworkPort supported by the Access Point.

13.8.7 CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 212 describes class CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort).

Table 212 - SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	A LANEndpoint that depends on a NetworkPort for connecting to its LAN segment.
Antecedent		Mandatory	The Logical network adapter device that connects to a LAN.

13.8.8 CIM_HostedAccessPoint (CIFS or NFS)

Created By: External
Modified By: Static
Deleted By: External
Requirement: Mandatory

Table 213 describes class CIM_HostedAccessPoint (CIFS or NFS).

Table 213 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The ServiceAccessPoint hosted on the FileServer. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		Mandatory	The Computer System hosting this Access Point. In the context of the Self-contained NAS System, these are alway FileServers (Dedicated=16).

13.8.9 CIM_HostedAccessPoint (IP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if an IPProtocolEndpoint exists.

Table 214 describes class CIM_HostedAccessPoint (IP).

Table 214 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (IP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The IPProtocolEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

13.8.10 CIM_HostedAccessPoint (LAN)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a LANEndpoint exists.

Table 215 describes class CIM_HostedAccessPoint (LAN).

Table 215 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (LAN)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The LANEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

13.8.11 CIM_HostedAccessPoint (TCP)

Created By: External
Modified By: Static
Deleted By: External
Requirement: This is required if a TCPProtocolEndpoint exists.

Table 216 describes class CIM_HostedAccessPoint (TCP).

Table 216 - SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The TCPProtocolEndpoint hosted on the file server.
Antecedent		Mandatory	The Computer System hosting this Access Point.

13.8.12 CIM_HostedDependency

Created By: External
Modified By: Static
Deleted By: External
Requirement: Optional

Table 217 describes class CIM_HostedDependency.

Table 217 - SMI Referenced Properties/Methods for CIM_HostedDependency

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The Virtual File Server ComputerSystem.
Antecedent		Mandatory	The hosting ComputerSystem.

13.8.13 CIM_IPProtocolEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 218 describes class CIM_IPProtocolEndpoint.

Table 218 - SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the IP Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the IP Protocol Endpoint.

Table 218 - SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	The CIM Class name of the IP Protocol Endpoint.
Name		Mandatory	The unique name of the IP Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name of the IP Protocol Endpoint.
ProtocolIFType		Mandatory	4096="IP v4", 4097="IP v6", and 4098 is both. (Note that 1="Other" is not supported.)
IPv4Address		Conditional	Conditional requirement: This is required if ProtocolIFType = 4096 or 4098.An IP v4 address in the format "A.B.C.D".
IPv6Address		Conditional	Conditional requirement: This is required if ProtocolIFType = 4097 or 4098.An IP v6 address.
SubnetMask		Conditional	Conditional requirement: This is required if ProtocolIFType = 4096 or 4098.An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		Conditional	Conditional requirement: This is required if ProtocolIFType = 4097 or 4098.For an IPv6 address.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
AddressOrigin	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

13.8.14 CIM_LANEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 219 describes class CIM_LANEndpoint.

Table 219 - SMI Referenced Properties/Methods for CIM_LANEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the LAN Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the LAN Endpoint.
CreationClassName		Mandatory	The CIM Class name of the LAN Endpoint.
Name		Mandatory	The unique name of the LAN Endpoint.
NameFormat		Mandatory	The Format of the Name for the LAN Endpoint.
ProtocolIFType		Mandatory	LAN endpoints supported are: 1="Other",6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".
OtherTypeDescription		Optional	If the LAN endpoint is a vendor-extension specified by "Other" and a description.
LANID		Optional	A unique id for the LAN segment that this device is connected to. Will be NULL if the LAN is not connected.
MACAddress		Mandatory	Primary Unicast address for this LAN device.
AliasAddresses		Mandatory	Other unicast addresses supported by this device.
GroupAddresses		Mandatory	Multicast addresses supported by this device.
MaxDataSize		Mandatory	The max size of packet supported by this LAN device. (If there were a Network subprofile, this would not be exposed in a Self-contained NAS System Profile).
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

13.8.15 CIM_LogicalDisk (Disk for FS)

Created By: Extrinsic_or_External

Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 220 describes class CIM_LogicalDisk (Disk for FS).

Table 220 - SMI Referenced Properties/Methods for CIM_LogicalDisk (Disk for FS)

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	CIM Class of the Self-contained NAS System Computer System that is the host of this LogicalDisk.
SystemName		Mandatory	Name of the Self-contained NAS System Computer System that hosts this LogicalDisk.
CreationClassName		Mandatory	CIM Class of this instance of LogicalDisk.
DeviceID		Mandatory	Opaque identifier for the LogicalDisk.
OperationalStatus		Mandatory	A subset of operational status that is applicable for LogicalDisks in a Self-contained NAS System.
ExtentStatus		Mandatory	This LogicalDisk is neither imported (16) nor exported (17).
Primordial		Mandatory	This represents a Concrete Logical Disk that is not primordial.
NameFormat		Mandatory	The format of the Name appropriate for LogicalDisks in the Self-contained NAS System. This should be coded as "1" ("other").
Name		Mandatory	Identifier for a local LogicalDisk that will be used for a filesystem; since this storage extent will be referenced by a client, it needs to have a unique name. We cannot constrain the format here, but the OS-specific format described in the Block Services specification is not appropriate, so "Other" is used.
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.

Table 220 - SMI Referenced Properties/Methods for CIM_LogicalDisk (Disk for FS)

Properties	Flags	Requirement	Description & Notes
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.
LocationIndicator	N	Optional	Not Specified in this version of the Profile.
DataOrganization	N	Optional	Not Specified in this version of the Profile.
Purpose	N	Optional	Not Specified in this version of the Profile.
Access	N	Optional	Not Specified in this version of the Profile.
ErrorMethodology	N	Optional	Not Specified in this version of the Profile.
SequentialAccess	N	Optional	Not Specified in this version of the Profile.
NameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameNamespace	N	Optional	Not Specified in this version of the Profile.
OtherNameFormat	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
Reset()		Optional	Not Specified in this version of the Profile.

13.8.16 CIM_NetworkPort

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 221 describes class CIM_NetworkPort.

Table 221 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Network adapter that provides a network port.
OperationalStatus		Mandatory	The operational status of the adapter.
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Network Port.
SystemName		Mandatory	The name of the Computer System hosting the Network Port.
CreationClassName		Mandatory	The CIM Class name of the Network Port.

Table 221 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
DeviceID		Mandatory	A unique ID for the device (in the context of the hosting System).
Speed		Optional	
MaxSpeed		Optional	
RequestedSpeed		Optional	
UsageRestriction		Optional	
PortType		Optional	
PortNumber		Optional	A unique number for the adapter in the context of the hosting System).
PermanentAddress		Mandatory	The hard-coded address of this port.
NetworkAddresses		Optional	An array of network addresses for this port.
LinkTechnology		Optional	1="Other", 2="Ethernet", 3="IB", 4="FC", 5="FDDI", 6="ATM", 7="Token Ring", 8="Frame Relay", 9="Infrared", 10="BlueTooth", 11="Wireless LAN. The link technology supported by this adapter.
OtherLinkTechnology		Optional	The vendor-specific "Other" link technology supported by this adapter.
FullDuplex		Optional	
AutoSense		Optional	
SupportedMaximumTransmissionUnit		Optional	
ActiveMaximumTransmissionUnit		Optional	
Caption	N	Optional	Not Specified in this version of the Profile.
Description	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
Name	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledState	N	Optional	Not Specified in this version of the Profile.
OtherEnabledState	N	Optional	Not Specified in this version of the Profile.
RequestedState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile.

Table 221 - SMI Referenced Properties/Methods for CIM_NetworkPort

Properties	Flags	Requirement	Description & Notes
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.
LocationIndicator	N	Optional	Not Specified in this version of the Profile.
OtherPortType	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
Reset()		Optional	Not Specified in this version of the Profile.

13.8.17 CIM_ProtocolEndpoint (CIFS or NFS)

Created By: External
Modified By: External
Deleted By: External
Requirement: Mandatory

Table 222 describes class CIM_ProtocolEndpoint (CIFS or NFS).

Table 222 - SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the Protocol Endpoint.
Name		Mandatory	The unique name of the Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	
Description		Mandatory	This shall be one of the NFS or CIFS protocol endpoints supported by the Self-contained NAS System.

Table 222 - SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS)

Properties	Flags	Requirement	Description & Notes
ProtocolIFType		Mandatory	This represents either NFS=4200 or CIFS=4201. Other protocol types are specified in subclasses of ProtocolEndpoint.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.
RequestStateChange()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

13.8.18 CIM_SystemDevice (Logical Disks)

Created By: Extrinsic_or_External_or_Static
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 223 describes class CIM_SystemDevice (Logical Disks).

Table 223 - SMI Referenced Properties/Methods for CIM_SystemDevice (Logical Disks)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this device.
PartComponent		Mandatory	The LogicalDisk that is a part of a computer system.

13.8.19 CIM_SystemDevice (Network Ports)

Created By: Extrinsic_or_External_or_Static
Modified By: Extrinsic_or_External
Deleted By: Extrinsic_or_External
Requirement: Mandatory

Table 224 describes class CIM_SystemDevice (Network Ports).

Table 224 - SMI Referenced Properties/Methods for CIM_SystemDevice (Network Ports)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this device.
PartComponent		Mandatory	The NetworkPort that is a part of a computer system.

13.8.20 CIM_TCPProtocolEndpoint

Created By: External
Modified By: External
Deleted By: External
Requirement: Optional

Table 225 describes class CIM_TCPProtocolEndpoint.

Table 225 - SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The CIM Class name of the Computer System hosting the TCP Protocol Endpoint.
SystemName		Mandatory	The name of the Computer System hosting the TCP Protocol Endpoint.
CreationClassName		Mandatory	The CIM Class name of the TCP Protocol Endpoint.
Name		Mandatory	The unique name of the TCP Protocol Endpoint.
NameFormat		Mandatory	The Format of the Name of the TCP Protocol Endpoint.
ProtocolIFType		Mandatory	4111="TCP". (Note that no other protocol type is supported by this endpoint.)
PortNumber		Mandatory	The number of the TCP Port that this element represents.
Caption	N	Optional	Not Specified in this version of the Profile.
ElementName	N	Optional	Not Specified in this version of the Profile.
InstallDate	N	Optional	Not Specified in this version of the Profile.
StatusDescriptions	N	Optional	Not Specified in this version of the Profile.
HealthState	N	Optional	Not Specified in this version of the Profile.
EnabledDefault	N	Optional	Not Specified in this version of the Profile.
OtherTypeDescription	N	Optional	Not Specified in this version of the Profile.
BroadcastResetSupported	N	Optional	Not Specified in this version of the Profile.

Table 225 - SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
RequestStateChange ()		Optional	Not Specified in this version of the Profile.
BroadcastReset()		Optional	Not Specified in this version of the Profile.

STABLE

Annex A: (Informative) State Transitions from Storage to File Shares

A filesystem is an abstract class that abstractly describes a hierarchical structuring of data into “files” contained within a tree of “directories” with a single “root” directory. A LocalFileSystem is a concrete class derived from FileSystem that implements it using one or more storage elements in which the storage element(s) has been structured to contain information about multiple files organized into directories as well as the content of these files. This internal organization of a LocalFileSystem, viz., what parts represent the components of files, what parts constitute directories, what the names of these files and directories are, how they are organized into a hierarchy, even the representation of the path to a file from the root directory through a sequence of sub-directories etc., is called “metadata” and is stored persistently inside the storage element(s). In addition to metadata, the internal organization contains information about ownership of files and directories, rights of users or other entities to access files and directories, and other attributes of files and directories. This information is sometimes included in metadata, but sometimes referred to independently as “attribute” information and is also stored persistently within the storage element(s). Finally, the contents of files are also stored persistently in the storage element(s).

In filesystem-related profiles, the collection of storage elements used by a LocalFileSystem is called a *LogicalDisk(s)*. There are multiple formats (both open and proprietary) for structuring a LogicalDisk into a LocalFileSystem—how the metadata, attributes, and content are stored and so on—that are referred to as the “type” of the LocalFileSystem. The information about the type of the LocalFileSystem (and possibly variant versions of the type) is also persistently stored in the LogicalDisk. The type of the LocalFileSystem in this and related profiles is represented as the “FileSystemType”.

Note: The Volume Composition SubProfile describes how multiple LogicalDisks can be merged into a single one. It is assumed that if more than one storage element is used, they are composed into a single LogicalDisk using the Volume Composition profile (see Clause 24: Volume Composition Profile) or other profile that similarly merges multiple storage elements into a single LogicalDisk.

A LocalFileSystem is not an autonomous entity but is a hosted component of a larger system, usually a ComputerSystem. This is represented using the HostedFileSystem association between a ComputerSystem and the LocalFileSystem. Since the LogicalDisk is a SystemDevice of a ComputerSystem, it is frequently the case that the LocalFileSystem will be hosted by the same ComputerSystem, but this is not required. It is generally the case that a LocalFileSystem will have an independent internal name that may be used to refer to it but it is not necessary that the name be constructed independently of the name of the LogicalDisk or the name of the hosting ComputerSystem. Some systems require that this internal name be globally unique, but others rely on the uniqueness of the LogicalDisk’s name or on other identifiers. In SMI-S, it is a requirement that a LocalFileSystem have a unique Name property relative to the hosting ComputerSystem (Name being one of the key properties of the FileSystem class).

The process by which an element of storage is finally made available as a FileShare is represented by Figure A.1. The process begins with an unused LogicalDisk that is owned by, or has been allocated to, the ComputerSystem for this purpose. The operation "Create a Filesystem", converts an unused LogicalDisk to a LocalFileSystem—Figure A.1 shows the name and the ComputerSystem that has a HostedFileSystem association to the LocalFileSystem. The other details of the LocalFileSystem are skipped.

.

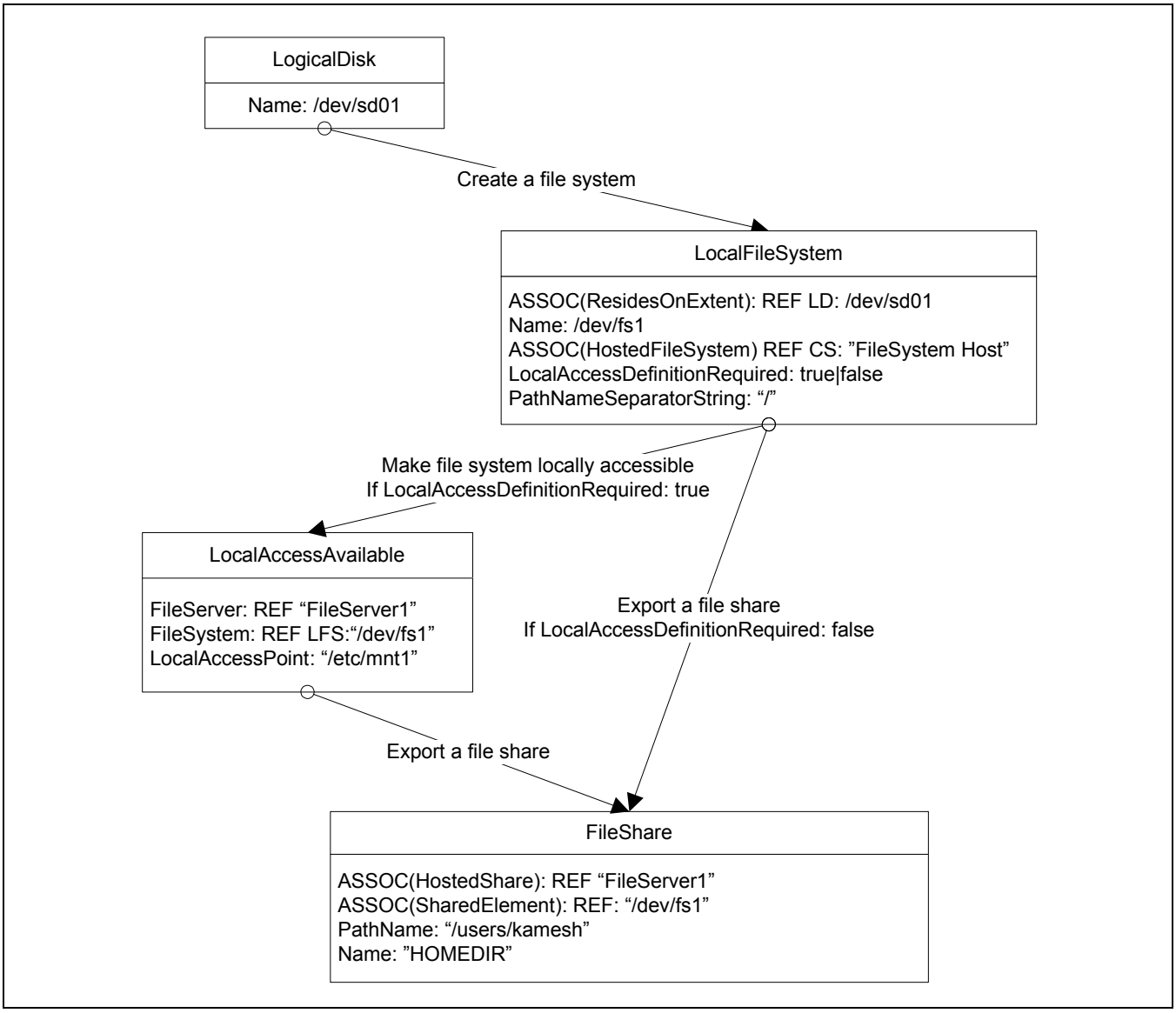

Figure A.1 - State Transitions From LogicalDisk to FileShare

Even after a LogicalDisk has been internally organized into a LocalFileSystem, it may not be usable by an operational user. That’s because the operational user needs a durable name (for referring to the LocalFileSystem) that is persistently supported by the implementation. There are multiple ways in which this problem has been solved. Since the LocalFileSystem must be hosted by a ComputerSystem and the LocalFileSystem has a unique name, a Uniform Resource Indicator (URI) can be constructed that is relative to the hosting ComputerSystem. However, an operational user needs to use an access path relative to the ComputerSystem that serves files to them (i.e., relative to a File Server), and this may differ from the hosting ComputerSystem.

Traditionally (i.e., before URIs), a LocalFileSystem was assigned a name in a hierarchical name space maintained by the File Server ComputerSystem. This assignment was called “mounting to” the name and the name was called the “mount-point” of the filesystem. For historical and other reasons, the hierarchical name space most commonly used for the purpose was based on the “root filesystem” of the File Server. This allowed a naming convention using “file path names” for objects in the namespace that could be extended uniformly to the meta-data and content of

the mounted filesystem (and would be represented in the SMI Specification as a property of a Capabilities element).

If the hosting ComputerSystem and the File Server ComputerSystem are the same, or can be referenced using a single identifier (for instance in a clustered computer system), or only one File Server can access a LocalFileSystem, it is possible to make the Name of the LocalFileSystem be the same as the mount-point. In that case, the act of “mounting to” the name is accomplished by default when the LocalFileSystem is created. But this does not work for implementations that allow a LocalFileSystem hosted by one ComputerSystem to be assigned differently named mount-points on multiple File Server ComputerSystems. The problem increases in complexity when a File Server can have multiple network identities (through a multiplicity of IP addresses and multiple fully-qualified domain names that map to each IP address).

Traditionally, Unix-based uni-processor systems have not made the Name of the LocalFileSystem the same as the mount-point. But many specialized systems follow such a policy, so whether mounting is not managed explicitly (because it is automatically specified by the name of the LocalFileSystem) or must be managed explicitly is a feature of an implementation.

In addition to specifying a mount-point for a LocalFileSystem, a File Server would also assign system resources needed for working with the LocalFileSystem. These include read and write buffers of appropriate capacity, restrictions on reading or writing (needed for systems that allow multiple mounts of a LocalFileSystem), and other implementation-dependent resources. The specification of these resources are explicitly manageable by some implementations and defaulted by others.

The terms “mount” and “mount-point” are also traditionally used for assigning a remote element (such as a shared file) a name in the local name space of a ComputerSystem. These terms by themselves appeared to be too generic for use in this specification, so the terms used are “make locally accessible” for “mount” and “local access point” for “mount-point”. The resources to be allocated for mounting are specified by “local access settings”.

In SMI-S, a “locally accessible filesystem” is represented by providing an association, LocalAccessAvailable, from the File Server to the LocalFileSystem. In addition to the key reference properties, this association provides the LocalAccessPoint string array property that specifies the “local access point”. Referring back to Figure A.1, the "Make a Filesystem Locally Accessible" operation creates the LocalAccessAvailable association between the File Server and the LocalFileSystem. This operation is supported in the Filesystem Manipulation Subprofile by providing appropriate parameters to the CreateFileSystem and ModifyFileSystem extrinsic methods. The LocalFileSystem element is not modified, but a new association is created. The LocalAccessPoint property provides the access point (shown in the standard Unix format as “/etc/mnt1”).

Note: The intent behind implementing "Make a Filesystem Locally Accessible" with CreateFileSystem and ModifyFileSystem methods is that it is preferable not to distinguish between implementations that implement a separate “Make Locally Accessible” function from those that do not.

The vendor implements this operation by providing the necessary parameters to the create and modify methods; this has the benefit that the operation does not have to be exposed separately to the management client. However all implementations that support multiple File Servers with independent names to access filesystems must support LocalAccessAvailable as that is the only place where a file-server-specific name for the LocalFileSystem is specified (by the LocalAccessPoint property). A vendor that provides accessibility by default might have a FileSystem.Name property that also functions as a path name from each file server (in one sample implementation), so it is likely that LocalAccessAvailable.LocalAccessPoint would be the same as the LocalFileSystem.Name property. The property LocalFileSystem.LocalAccessDefinitionRequired is required to indicate that this feature is used and that the client must examine that property to understand how a vendor implements this model.

The creation of a file share and its behavior are detailed in the related File Export and File Export Manipulation subprofiles. Figure A.1 shows the "Export a file share" operation that creates a FileShare and an SharedElement association. The FileShare provides a name “HOMEDIR” and is hosted by the File Server. The SharedElement association links to the LocalFileSystem and also provides the pathname “/users/kamesh” to the specific user’s home directory.

Note: Once a LocalFileSystem has been made locally accessible via a File Server, the File Server can share its contents with remote operational users. The contents of such a filesystem can be shared all the way from the root directory at the top of the hierarchy, or the contents of sub-tree below some contained internal directory may be shared, or a specific file contained in the filesystem may be shared. When a directory (root or otherwise) is shared, all files and sub-directories of that directory are automatically also shared recursively. The semantics of sharing an individual file or directory are ultimately controlled by the implementation of the filesystem, so sharing cannot violate the access rules specified internally to the filesystem. In addition to specifying the object (file or directory) to be shared, the File Server may specify the protocol to use for sharing and a correlatable name by which remote users can refer to the shared object—the protocol, the unique server id, and the share name can be used to construct a URI for the shared object. The base URI can be extended to construct a reference URI for files or subdirectories within the shared object.

In SMI-S, there is a FileShare element created to represent the externally accessible share. This element is associated via SharedElement to the LocalFileSystem. The FileShare element will provide the PathName string property that specifies the shared object (the contained file or directory name).