



# **Storage Management Technical Specification, Part 4 File Systems**

## **Version 1.2.0, Revision 6**

"This document has been released and approved by the SNIA. The SNIA believes that the ideas, methodologies and technologies described in this document accurately represent the SNIA goals and are appropriate for widespread distribution. Suggestion for revision should be directed to the Technical Council Managing Director at [tcmd@snia.org](mailto:tcmd@snia.org)."

***SNIA Technical Position***

***22 October, 2007***



## Errata/Change Log

20071022

No errata have been identified for 1.2.0.

The SNIA hereby grants permission for individuals to use this document for personal use only, and for corporations and other business entities to use this document for internal use only (including internal copying, distribution, and display) provided that:

- 1) Any text, diagram, chart, table or definition reproduced must be reproduced in its entirety with no alteration, and,
- 2) Any document, printed or electronic, in which material from this document (or any portion hereof) is reproduced must acknowledge the SNIA copyright on that material, and must credit the SNIA for granting permission for its reuse.

Other than as explicitly provided above, you may not make any commercial use of this document, sell any or this entire document, or distribute this document to third parties. All rights not explicitly granted are expressly reserved to SNIA.

Permission to use this document for purposes other than those enumerated above may be requested by e-mailing [tcmd@snia.org](mailto:tcmd@snia.org) please include the identity of the requesting individual and/or company and a brief description of the purpose, nature, and scope of the requested use.

Copyright © 2003-2007 Storage Networking Industry Association.

## 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/>.

Copyright © 2003-2007 SNIA. All rights reserved. All other trademarks or registered trademarks are the property of their respective owners.

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

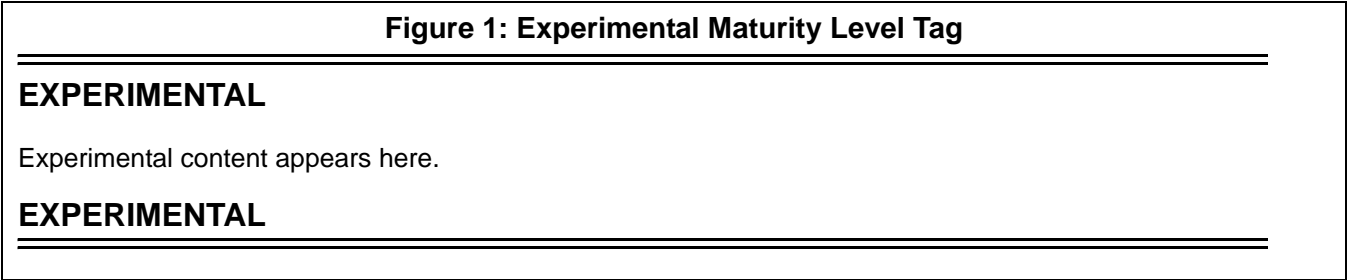
This specification has been structured to convey both the formal requirements and assumptions of the SMI-S API and its emerging implementation and deployment lifecycle. Over time, the intent is that all content in the specification will represent a mature and stable design, be verified by extensive implementation 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.

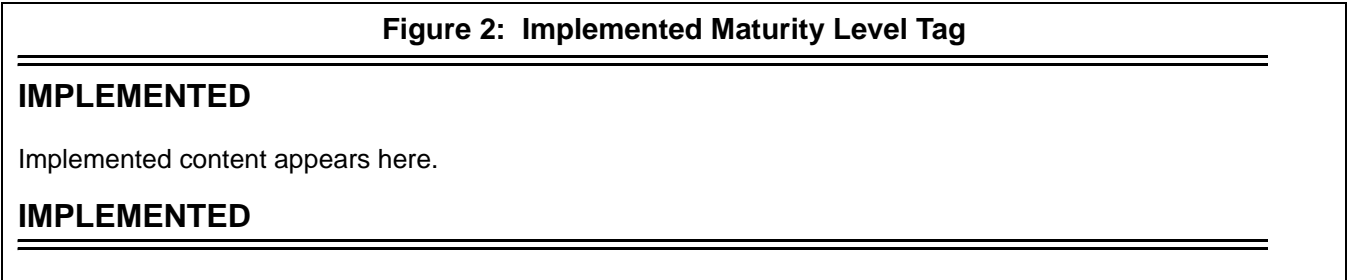
**Experimental Maturity Level**

No material is included in this specification unless its initial architecture has been completed and reviewed. 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. 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. 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.



**Implemented Maturity Level**

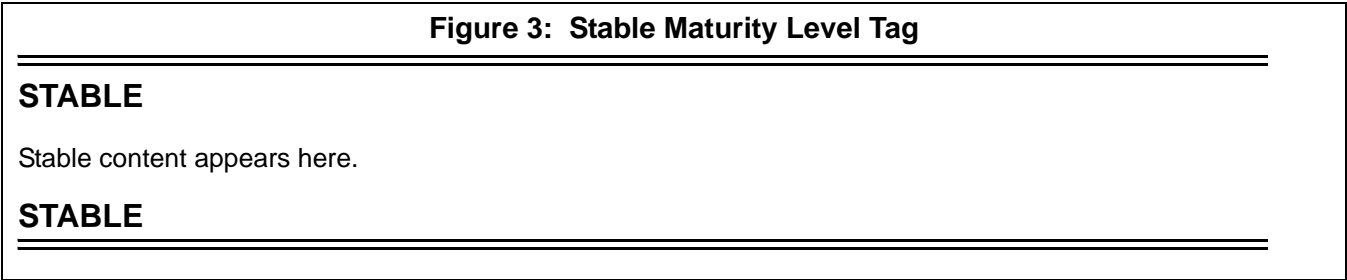
Profiles for which initial implementations have been completed are classified as “Implemented”. This indicates that at least two different vendors have implemented the profile, including at least one provider implementation. At this maturity level, the underlying architecture and modeling are stable, 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.



**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.



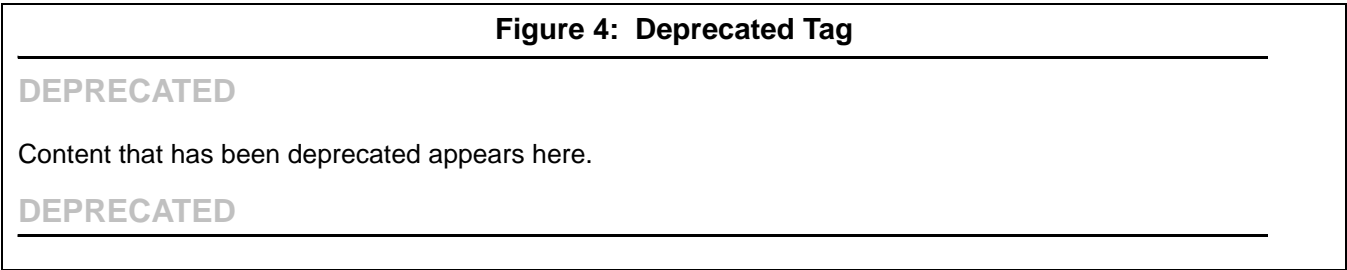
Finalized Maturity Level

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.



## USAGE

The SNIA hereby grants permission for individuals to use this document for personal use only, and for corporations and other business entities to use this document for internal use only (including internal copying, distribution, and display) provided that:

- 1) Any text, diagram, chart, table or definition reproduced shall be reproduced in its entirety with no alteration.
- 2) Any document, printed or electronic, in which material from this document (or any portion hereof) is reproduced shall acknowledge the SNIA copyright on that material, and shall credit the SNIA for granting permission for its reuse.

Other than as explicitly provided above, you may not make any commercial use of this document, sell any or this entire document, or distribute this document to third parties. All rights not explicitly granted are expressly reserved to SNIA.

Permission to use this document for purposes other than those enumerated above may be requested by e-mailing [tcmd@snia.org](mailto:tcmd@snia.org) please include the identity of the requesting individual and/or company and a brief description of the purpose, nature, and scope of the requested use.



# Contents

Errata/Change Log .....	iii
List of Tables .....	xi
List of Figures .....	xv
Foreword.....	xvii
<b>1. Scope .....</b>	<b>1</b>
<b>2. Normative References .....</b>	<b>3</b>
2.1 General.....	3
2.2 Approved references .....	3
2.3 References under development .....	3
2.4 Other references.....	3
<b>3. Terms and definitions .....</b>	<b>5</b>
3.1 General.....	5
3.2 Definitions.....	5
<b>4. File Export Profile .....</b>	<b>7</b>
4.1 Description.....	7
4.2 Health and Fault Management Consideration .....	9
4.3 Cascading Considerations.....	9
4.4 Supported Profiles, Subprofiles, and Packages .....	9
4.5 Methods of the Profile.....	9
4.6 Client Considerations and Recipes .....	10
4.7 Registered Name and Version .....	10
4.8 CIM Elements .....	11
<b>5. File Export Manipulation Subprofile .....</b>	<b>17</b>
5.1 Description.....	17
5.2 Health and Fault Management Considerations .....	22
5.3 Cascading Considerations.....	24
5.4 Supported Subprofiles and Packages .....	24
5.5 Methods of the Profile.....	25
5.6 Client Considerations and Recipes .....	40
5.7 Registered Name and Version .....	51
5.8 CIM Elements.....	52
<b>6. File Storage Profile .....</b>	<b>69</b>
6.1 Description.....	69
6.2 Health and Fault Management Consideration .....	70
6.3 Cascading Considerations.....	70
6.4 Supported Profiles, Subprofiles, and Packages .....	72
6.5 Methods of the Profile.....	72
6.6 Client Considerations and Recipes .....	73
6.7 Registered Name and Version .....	73
6.8 CIM Elements.....	73
<b>7. Filesystem Profile .....</b>	<b>75</b>
7.1 Description.....	75
7.2 Health and Fault Management Consideration .....	78
7.3 Cascading Considerations.....	79
7.4 Supported Profiles, Subprofiles, and Packages .....	80
7.5 Methods of the Profile.....	80
7.6 Client Considerations: Use Cases.....	80
7.7 Registered Name and Version .....	90
7.8 CIM Elements .....	91
<b>8. Filesystem Manipulation Subprofile .....</b>	<b>107</b>
8.1 Description.....	107
8.2 Health and Fault Management Considerations .....	114

8.3	Cascading Considerations.....	116
8.4	Supported Subprofiles and Packages .....	116
8.5	Methods of the Profile.....	117
8.6	Client Considerations and Recipes .....	139
8.7	Registered Name and Version .....	158
8.8	CIM Elements .....	159
<b>9.</b>	<b>Filesystem Quotas Profile.....</b>	<b>193</b>
9.1	Description.....	193
9.2	Health and Fault Management Considerations .....	196
9.3	Supported Profiles, Subprofiles, and Packages .....	196
9.4	Methods of the Profile.....	196
9.5	Client Considerations and sample code .....	199
9.6	Registered Name and Version .....	205
9.7	CIM Elements .....	206
<b>10.</b>	<b>NAS Head Profile .....</b>	<b>215</b>
10.1	Description.....	215
10.2	Health and Fault Management Considerations .....	222
10.3	Cascading Considerations.....	223
10.4	Supported Subprofiles and Packages .....	223
10.5	Methods of the Profile.....	224
10.6	Client Considerations and Recipes .....	225
10.7	Registered Name and Version .....	225
10.8	CIM Elements .....	226
<b>11.</b>	<b>Self-Contained NAS Profile .....</b>	<b>249</b>
11.1	Description.....	249
11.2	Health and Fault Management Considerations .....	254
11.3	Cascading Considerations.....	255
11.4	Supported Subprofiles and Packages .....	255
11.5	Methods of the Profile.....	256
11.6	Client Considerations and Recipes .....	256
11.7	Registered Name and Version .....	257
11.8	CIM Elements .....	258
<b>Annex A.</b>	<b>(Informative) State Transitions from Storage to File Shares.....</b>	<b>277</b>

## List of Tables

Table 1.	FileShare OperationalStatus .....	9
Table 2.	Supported Profiles for File Export.....	9
Table 3.	CIM Elements for File Export.....	11
Table 4.	SMI Referenced Properties/Methods for CIM_ConcreteDependency .....	12
Table 5.	SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare).....	12
Table 6.	SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (Setting) .....	13
Table 7.	SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share).....	14
Table 8.	SMI Referenced Properties/Methods for CIM_HostedShare.....	15
Table 9.	SMI Referenced Properties/Methods for CIM_SAPAvailableForElement .....	15
Table 10.	SMI Referenced Properties/Methods for SNIA_SharedElement .....	15
Table 11.	Operational Status for FileExport Service .....	22
Table 12.	Operational Status for File Server ComputerSystem .....	23
Table 13.	Supported Profiles for File Export Manipulation .....	24
Table 14.	FileExportManipulation Methods .....	25
Table 15.	Parameters for Extrinsic Method ExportedFileShareCapabilities.CreateGoalSettings .....	27
Table 16.	Parameters for Extrinsic Method FileExportServices.CreateExportedShare .....	30
Table 17.	Parameters for Extrinsic Method FileExportServices.ModifyExportedShare.....	34
Table 18.	Parameters for Extrinsic Method FileExportServices.ReleaseExportedShare .....	39
Table 19.	SMI-S File Export Supported Capabilities Patterns.....	51
Table 20.	CIM Elements for File Export Manipulation .....	52
Table 21.	SMI Referenced Properties/Methods for SNIA_FileExportService.....	54
Table 22.	SMI Referenced Properties/Methods for CIM_HostedService .....	55
Table 23.	SMI Referenced Properties/Methods for SNIA_FileExportCapabilities (FES Configuration) .....	55
Table 24.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (FES Configuration) .....	57
Table 25.	SMI Referenced Properties/Methods for SNIA_ExportedFileShareCapabilities (FES Capabilities) .....	57
Table 26.	SMI Referenced Properties/Methods for SNIA_ElementCapabilities (FES Capabilities) .....	58
Table 27.	SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (Pre-defined).....	59
Table 28.	SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Pre-defined).....	61
Table 29.	SMI Referenced Properties/Methods for SNIA_ExportedFileShareSetting (FileShare Setting).....	61
Table 30.	SMI Referenced Properties/Methods for SNIA_FileShare (Exported File Share) .....	63
Table 31.	SMI Referenced Properties/Methods for SNIA_HostedShare.....	64
Table 32.	SMI Referenced Properties/Methods for CIM_ServiceAffectsElement .....	65
Table 33.	SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare Setting).....	65
Table 34.	SMI Referenced Properties/Methods for CIM_LogicalFile (Subelement).....	66
Table 35.	SMI Referenced Properties/Methods for SNIA_SharedElement .....	66
Table 36.	SMI Referenced Properties/Methods for CIM_FileStorage (Subelement).....	67
Table 37.	SMI Referenced Properties/Methods for CIM_ConcreteDependency .....	67
Table 38.	SMI Referenced Properties/Methods for CIM_SAPAvailableForElement .....	68
Table 39.	Cascaded Storage.....	72
Table 40.	CIM Elements for File Storage .....	73
Table 41.	SMI Referenced Properties/Methods for CIM_ResidesOnExtent.....	74
Table 42.	Filesystem OperationalStatus.....	79
Table 43.	Supported Profiles for Filesystem.....	80
Table 44.	CIM Elements for Filesystem.....	91
Table 45.	SMI Referenced Properties/Methods for CIM_ElementSettingData (FileSystem) .....	92
Table 46.	SMI Referenced Properties/Methods for CIM_ElementSettingData (Local Access Required) .....	93
Table 47.	SMI Referenced Properties/Methods for CIM_FileStorage .....	93

Table 48.	SMI Referenced Properties/Methods for SNIA_LocalAccessAvailable .....	94
Table 49.	SMI Referenced Properties/Methods for CIM_HostedDependency (Local Access Required) .....	94
Table 50.	SMI Referenced Properties/Methods for CIM_Dependency .....	95
Table 51.	SMI Referenced Properties/Methods for SNIA_FileSystemSetting .....	95
Table 52.	SMI Referenced Properties/Methods for CIM_HostedFileSystem (LocalFileSystem) .....	97
Table 53.	SMI Referenced Properties/Methods for SNIA_LocalFileSystem .....	97
Table 54.	SMI Referenced Properties/Methods for CIM_LogicalFile .....	99
Table 55.	SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting .....	100
Table 56.	SMI Referenced Properties/Methods for CIM_Dependency (Uses Directory Services From) .....	105
Table 57.	LocalFileSystem OperationalStatus .....	114
Table 58.	Supported Profiles for Filesystem Manipulation .....	116
Table 59.	Filesystem Manipulation Methods that cause Instance Creation, Deletion or Modification .....	117
Table 60.	Parameters for Extrinsic Method FileSystemCapabilities.CreateGoalSettings .....	119
Table 61.	Parameters for Extrinsic Method FileSystemCapabilities.GetRequiredStorageSize .....	120
Table 62.	Parameters for Extrinsic Method LocallyAccessibleFileSystemCapabilities.CreateGoalSettings .....	123
Table 63.	Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem .....	127
Table 64.	Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem .....	134
Table 65.	Parameters for Extrinsic Method FileSystemConfigurationService.DeleteFileSystem .....	138
Table 66.	Filesystem Manipulation Supported Capabilities Patterns .....	158
Table 67.	CIM Elements for Filesystem Manipulation .....	159
Table 68.	SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationService .....	163
Table 69.	SMI Referenced Properties/Methods for CIM_HostedService .....	164
Table 70.	SMI Referenced Properties/Methods for SNIA_FileSystemConfigurationCapabilities .....	165
Table 71.	SMI Referenced Properties/Methods for SNIA_ElementCapabilities (FS Configuration Capabilities) .....	168
Table 72.	SMI Referenced Properties/Methods for SNIA_FileSystemCapabilities .....	168
Table 73.	SMI Referenced Properties/Methods for SNIA_ElementCapabilities (At Least One) .....	169
Table 74.	SMI Referenced Properties/Methods for SNIA_ElementCapabilities (At Least One) .....	169
Table 75.	SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Pre-defined FS Settings) .....	170
Table 76.	SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Pre-defined FS Settings) .....	171
Table 77.	SMI Referenced Properties/Methods for SNIA_FileSystemSetting (Attached to FileSystem) .....	172
Table 78.	SMI Referenced Properties/Methods for CIM_ElementSettingData (Attached to Filesystem) .....	174
Table 79.	SMI Referenced Properties/Methods for SNIA_LocalFileSystem .....	175
Table 80.	SMI Referenced Properties/Methods for CIM_HostedFileSystem .....	177
Table 81.	SMI Referenced Properties/Methods for CIM_FileStorage (Root Directory) .....	177
Table 82.	SMI Referenced Properties/Methods for CIM_LogicalFile (Shared Files and Directories) .....	178
Table 83.	SMI Referenced Properties/Methods for CIM_FileStorage (Shared Files and Directories) .....	179
Table 84.	SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemSetting .....	179
Table 85.	SMI Referenced Properties/Methods for CIM_Dependency (Uses Directory Services From) .....	184
Table 86.	SMI Referenced Properties/Methods for SNIA_SettingsDefineCapabilities (Pre-defined Local Access Settings) .....	185
Table 87.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (Local Access Configuration Capabilities) ....	186
Table 88.	SMI Referenced Properties/Methods for SNIA_LocallyAccessibleFileSystemCapabilities .....	186
Table 89.	SMI Referenced Properties/Methods for CIM_HostedDependency (Attached to File System) .....	188
Table 90.	SMI Referenced Properties/Methods for CIM_HostedDependency (Pre-Defined) .....	189
Table 91.	SMI Referenced Properties/Methods for CIM_HostedDependency (Pre-Defined) .....	189
Table 92.	SMI Referenced Properties/Methods for CIM_ElementSettingData (Local Access Required) .....	190
Table 93.	SMI Referenced Properties/Methods for SNIA_LocalAccessAvailable .....	190
Table 94.	Supported Profiles for FileSystem Quotas .....	196
Table 95.	CIM Elements for FileSystem Quotas .....	206
Table 96.	SMI Referenced Properties/Methods for SNIA_FSDomainIdentity .....	207

Table 97.	SMI Referenced Properties/Methods for SNIA_FSQuotaCapabilities.....	207
Table 98.	SMI Referenced Properties/Methods for SNIA_FSQuotaReportRecord.....	208
Table 99.	SMI Referenced Properties/Methods for SNIA_FSQuotaIndication.....	209
Table 100.	SMI Referenced Properties/Methods for SNIA_FSQuotaManagementService.....	210
Table 101.	SMI Referenced Properties/Methods for SNIA_FSQuotaConfigEntry.....	211
Table 102.	SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToElement.....	212
Table 103.	SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToPrincipal.....	212
Table 104.	SMI Referenced Properties/Methods for SNIA_FSQuotaAppliesToTree.....	212
Table 105.	NetworkPort OperationalStatus.....	222
Table 106.	ProtocolEndpoint OperationalStatus.....	222
Table 107.	Supported Profiles for NAS Head.....	223
Table 108.	CIM Elements for NAS Head.....	226
Table 109.	SMI Referenced Properties/Methods for CIM_BindsTo (CIFS or NFS).....	229
Table 110.	SMI Referenced Properties/Methods for CIM_BindsTo (TCP).....	229
Table 111.	SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint.....	230
Table 112.	SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level).....	230
Table 113.	SMI Referenced Properties/Methods for CIM_ComputerSystem (File Server).....	232
Table 114.	SMI Referenced Properties/Methods for CIM_ConcreteComponent.....	233
Table 115.	SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort).....	233
Table 116.	SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort).....	234
Table 117.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS).....	234
Table 118.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP).....	235
Table 119.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (IP).....	235
Table 120.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (LAN).....	235
Table 121.	SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint.....	236
Table 122.	SMI Referenced Properties/Methods for CIM_LANEndpoint.....	237
Table 123.	SMI Referenced Properties/Methods for CIM_LogicalDisk (LD for FS).....	239
Table 124.	SMI Referenced Properties/Methods for CIM_NetworkPort.....	240
Table 125.	SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS).....	242
Table 126.	SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial).....	243
Table 127.	SMI Referenced Properties/Methods for CIM_SystemDevice (Storage Extents).....	245
Table 128.	SMI Referenced Properties/Methods for CIM_SystemDevice (Logical Disks).....	246
Table 129.	SMI Referenced Properties/Methods for CIM_SystemDevice (Network Ports).....	246
Table 130.	SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint.....	246
Table 131.	NetworkPort OperationalStatus.....	255
Table 132.	ProtocolEndpoint OperationalStatus.....	255
Table 133.	Supported Profiles for Self-contained NAS System.....	255
Table 134.	CIM Elements for Self-contained NAS System.....	258
Table 135.	SMI Referenced Properties/Methods for CIM_BindsTo (CIFS or NFS).....	261
Table 136.	SMI Referenced Properties/Methods for CIM_BindsTo (TCP).....	261
Table 137.	SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint.....	261
Table 138.	SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level).....	262
Table 139.	SMI Referenced Properties/Methods for CIM_ComputerSystem (File Server).....	263
Table 140.	SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort).....	265
Table 141.	SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort).....	265
Table 142.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS).....	266
Table 143.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP).....	266
Table 144.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (IP).....	266
Table 145.	SMI Referenced Properties/Methods for CIM_HostedAccessPoint (LAN).....	267

Table 146. SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint.....	267
Table 147. SMI Referenced Properties/Methods for CIM_LANEndpoint .....	268
Table 148. SMI Referenced Properties/Methods for CIM_LogicalDisk (Disk for FS) .....	270
Table 149. SMI Referenced Properties/Methods for CIM_NetworkPort.....	271
Table 150. SMI Referenced Properties/Methods for CIM_ProtocolEndpoint (CIFS or NFS) .....	273
Table 151. SMI Referenced Properties/Methods for CIM_SystemDevice (Logical Disks) .....	274
Table 152. SMI Referenced Properties/Methods for CIM_SystemDevice (Network Ports).....	275
Table 153. SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint .....	275

## List of Figures

Figure 1.	Experimental Maturity Level Tag.....	vi
Figure 2.	Implemented Maturity Level Tag.....	vi
Figure 3.	Stable Maturity Level Tag.....	vii
Figure 4.	Deprecated Tag.....	vii
Figure 5.	File Export Instance.....	8
Figure 6.	File Export Manipulation Subprofile Instance .....	18
Figure 7.	Capabilities and Settings for Exported File Share Creation .....	21
Figure 8.	File Storage Instance .....	69
Figure 9.	Cascading File Storage .....	71
Figure 10.	Filesystem Instance.....	76
Figure 11.	LocalFileSystem Creation Instance Diagram .....	108
Figure 12.	Capabilities and Settings for Filesystem Creation .....	113
Figure 13.	Filesystem Quotas Instance Diagram .....	196
Figure 14.	NAS Head Profiles and Subprofiles .....	216
Figure 15.	NAS Head Instance.....	217
Figure 16.	NAS Storage Instance.....	219
Figure 17.	NAS Head Cascading Support Instance .....	221
Figure 18.	Self-Contained NAS Profile and Subprofiles .....	250
Figure 19.	Self-Contained NAS Instance.....	251
Figure 20.	NAS Storage Instance.....	253
Figure 21.	State Transitions From LogicalDisk to FileShare .....	278





# 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, Part 1 Common Architecture*
- *Storage Management Technical Specification, Part 2 Common Profiles*
- *Storage Management Technical Specification, Part 3 Block Devices*
- *Storage Management Technical Specification, Part 4 File Systems*
- *Storage Management Technical Specification, Part 5 Fabric*
- *Storage Management Technical Specification, Part 6 Host Elements*
- *Storage Management Technical Specification, Part 7 Information Lifecycle Management*
- *Storage Management Technical Specification, Part 8 Media Libraries*

## Acknowledgements

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 Communications Systems .....John Crandall

EMC Corporation .....Kamesh Aiyer

.....Edgar St. Pierre

Hewlett-Packard.....Steve Peters

Hitachi Data Systems.....Steve Quinn

IBM.....Duane Baldwin

.....Jack Gelb

.....Mike Walker

iStor Networks, Inc. ....Scott Baker

Network Appliance .....Alan Yoder

Sun Microsystems.....Mark Carlson

Symantec .....Steve Hand

.....Paul von Behren

## 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.

## 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.

## Scope

- 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]

ISO/IEC 24775 Storage Management

### **2.3 References under development**

*Storage Management Technical Specification, Part 1 Common Architecture*

*Storage Management Technical Specification, Part 2 Common Profiles*

*Storage Management Technical Specification, Part 3 Block Devices*

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

## Normative References

## 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* 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**

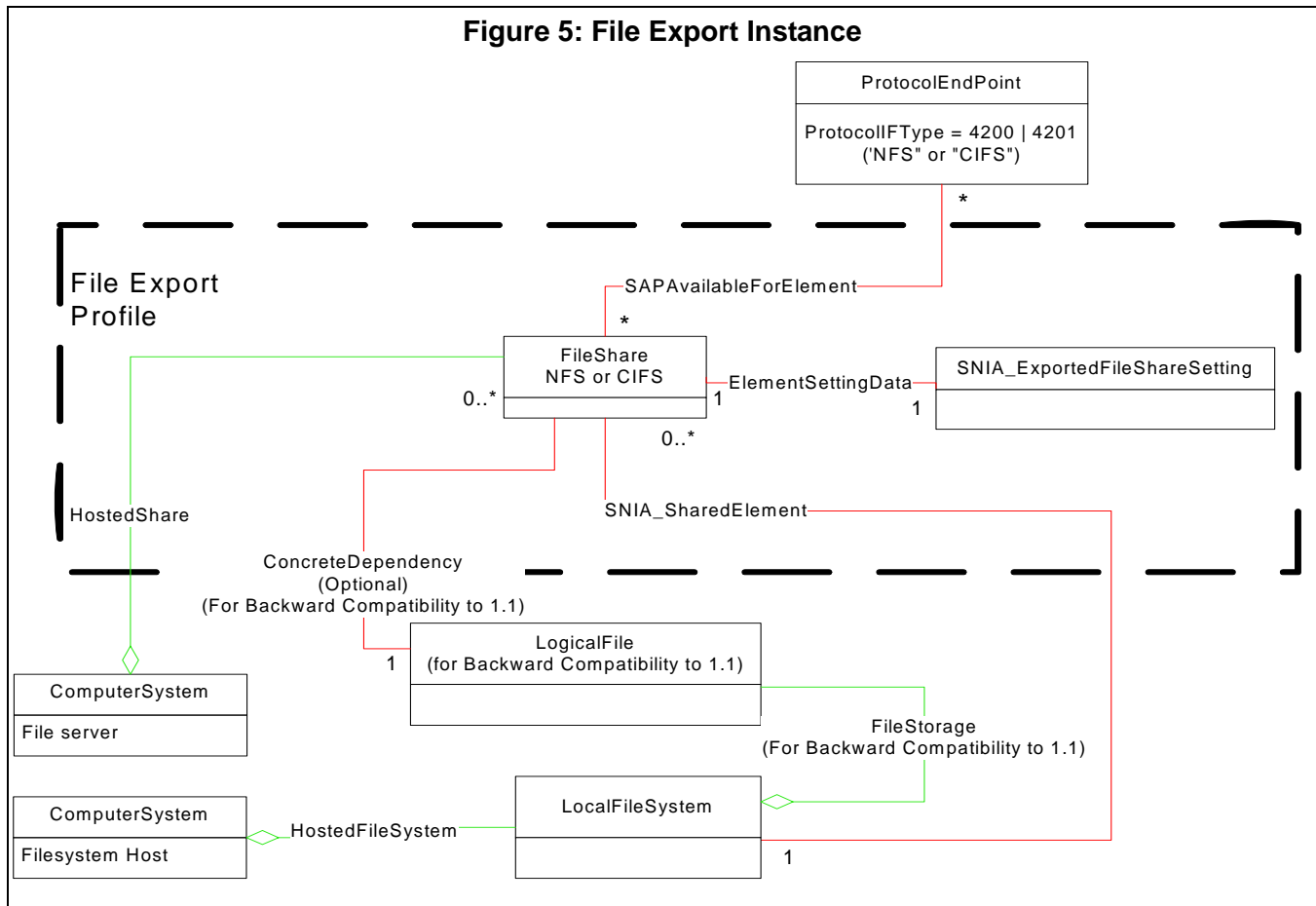
---

---

**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 1 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 2: Supported Profiles for File Export**

Registered Profile Names	Mandatory	Version
Indication	Yes	1.2.0

## 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 1.2.0

## 4.8 CIM Elements

**Table 3: CIM Elements for File Export**

Element Name	Requirement	Description
CIM_ConcreteDependency (4.8.1)	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.
CIM_ElementSettingData (FileShare) (4.8.2)	Mandatory	Associates a FileShare and ExportedFileShareSetting elements.
SNIA_ExportedFileShareSetting (Setting) (4.8.3)	Mandatory	The configuration settings for an Exported FileShare that is a setting for a FileShare available for exporting.
CIM_FileShare (Exported File Share) (4.8.4)	Mandatory	Represents the sharing characteristics of a particular file element.
CIM_HostedShare (4.8.5)	Mandatory	Represents that a shared element is hosted by a Computer System.
CIM_SAPAvailableForElement (4.8.6)	Mandatory	Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
SNIA_SharedElement (4.8.7)	Mandatory	Associates a FileShare to the LocalFileSystem on which it is based.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.CIM_FileShare::OperationalStatus[*] <> PreviousInstance.CIM_FileShare::OperationalStatus[*]	Optional	Experimental CQL - 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.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.

### 4.8.1 CIM\_ConcreteDependency

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: 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

Class Mandatory: 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 SNIA\_ExportedFileShareSetting (Setting)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 6 describes class SNIA\_ExportedFileShareSetting (Setting).

**Table 6: SMI Referenced Properties/Methods for SNIA\_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').
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

Class Mandatory: 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
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

Class Mandatory: 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

Class Mandatory: 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 SNIA\_SharedElement

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 10 describes class SNIA\_SharedElement.

**Table 10: SMI Referenced Properties/Methods for SNIA\_SharedElement**

Properties	Flags	Requirement	Description & Notes
SystemElement		Mandatory	The LocalFileSystem that is exporting some contained file or directory as a FileShare.

**Table 10: SMI Referenced Properties/Methods for SNIA\_SharedElement**

Properties	Flags	Requirement	Description & Notes
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.

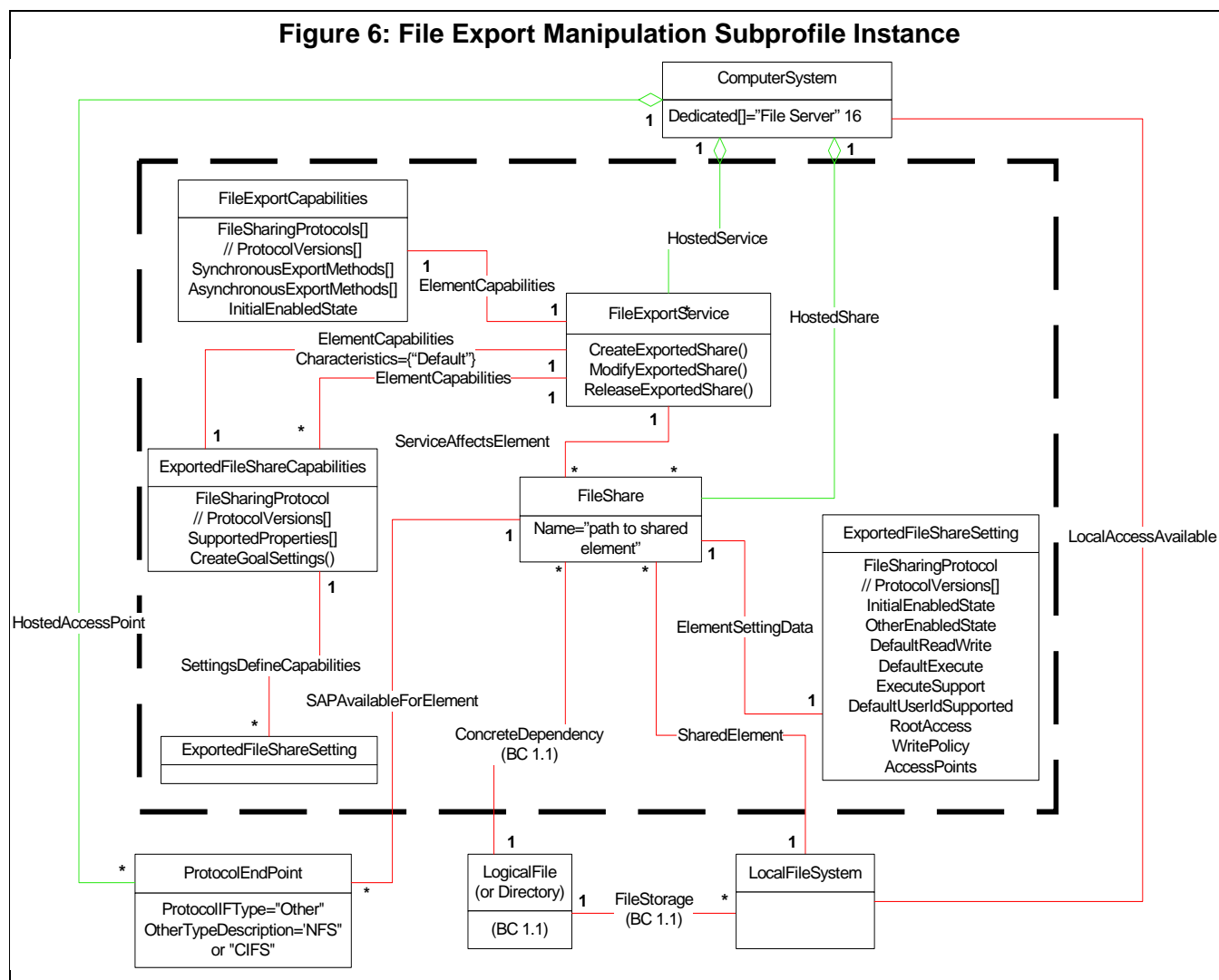
In SMI-S we use a FileShare element to represent the externally accessible file share. A SharedElement association will exist between the FileShare and the LocalFileSystem. The FileShare.Name property indicates the shared object (it is the filesystem-specific path to the contained file or directory that 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.

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 the `SettingsDefineCapabilities.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 generate an array of embedded `SettingData` elements as the output `SupportedGoalSettings` parameter. However, in this profile, we only use 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:

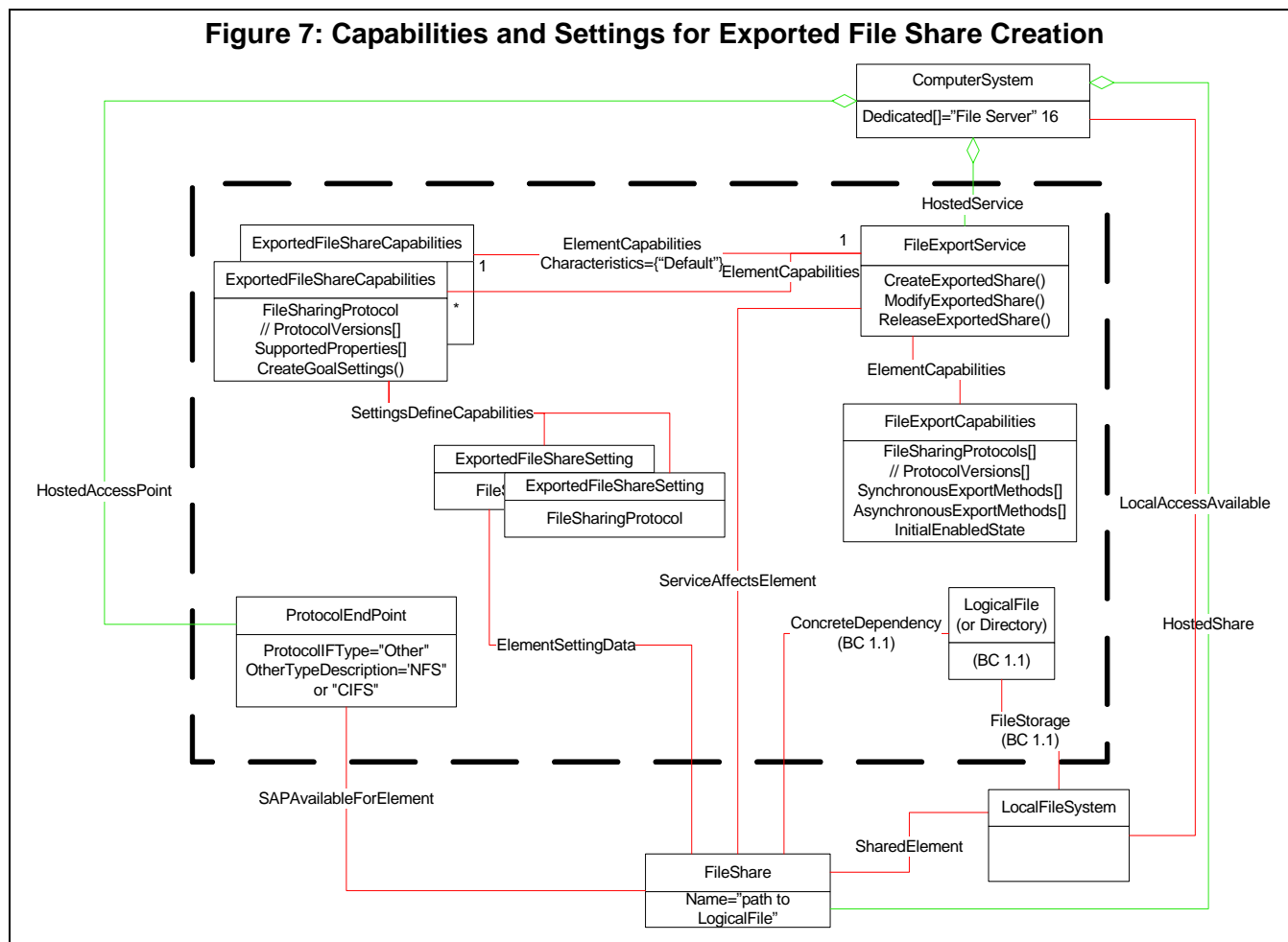
- `HostedFileShare` association between the `FileShare` and the file server `ComputerSystem` that hosts it.
- `SharedElement` association between the `FileShare` and the `LocalFileSystem`. In addition, the `FileShare` element specifies the pathname for the shared element (file or directory) relative 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*) 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:, "Capabilities and Settings for Exported File Share Creation" illustrates an instance diagram showing the elements that shall exist for supporting fileshare 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.

**Table 12: Operational Status for File Server ComputerSystem**

Primary OperationalStatus	Description
10 "Stopped"	The ComputerSystem cannot be accessed operationally because it is stopped -- if this did not happen because of operator intervention or happened in real-time, the OperationalStatus would have been "Lost Communication" rather than "Stopped".
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 13: Supported Profiles for File Export Manipulation**

Registered Profile Names	Mandatory	Version
Job Control	No	1.2.0
File Export	Yes	1.2.0
Security	No	1.2.0
Indication	Yes	1.2.0

## 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) <i>(for bc to 1.1)</i> 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	<b>EmbeddedInstance ("SNIA_ExportedFileShareSetting")</b> 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	<b>EmbeddedInstance ("SNIA_ExportedFileShareSetting")</b> 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".
<b>Normal Return</b>			
Status		uint32	"Success", "Failed", "Timeout", "Alternative Proposed"
<b>Error Returns</b>			
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	<p>An end user relevant name for the FileShare being created. If NULL, then a system-supplied default name can be used.</p> <p>The value shall be stored in the 'ElementName' property for the created element.</p>
Comment	IN	string	<p>An end user relevant comment for the FileShare being created. If NULL, then a system-supplied default comment can be used.</p> <p>The value shall be stored in the 'Description' property for the created element.</p>
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	<p>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.</p> <p>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.</p> <p>If SharedElementPath is NULL or is the empty string, it indicates the \"root\" directory of the file system indicated by Root.</p> <p>The value shall be stored in the 'Name' property for the created element.</p>



**Table 16: Parameters for Extrinsic Method FileExportServices.CreateExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
Goal	IN, OUT, EI	string	<p><b><i>EmbeddedInstance ("SNIA_ExportedFileShareSetting")</i></b></p> <p>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.</p>
TheShare	OUT, REF	CIM_FileShare	If successful, this returns a reference to the created file share.
DefaultUserId	IN, OUT, REF, NULL allowed,	CIM_identity	<p>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.</p> <p>A default user id per share is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.</p>
RootAccessHosts[]	IN, OUT, URI, NULL allowed	string	<p>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.</p> <p>Root Access is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.</p>

**Table 16: Parameters for Extrinsic Method FileExportServices.CreateExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
AccessPointPorts[]	IN, OUT, REF, NULL Allowed	CIM_Service AccessPoints	<p>An array of references to the ProtocolEndpoints that can connect to this Share, if the SNIA_ExportedFileShareSetting.AccessPoints property is set to 'Named Ports'.</p> <p>If the parameter is NULL, all access points will be denied access, effectively overriding the value of the property ExportedFileShareSetting.AccessPoints.</p> <p>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.</p> <p>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.</p> <p>The CIFS protocol does not support multiple ProtocolEndpoints, so this is ignored if the Goal specifies creating a CIFSShare.</p>
<b>Normal Return</b>			
Status	OUT	uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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, in this subprofile we do 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	<p>A reference indicating a LocalFileSystem element whose sub-element is being exported. In the <code>ModifyExportedShare</code> 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).</p> <p>If Root is NULL on input it is ignored.</p> <p>As an OUT parameter, a reference to the LocalFileSystem is returned.</p>
SharedElement Path	IN, OUT	string	<p>A string representing a path to the shared element from the root directory of the LocalFileSystem indicated by Root.</p> <p>The <code>ModifyExportedShare</code> 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.</p> <p>If <code>SharedElementPath</code> is NULL, it indicates no change to the current path. If <code>SharedElementPath</code> consists of a single empty string, it indicates the root directory of the FileSystem indicated by Root.</p> <p>As an OUT parameter, the current path is returned.</p> <p>The value shall be stored in the 'Name' property for the created element.</p>

**Table 17: Parameters for Extrinsic Method FileExportServices.ModifyExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
Goal	IN, OUT, EI	string	<p><b><i>EmbeddedInstance ("SNIA_ExportedFileShareSetting")</i></b></p> <p>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.</p> <p>As an OUT parameter, the current Setting is returned.</p>
TheShare	IN, OUT, REF	CIM_FileShare	<p>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.</p>
DefaultUserId	IN, OUT, REF, NULL allowed,	CIM_identity	<p>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.</p> <p>A default user per share is not supported by the CIFS Protocol so this is ignored if the file share is a CIFSShare.</p>

**Table 17: Parameters for Extrinsic Method FileExportServices.ModifyExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
RootAccessHosts[]	IN, OUT, URI, NULL allowed	string	<p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>The provider is expected to surface this access using the Authorization subprofile.</p> <p>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.</p> <p>Root Access is not supported by the CIFS Protocol so this is ignored if the Goal specifies creating a CIFSShare.</p>

**Table 17: Parameters for Extrinsic Method FileExportServices.ModifyExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
AccessPointPorts[]	IN, OUT, REF, NULL Allowed	CIM_Service AccessPoints	<p>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.</p> <p>If the parameter is NULL, all access points will be denied access, effectively overriding the value of the property SNIA_ExportedFileShareSetting.AccessPoints.</p> <p>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.</p> <p>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.</p> <p>The CIFS protocol does not support multiple ProtocolEndpoints, so this is ignored if the Goal specifies creating a CIFSShare.</p>
InUseOptions	IN	uint16	<p>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.</p> <p>This option is only relevant if the FileShare needs to be made unavailable while the request is being executed.</p>
WaitTime	IN	uint16	<p>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.</p> <p>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.</p>

**Table 17: Parameters for Extrinsic Method FileExportServices.ModifyExportedShare**

Parameter Name	Qualifier	Type	Description & Notes
<b>Normal Return</b>			
Status	OUT	uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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.
<b>Normal Return</b>			
Status	OUT	uint32	ValueMap{}, Values{}  "Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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.
- We use `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.
//  proppnames: 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[] $proppnames,        // 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,
    ///                                     $fescapability);
    //
    // Get a FileExportService via the HostedService association to
    // the file server ComputerSystem
    //
    $feservice = Associators($server,
                            "CIM_HostedService",
                            "SNIA_FileExportService",

```

## File Export Manipulation Subprofile

```
        "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];

//

```

## File Export Manipulation Subprofile

```
// 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

```

## File Export Manipulation Subprofile

```
//      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");

```

## File Export Manipulation Subprofile

```
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 ($newsetting==NULL)
#i = 0;
while ($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:

```

## File Export Manipulation Subprofile

```
// 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 1.2.0

## 5.8 CIM Elements

**Table 20: CIM Elements for File Export Manipulation**

Element Name	Requirement	Description
SNIA_FileExportService (5.8.1)	Mandatory	The File Export Service provides the methods to create and export file elements as shares.
CIM_HostedService (5.8.2)	Mandatory	Associates the File Export Service to the hosting File Server Computer System.
SNIA_FileExportCapabilities (FES Configuration) (5.8.3)	Mandatory	This element represents the management capabilities of the File Export Service.
CIM_ElementCapabilities (FES Configuration) (5.8.4)	Mandatory	Associates the File Export Service to the FileExportCapabilities element that describes the service capabilities.
SNIA_ExportedFileShareCapabilities (FES Capabilities) (5.8.5)	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.
SNIA_ElementCapabilities (FES Capabilities) (5.8.6)	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.
SNIA_ExportedFileShareSetting (Pre-defined) (5.8.7)	Optional	This element represents a pre-defined configuration settings for exported FileShares that is used to define a Capabilities element associated with the FileExportService.
SNIA_SettingsDefineCapabilities (Pre-defined) (5.8.8)	Optional	Represents the association between a ExportedFileShareCapabilities and a pre-defined ExportedFileShareSetting element that specifies what the Capabilities can support.
SNIA_ExportedFileShareSetting (FileShare Setting) (5.8.9)	Mandatory	<p>The configuration settings for an Exported FileShare; i.e., a setting for a FileShare available for exporting.</p> <p>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 pre-existent FileShares. All of these will be surfaced.</p>
SNIA_FileShare (Exported File Share) (5.8.10)	Mandatory	Represents the sharing characteristics of a particular file element.

**Table 20: CIM Elements for File Export Manipulation**

Element Name	Requirement	Description
SNIA_HostedShare (5.8.11)	Mandatory	Represents that a shared element is hosted by a ComputerSystem.
CIM_ServiceAffectsElement (5.8.12)	Mandatory	Associates the File Export Service to the elements that the service manages (such as a FileShare configured for exporting a LogicalFile).
CIM_ElementSettingData (FileShare Setting) (5.8.13)	Mandatory	Associates a FileShare and ExportedFileShareSetting elements.
CIM_LogicalFile (Subelement) (5.8.14)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>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 the SMI Specification Release 1.1.</p>
SNIA_SharedElement (5.8.15)	Mandatory	Associates a FileShare to the LocalFileSystem on which it is based.
CIM_FileStorage (Subelement) (5.8.16)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>Represents that a file or directory that is made available for export is contained by a LocalFileSystem specified as a dangling reference.</p>
CIM_ConcreteDependency (5.8.17)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>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 FileExport Subprofile v1.1.</p>
CIM_SAPAvailableForElement (5.8.18)	Mandatory	Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA SNIA_FileShare	Mandatory	<p>Creation of an exported file share.</p> <p>This indication returns the newly created FileShare.</p>

**Table 20: CIM Elements for File Export Manipulation**

Element Name	Requirement	Description
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.SNIA_FileShare::OperationalStatus[*] <> PreviousInstance.SNIA_FileShare::OperationalStatus[*]	Optional	Experimental CQL - 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.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.

**5.8.1 SNIA\_FileExportService**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 21 describes class SNIA\_FileExportService.

**Table 21: 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.



**Table 21: SMI Referenced Properties/Methods for SNIA\_FileExportService**

Properties	Flags	Requirement	Description & Notes
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.

**5.8.2 CIM\_HostedService**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 22 describes class CIM\_HostedService.

**Table 22: SMI Referenced Properties/Methods for CIM\_HostedService**

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

**5.8.3 SNIA\_FileExportCapabilities (FES Configuration)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 23 describes class SNIA\_FileExportCapabilities (FES Configuration).

**Table 23: 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.

**Table 23: SMI Referenced Properties/Methods for SNIA\_FileExportCapabilities (FES Configuration)**

Properties	Flags	Requirement	Description & Notes
FileSharingProtocol		Mandatory	<p>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.</p> <p>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.</p>
ProtocolVersions		Optional	<p>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.</p> <p>At this point there is no standard mechanism for naming versions of CIFS or NFS, so this property is optional in this subprofile.</p>
SupportedSynchronousMethods	N	Mandatory	<p>An array listing the extrinsic methods of the FileExportService that can be called synchronously.</p> <p>Note: Every supported method shall be listed either in this property or in the SupportedAsynchronousMethods array property.</p>
SupportedAsynchronousMethods	N	Mandatory	<p>An array listing the extrinsic methods of the FileExportService that can be called synchronously.</p> <p>Note: Every supported method shall be listed either in this property or in the SupportedSynchronousMethods array property.</p>
InitialEnabledState		Optional	This represents the state of initialization of a FileShare on initial creation.

**5.8.4 CIM\_ElementCapabilities (FES Configuration)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 24 describes class CIM\_ElementCapabilities (FES Configuration).

**Table 24: SMI Referenced Properties/Methods for CIM\_ElementCapabilities (FES Configuration)**

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

#### 5.8.5 SNIA\_ExportedFileShareCapabilities (FES Capabilities)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 25 describes class SNIA\_ExportedFileShareCapabilities (FES Capabilities).

**Table 25: 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	<p>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.</p> <p>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.</p>
SupportedProperties		Mandatory	<p>This is the list of configuration properties (of ExportedFileShareSetting) that are supported for specification at creation time by this Capabilities element.</p> <p>Properties that can appear in this array are:            "DefaultReadWrite" ("2"), "DefaultExecute" ("3"),            "DefaultUserId" ("4"), "RootAccess" ("5"), "WritePolicy" ("6"),            "AccessPoints" ("7"), and "InitialEnabledState" ("8").</p>

**Table 25: SMI Referenced Properties/Methods for SNIA\_ExportedFileShareCapabilities (FES Capabilities)**

Properties	Flags	Requirement	Description & Notes
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].

**5.8.6 SNIA\_ElementCapabilities (FES Capabilities)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 26 describes class SNIA\_ElementCapabilities (FES Capabilities).

**Table 26: 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.
ManagedElement		Mandatory	The FileExportService.

**5.8.7 SNIA\_ExportedFileShareSetting (Pre-defined)**

Created By: Static\_or\_External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 27 describes class SNIA\_ExportedFileShareSetting (Pre-defined).

**Table 27: 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).
ProtocolVersions		Optional	<p>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.</p> <p>At this point there is no standard mechanism for naming versions of CIFS or NFS, so this is being made optional.</p>
InitialEnabledState		Optional	<p>This indicates the enabled/disabled states initially set for a created FileShare by this Setting element.</p> <p>Valid values are "1" ("Other"), "2" ("Enabled"), "3" ("Disabled"), "7" ("In Test"), "8" ("Deferred") or "9" ("Quiesce")</p>
OtherEnabledState		Optional	A vendor-specific description of the initial enabled state of a created fileshare if InitialEnabledState=1("Other").
DefaultUserIdSupported		Optional	<p>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.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Valid values are "2" ("No Default User Id"), "3" ("System-Specified Default User Id") or "4" ("Share-Specified Default User Id").</p>
RootAccess		Optional	<p>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.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Valid values are "2" ("No Root Access") or "3" ("Allow Root Access").</p>

**Table 27: SMI Referenced Properties/Methods for SNIA\_ExportedFileShareSetting (Pre-defined)**

Properties	Flags	Requirement	Description & Notes
AccessPoints		Optional	<p>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.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Valid values are "2" ("None"), "3" ("Service Default"), "4" ("All") or "5" ("Named Points").</p>
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
DefaultReadWrite		Optional	<p>Indicates the default privileges that are supported for read and write authorization when creating or modifying a FileShare using this Setting element.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Not Specified in this version of the Profile</p>
DefaultExecute		Optional	<p>Indicates the default privileges that are supported for execute authorization when creating or modifying a FileShare using this Setting element.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Not Specified in this version of the Profile</p>
ExecuteSupport		Optional	<p>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).</p> <p>Not Specified in this version of the Profile</p>
WritePolicy		Optional	<p>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.</p> <p>This policy may be overridden or surfaced using the Policy sub-profile.</p> <p>Not Specified in this version of the Profile</p>

### 5.8.8 SNIA\_SettingsDefineCapabilities (Pre-defined)

Created By: Static\_or\_External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 28 describes class SNIA\_SettingsDefineCapabilities (Pre-defined).

**Table 28: 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.

### 5.8.9 SNIA\_ExportedFileShareSetting (FileShare Setting)

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 29 describes class SNIA\_ExportedFileShareSetting (FileShare Setting).

**Table 29: 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.

**Table 29: SMI Referenced Properties/Methods for SNIA\_ExportedFileShareSetting (FileShare Setting)**

Properties	Flags	Requirement	Description & Notes
ProtocolVersions		Mandatory	<p>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.</p> <p>At this point there is no standard mechanism for naming versions of CIFS or NFS, so this is being made optional.</p>
InitialEnabledState	N	Optional	<p>This indicates the enabled/disabled states initially set for a created FileShare by this Setting element.</p> <p>Valid values are "1" ("Other"), "2" ("Enabled"), "3" ("Disabled"), "7" ("In Test"), "8" ("Deferred") or "9" ("Quiesce")</p> <p>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.</p>
OtherEnabledState	N	Optional	<p>This should be filled in if the InitialEnabledState is "1" ("Other").</p>
DefaultUserIdSupported	N	Optional	<p>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.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Valid values are "2" ("No Default User Id"), "3" ("System-Specified Default User Id") or "4" ("Share-Specified Default User Id").</p>
RootAccess	N	Optional	<p>Indicates whether the associated FileShare will support default access privileges to administrative users from specified hosts.</p> <p>Note: The resulting access privileges shall be surfaced using the Authorization subprofile.</p> <p>Valid values are "2" ("No Root Access") or "3" ("Allow Root Access").</p>



**Table 29: SMI Referenced Properties/Methods for SNIA\_ExportedFileShareSetting (FileShare Setting)**

Properties	Flags	Requirement	Description & Notes
AccessPoints	N	Optional	<p>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.</p> <p>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 sub-profile.</p> <p>Valid values are "2" ("None"), "3" ("Service Default"), "4" ("All") or "5" ("Named Points").</p>
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

**5.8.10 SNIA\_FileShare (Exported File Share)**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 30 describes class SNIA\_FileShare (Exported File Share).

**Table 30: 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.

**Table 30: SMI Referenced Properties/Methods for SNIA\_FileShare (Exported File Share)**

Properties	Flags	Requirement	Description & Notes
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
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

**5.8.11 SNIA\_HostedShare**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 31 describes class SNIA\_HostedShare.

**Table 31: 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.

**5.8.12 CIM\_ServiceAffectsElement**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 32 describes class CIM\_ServiceAffectsElement.

**Table 32: 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.
OtherElementEffects Descriptions		Mandatory	A description of other element effects that this association might be exposing.
AffectedElement		Mandatory	The FileShare.
AffectingElement		Mandatory	The FileExportService.

### 5.8.13 CIM\_ElementSettingData (FileShare Setting)

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 33 describes class CIM\_ElementSettingData (FileShare Setting).

**Table 33: 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 sub-profile 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"

**5.8.14 CIM\_LogicalFile (Subelement)**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: BC1.1

Table 34 describes class CIM\_LogicalFile (Subelement).

**Table 34: 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.
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.

**5.8.15 SNIA\_SharedElement**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 35 describes class SNIA\_SharedElement.

**Table 35: 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.

**5.8.16 CIM\_FileStorage (Subelement)**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: BC1.1

Table 36 describes class CIM\_FileStorage (Subelement).

**Table 36: 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.

**5.8.17 CIM\_ConcreteDependency**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: BC1.1

Table 37 describes class CIM\_ConcreteDependency.

**Table 37: 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.

**5.8.18 CIM\_SAPAvailableForElement**

Created By: Extrinsic: CreateExportedShare

Modified By: Extrinsic: ModifyExportedShare

Deleted By: Extrinsic: ReleaseExportedShare

Class Mandatory: Mandatory

Table 38 describes class CIM\_SAPAvailableForElement.

**Table 38: 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.

## **EXPERIMENTAL**

---

**STABLE****Clause 6: File Storage Profile****6.1 Description****6.1.1 Synopsis**

Profile Name: File Storage

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: N/A

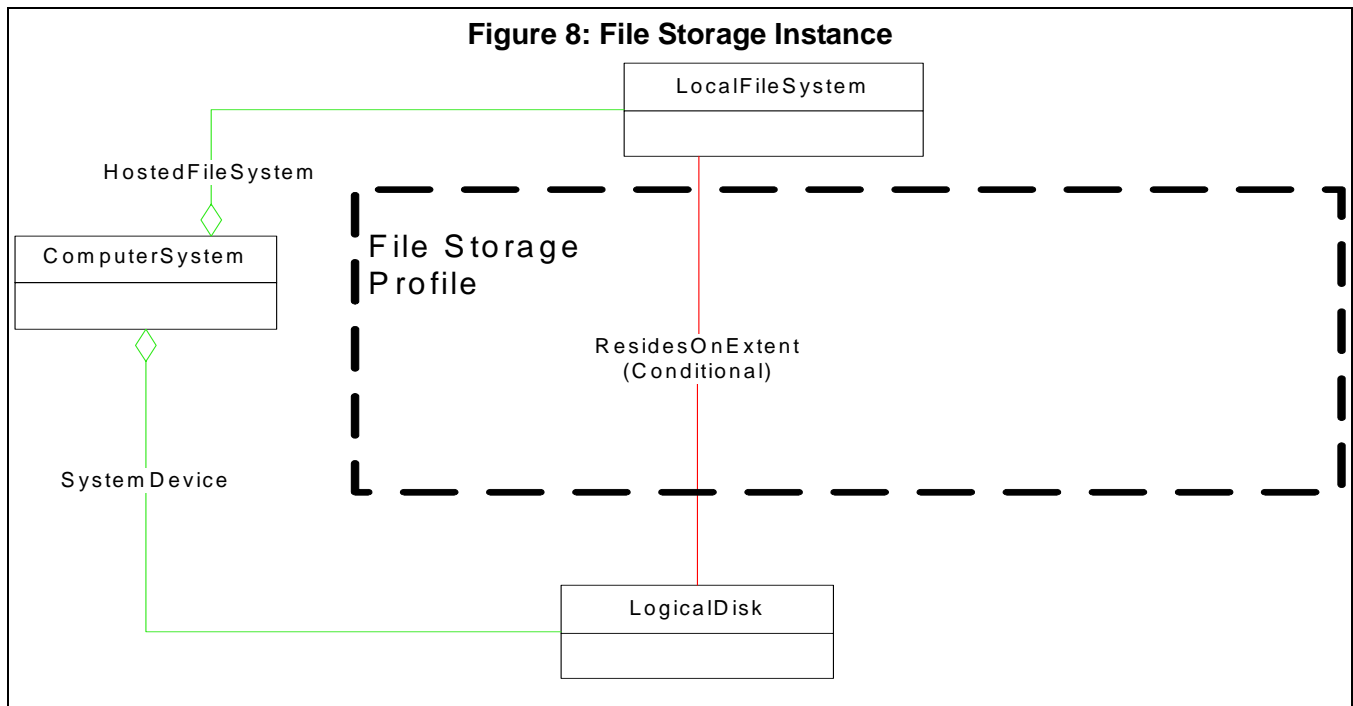
Scoping Class: ComputerSystem

**6.1.2 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.

**6.1.3 Implementation**

Figure 8 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.



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.

## **6.2 Health and Fault Management Consideration**

None.

---

---

## **EXPERIMENTAL**

## **6.3 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.

### **6.3.1 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 9 illustrates the constructs to be used to model this cascading relationship.



Figure 9: Cascading File Storage

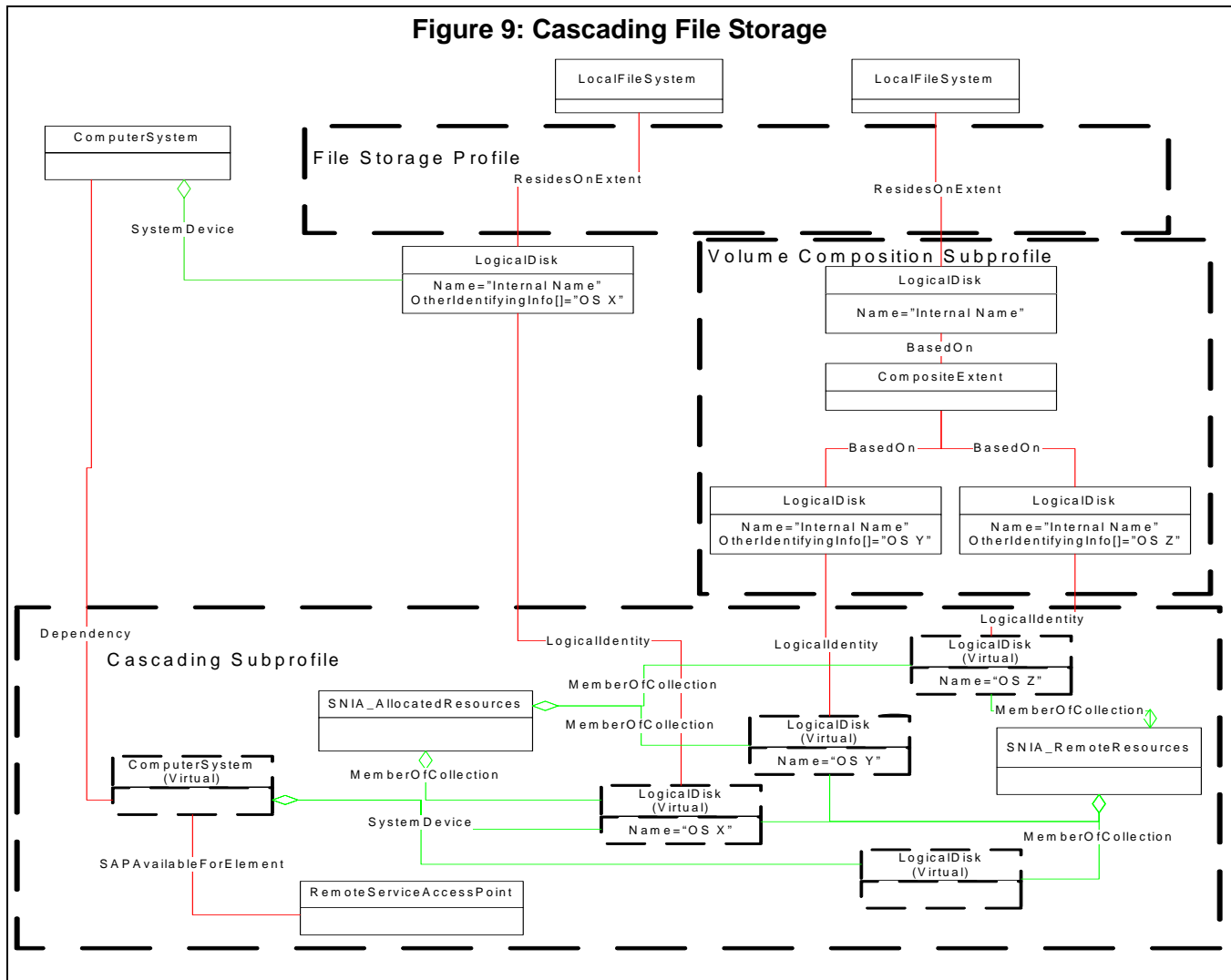


Figure 9 shows 2 filesystems (**LocalFileSystem**). Both reside on one **LogicalDisk**. But the **LogicalDisk** on the right is a composite of lower level **LogicalDisks**. The storage that is imported from the remote profile are **LogicalDisks** at the lowest level of the Filesystem Profile. So, in 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 39 provides the specific cascading information for cascading file storage.

**Table 39: Cascaded Storage**

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

### 6.3.2 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.

### 6.3.3 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

---

## 6.4 Supported Profiles, Subprofiles, and Packages

Not defined in this standard.

## 6.5 Methods of the Profile

### 6.5.1 Extrinsic Methods of the Profile

None

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

### 6.5.2 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

## 6.6 Client Considerations and Recipes

None.

## 6.7 Registered Name and Version

File Storage version 1.2.0

## 6.8 CIM Elements

**Table 40: CIM Elements for File Storage**

Element Name	Requirement	Description
CIM_ResidesOnExtent (6.8.1)	Conditional	Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.

### 6.8.1 CIM\_ResidesOnExtent

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: NASHead|SCNAS

Table 41 describes class CIM\_ResidesOnExtent.

**Table 41: 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 7: Filesystem Profile****7.1 Description****7.1.1 Synopsis**

Profile Name: Filesystem

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: LocalFileSystem

Scoping Class: ComputerSystem

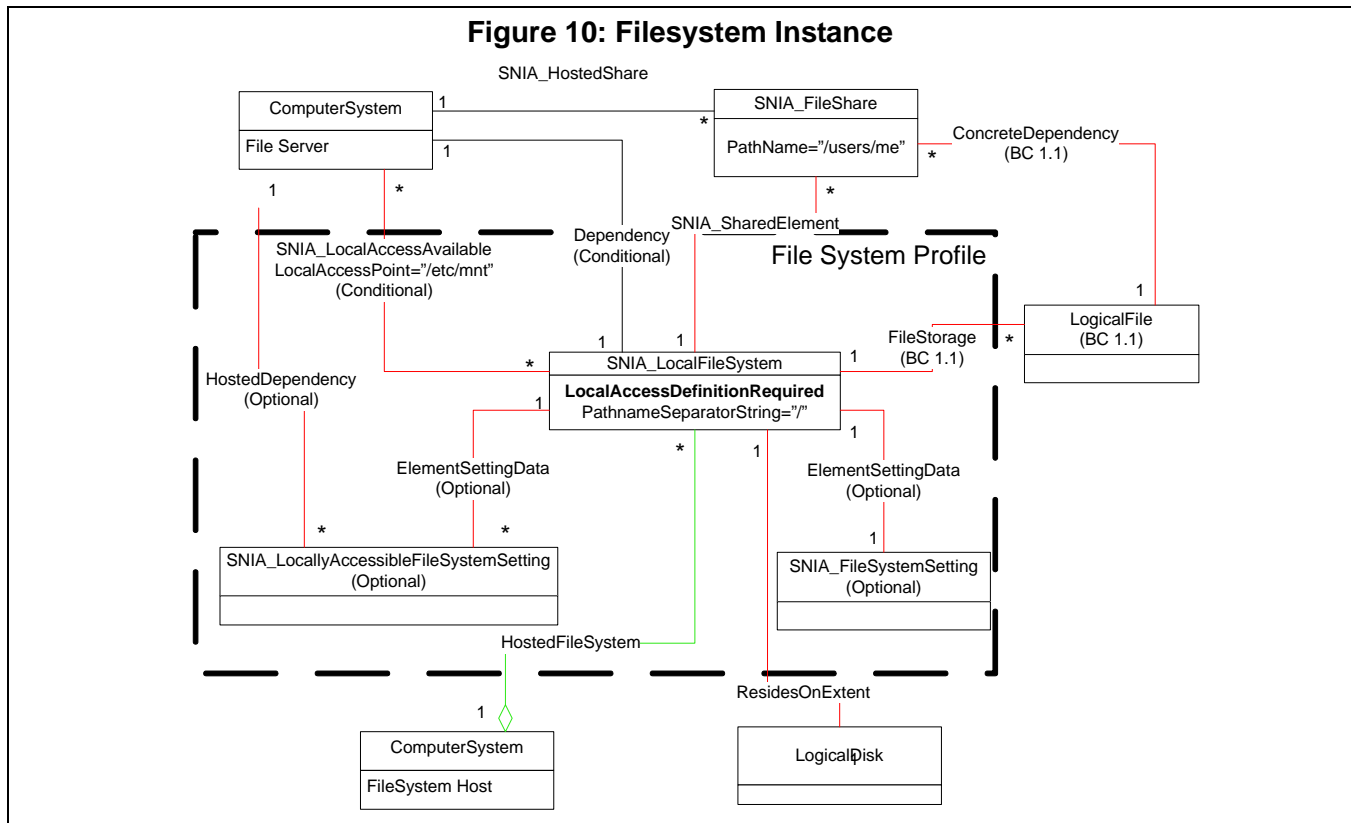
The Filesystem Profile is a subprofile for autonomous profiles that support filesystems. Specifically, in this release of SMI-S, this includes the NAS Head and the Self-Contained NAS profiles. A number of other profiles and subprofiles make use of elements of the Filesystem profile and will be referred 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.

**7.1.2 Instance Diagrams**

Figure 10 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



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 32: Multiple Computer System Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles*.

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

### 7.1.2.1 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 we are unable to represent that 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

### 7.1.2.2 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

## 7.2 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 53: SMI Referenced Properties/Methods for SNIA\_LocalFileSystem)



### 7.2.1 OperationalStatus for Filesystems

**Table 42: 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 happen 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.

### 7.3 Cascading Considerations

None.

## 7.4 Supported Profiles, Subprofiles, and Packages

**Table 43: Supported Profiles for Filesystem**

Registered Profile Names	Mandatory	Version
Indication	Yes	1.2.0

## 7.5 Methods of the Profile

### 7.5.1 Extrinsic Methods of the Profile

None.

### 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: 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.

### 7.6.1 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
```

## Filesystem Profile

```
// 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 occurred>
}
return $FileSystems[]
}
```

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

### 7.6.2 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];
}
```

### 7.6.3 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:
```

## Filesystem Profile

```
// 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);
}
```

### 7.6.4 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;
}

```

#### 7.6.5 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",

```

## Filesystem Profile

```
        "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;
}
```

### 7.6.6 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",
```



## Filesystem Profile

```
        "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;
}
```

### 7.6.7 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 Profile

```
// 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;
}
```

### 7.6.8 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",
```

## Filesystem Profile

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

## EXPERIMENTAL

---

### 7.7 Registered Name and Version

Filesystem version 1.2.0

## 7.8 CIM Elements

**Table 44: CIM Elements for Filesystem**

Element Name	Requirement	Description
CIM_ElementSettingData (FileSystem) (7.8.1)	Optional	Associates a LocalFileSystem to its FileSystemSetting element.
CIM_ElementSettingData (Local Access Required) (7.8.2)	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true. Associates a LocalFileSystem to LocallyAccessibleFileSystemSetting elements, one for each file server that has local access.
CIM_FileStorage (7.8.3)	Mandatory	Associates a LogicalFile (or Directory) to the LocalFileSystem that contains it. This is provided for backward compatibility with SMI-S 1.1
SNIA_LocalAccessAvailable (7.8.4)	Conditional	Conditional requirement: Required if LocalFileSystem.LocalAccessDefinitionRequired=true. Associates a LocalFileSystem to a file server ComputerSystem that can export files or directories as shares.
CIM_HostedDependency (Local Access Required) (7.8.5)	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.
CIM_Dependency (7.8.6)	Mandatory	Associates a ComputerSystem that indicates a directory service that supports the dependent LocalFileSystem.
SNIA_FileSystemSetting (7.8.7)	Optional	This element represents the configuration settings of a filesystem represented by a LocalFileSystem.
CIM_HostedFileSystem (LocalFileSystem) (7.8.8)	Mandatory	Associates a LocalFileSystem to the ComputerSystem that hosts it.
SNIA_LocalFileSystem (7.8.9)	Mandatory	Represents a filesystem in a Filesystem related profile.
CIM_LogicalFile (7.8.10)	Mandatory	In SMI-S 1.1. 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 SMI-S 1.2 profiles to maintain backward compatibility with clients conforming to SMI-S 1.1.

**Table 44: CIM Elements for Filesystem**

Element Name	Requirement	Description
SNIA_LocallyAccessibleFileSystemSetting (7.8.11)	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.
CIM_Dependency (Uses Directory Services From) (7.8.12)	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.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_LocalFileSystem AND SourceInstance.SNIA_LocalFileSystem::OperationalStatus[*] <> PreviousInstance.SNIA_LocalFileSystem::OperationalStatus[*]	Optional	Experimental CQL - 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 SNIA_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.

**7.8.1 CIM\_ElementSettingData (FileSystem)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Optional

Table 45 describes class CIM\_ElementSettingData (FileSystem).

**Table 45: SMI Referenced Properties/Methods for CIM\_ElementSettingData (FileSystem)**

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The LocalFileSystem.

**Table 45: SMI Referenced Properties/Methods for CIM\_ElementSettingData (FileSystem)**

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

**7.8.2 CIM\_ElementSettingData (Local Access Required)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LocalAccessDefinitionRequired

Table 46 describes class CIM\_ElementSettingData (Local Access Required).

**Table 46: 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.

**7.8.3 CIM\_FileStorage**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 47 describes class CIM\_FileStorage.

**Table 47: 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.

**7.8.4 SNIA\_LocalAccessAvailable**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LocalAccessDefinitionRequired

Table 48 describes class SNIA\_LocalAccessAvailable.

**Table 48: SMI Referenced Properties/Methods for SNIA\_LocalAccessAvailable**

Properties	Flags	Requirement	Description & Notes
LocalAccessPoint		Optional	<p>The name used by the file server ComputerSystem to identify the filesystem. Sometimes referred to as a mount-point.</p> <p>For many UNIX-based systems, this will be a qualified full pathname.</p> <p>For Windows systems this could also be the drive letter used for the LogicalDisk that the filesystem is resident on.</p>
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.

**7.8.5 CIM\_HostedDependency (Local Access Required)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LocalAccessDefinitionRequired

Table 49 describes class CIM\_HostedDependency (Local Access Required).

**Table 49: 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.



### 7.8.6 CIM\_Dependency

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory:

Table 50 describes class CIM\_Dependency.

**Table 50: SMI Referenced Properties/Methods for CIM\_Dependency**

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.

### 7.8.7 SNIA\_FileSystemSetting

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 51 describes class SNIA\_FileSystemSetting.

**Table 51: SMI Referenced Properties/Methods for SNIA\_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.

**Table 51: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting**

Properties	Flags	Requirement	Description & Notes
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.
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.

### 7.8.8 CIM\_HostedFileSystem (LocalFileSystem)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 52 describes class CIM\_HostedFileSystem (LocalFileSystem).

**Table 52: 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.

### 7.8.9 SNIA\_LocalFileSystem

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 53 describes class SNIA\_LocalFileSystem.

**Table 53: SMI Referenced Properties/Methods for SNIA\_LocalFileSystem**

Properties	Flags	Requirement	Description & Notes
CSCreationClassName		Mandatory	The CIM class of the hosting ComputerSystem element.
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.

**Table 53: SMI Referenced Properties/Methods for SNIA\_LocalFileSystem**

Properties	Flags	Requirement	Description & Notes
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.
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.
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.

**Table 53: SMI Referenced Properties/Methods for SNIA\_LocalFileSystem**

Properties	Flags	Requirement	Description & Notes
DirectoryServiceUsage		Optional	<p>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:</p> <p>"Not Used" indicates that the filesystem will not support security principal information and so will not require support from a directory service.</p> <p>"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.</p> <p>"Required" indicates that the filesystem supports security principal information and will require support from a directory service. The Dependency association described above must exist.</p>

**7.8.10 CIM\_LogicalFile**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 54 describes class CIM\_LogicalFile.

**Table 54: 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.
FSName		Mandatory	The Name property of the LocalFileSystem that represents the filesystem containing this file.

**Table 54: SMI Referenced Properties/Methods for CIM\_LogicalFile**

Properties	Flags	Requirement	Description & Notes
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.

**7.8.11 SNIA\_LocallyAccessibleFileSystemSetting**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: LocalAccessDefinitionRequired

Table 55 describes class SNIA\_LocallyAccessibleFileSystemSetting.

**Table 55: 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.

**Table 55: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
InitialEnabledState		Optional	<p>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:</p> <p>Enabled (2) indicates that the element will execute commands, will process any queued commands, and will queue new requests.</p> <p>Disabled (3) indicates that the element will not execute commands and will drop any new requests.</p> <p>In Test (7) indicates that the element will be in a test state.</p> <p>Deferred (8) indicates that the element will not process any commands but will queue new requests.</p> <p>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.</p>
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.
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.

**Table 55: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
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 55: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
AttributeCaching		Optional	<p>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.</p> <p>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'.</p>
AttributeCachingObjects		Optional	<p>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).</p>
AttributeCachingTimeMin		Optional	<p>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.</p>

**Table 55: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemServiceSetting**

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.
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.

**Table 55: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemServiceSetting**

Properties	Flags	Requirement	Description & Notes
RootAccessPref		Optional	<p>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.</p> <p>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.</p>

**7.8.12 CIM\_Dependency (Uses Directory Services From)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: DirectoryServiceUsage

Table 56 describes class CIM\_Dependency (Uses Directory Services From).

**Table 56: 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.

**STABLE**



---

---

## EXPERIMENTAL

### Clause 8: Filesystem Manipulation Subprofile

#### 8.1 Description

##### 8.1.1 Synopsis

Profile Name: Filesystem Manipulation

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: FileSystemConfigurationService

Scoping Class: ComputerSystem

##### 8.1.2 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.

##### 8.1.2.1 Backward Compatibility Note

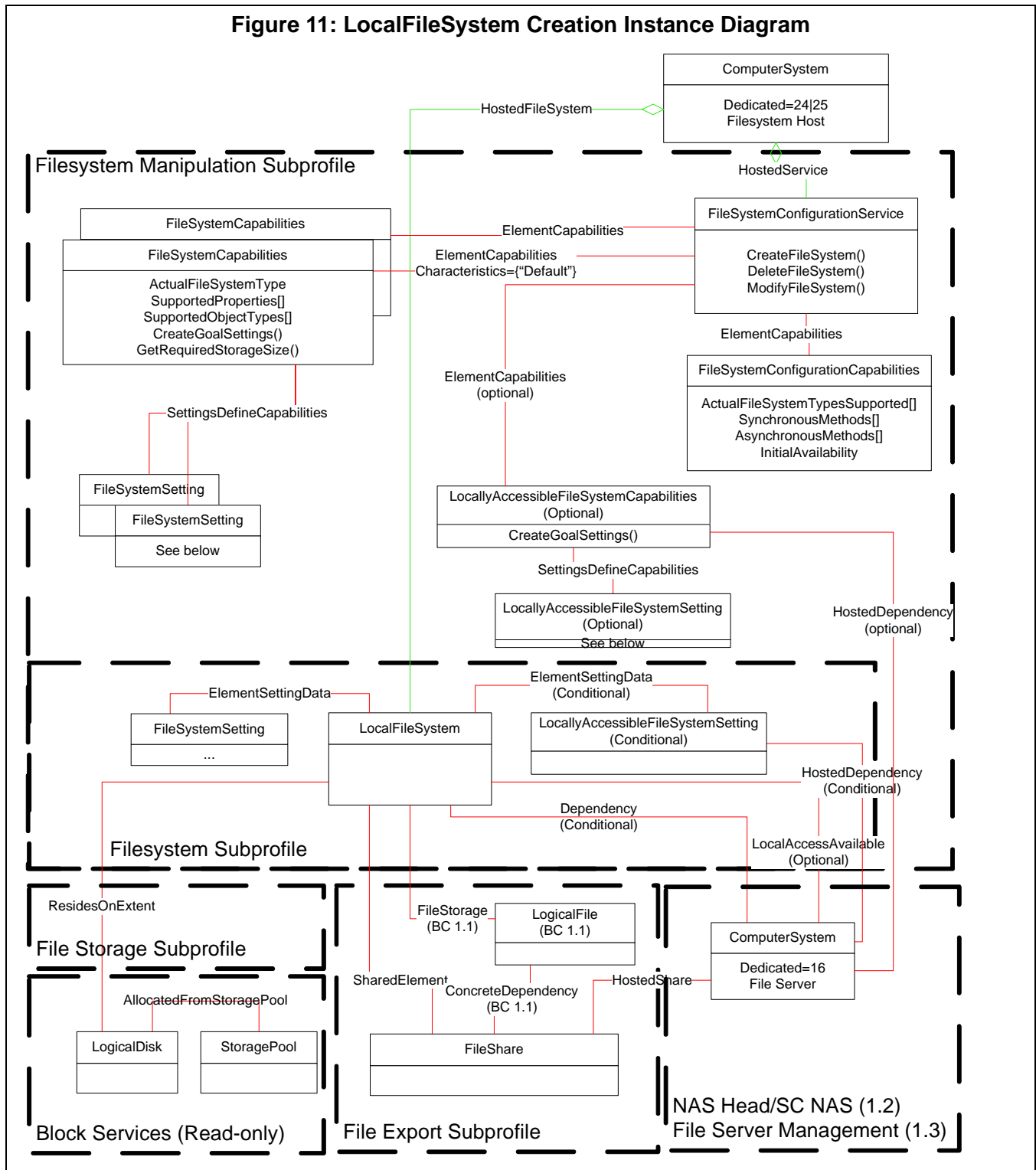
This profile has seen some incompatible changes from SMI-S 1.1. It is still "Experimental". Three major changes to the methods CreateFileSystem and ModifyFileSystem are intended to accommodate requirements from a proposed Hosted Filesystem profile (now postponed to a future release of SMI-S) and to support the local access ("mount") related changes. First, we now allow 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, we now support 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.

##### 8.1.3 Instance Diagrams

##### 8.1.3.1 Filesystem Creation classes and associations

Figure 11 illustrate 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 11: 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 8.5.1, "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, we only use 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 8.5.1.1, "`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.

#### 8.1.3.1.1 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:** We expect 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.

#### 8.1.3.1.2 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 require 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 server, 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.

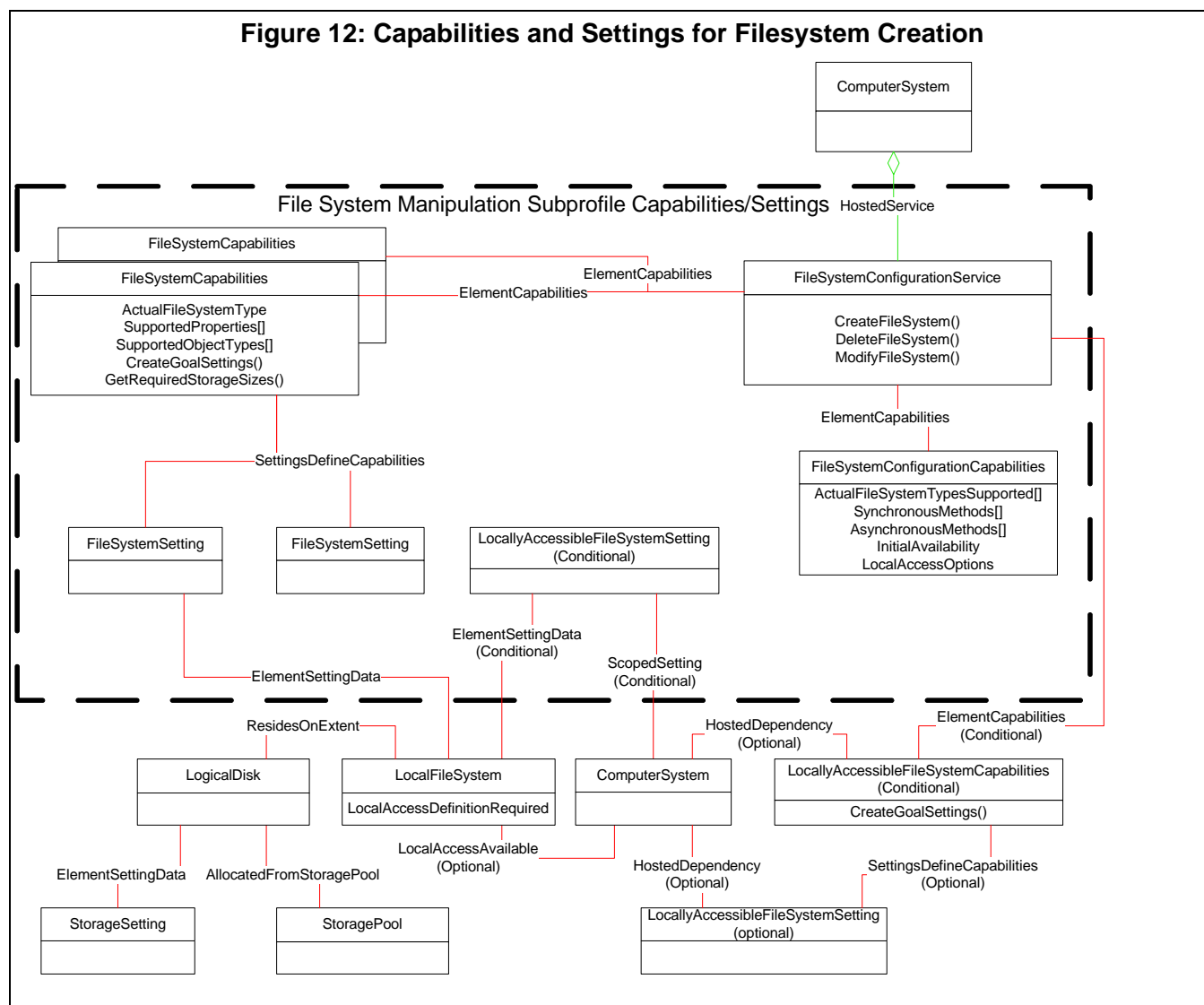
#### 8.1.3.1.3 Summary of this profile

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

- 1) One LocallyAccessibleFileSystemCapabilities scoped to the file server ComputerSystem
- 2) ElementCapabilities association to the FileSystemConfigurationService
- 3) SettingsDefineCapabilities association to LocallyAccessibleFileSystemSetting
- 4) LocallyAccessibleFileSystemSetting for defining LocallyAccessibleFileSystemCapabilities
- 5) HostedDependency (ScopedSetting) association from the File Server ComputerSystem to LocallyAccessibleFileSystemSetting
- 6) A HostedDependency association from the same file server ComputerSystem to the defined LocallyAccessibleFileSystemCapabilities
- 7) 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.
- 8) A Dependency association from the LocalFileSystem element to a file server ComputerSystem.

### 8.1.3.2 Finding FileSystem Configuration Services, Capabilities and Pre-defined Settings

When creating a filesystem the first step is to determine what can be created. Figure 12: "Capabilities and Settings for Filesystem Creation" illustrates an instance diagram showing the instances that shall exist for supporting 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 8.5.1.1.1 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.

## 8.2 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.

### 8.2.1 OperationalStatus for FileSystemConfigurationService

### 8.2.2 OperationalStatus for LocalFileSystem

**Table 57: 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 57: 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 happen 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.

**Table 57: LocalFileSystem OperationalStatus**

Primary OperationalStatus	Secondary OperationalStatus	Description
15 "Dormant"		The Filesystem is offline; and the reason for not being accessible is unknown.

### 8.3 Cascading Considerations

Under Consideration for a future standard.

### 8.4 Supported Subprofiles and Packages

**Table 58: Supported Profiles for Filesystem Manipulation**

Registered Profile Names	Mandatory	Version
Job Control	No	1.2.0
Filesystem	Yes	1.2.0
Indication	Yes	1.2.0
Volume Composition	No	1.2.0

## 8.5 Methods of the Profile

### 8.5.1 Extrinsic Methods of the Profile

**Table 59: 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) LocallyAccessibleFileSystemSetting(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) LocallyAccessibleFileSystemSetting(s) ElementSettingData(s) HostedDependency Dependency	N/A
FileSystemConfigurationService.ModifyFileSystem	(IF REQUESTED) LogicalDisk(s) StorageSetting(s) LocalAccessAvailable LocallyAccessibleFileSystemSetting ElementSettingData(s) HostedDependency	(if Local Access is modified) LocalAccessAvailable LocallyAccessibleFileSystemSetting ElementSettingData(s) HostedDependency	FileSystemSetting (if changed) ResidesOnExtent (if added)
FileSystemCapabilities.CreateGoalSettings	N/A	N/A	N/A
LocallyAccessibleFileSystemCapabilities.CreateGoalSettings	N/A	N/A	N/A
FileSystemCapabilities.GetRequiredStorageSize	N/A	N/A	N/A

#### 8.5.1.1 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.

#### 8.5.1.1.1 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

#### 8.5.1.1.2 Signature and Parameters of FileSystemCapabilities.CreateGoalSettings

**Table 60: Parameters for Extrinsic Method FileSystemCapabilities.CreateGoalSettings**

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	<b>EmbeddedInstance ("SNIA_FileSystemSetting")</b>  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	<b>EmbeddedInstance("SNIA_FileSystemSetting")</b>  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".
<b>Normal Return</b>			
Status		uint32	<b>ValueMap{ }, Values{ }</b>  "Success", "Failed", "Timeout", "Alternative Proposed"
<b>Error Returns</b>			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value

**Table 60: Parameters for Extrinsic Method FileSystemCapabilities.CreateGoalSettings**

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.
-------------------------------	-----------------	-----------	--

**8.5.1.2 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.

**8.5.1.2.1 Signature and Parameters of GetRequiredStorageSize****Table 61: Parameters for Extrinsic Method FileSystemCapabilities.GetRequiredStorageSize**

Parameter Name	Qualifier	Type	Description & Notes
FileSystemGoal	INOUT, EI	string	<b>EmbeddedInstance("SNIA_FileSystemSetting")</b> 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	<b>EmbeddedInstance("CIM_StorageSetting")</b> 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".

**Table 61: Parameters for Extrinsic Method FileSystemCapabilities.GetRequiredStorageSize (Con-**

Parameter Name	Qualifier	Type	Description & Notes
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.
<b>Normal Return</b>			
Status		uint32	<b>ValueMap{}</b> , <b>Values{}</b>  "Success", "Failed", "Timeout"
<b>Error Returns</b>			
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.

**8.5.1.3 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.

#### 8.5.1.3.1 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.

## 8.5.1.3.2 Signature and Parameters of CreateGoalSettings

**Table 62: Parameters for Extrinsic Method  
LocallyAccessibleFileSystemCapabilities.CreateGoalSettings**

Parameter Name	Qualifier	Type	Description & Notes
TemplateGoalSettings[]	IN	string	<b>EmbeddedInstance("SNIA_LocallyAccessibleFileSystemSetting")</b> 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.
SupportedGoalSettings[]	INOUT	string	<b>EmbeddedInstance("SNIA_LocallyAccessibleFileSystemSetting")</b> 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".
<b>Normal Return</b>			
Status		uint32	<b>ValueMap{ }, Values{ }</b> "Success", "Failed", "Timeout", "Alternative Proposed"
<b>Error Returns</b>			
Invalid Property Value	OUT, Indication	CIM_Error	A single named property of an instance parameter (either reference or embedded) has an invalid value

**Table 62: Parameters for Extrinsic Method LocallyAccessibleFileSystemCapabilities.CreateGoal-Settings (Continued)**

Parameter Name	Qualifier	Type	Description & Notes
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.

**8.5.1.4 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 separately).

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.

**8.5.1.4.1 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 we expect 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).

#### 8.5.1.4.2 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 we can extend this method 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.

#### 8.5.1.4.3 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.



## 8.5.2 Signature and Parameters of CreateFileSystem.

Table 63: 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	<b><i>EmbeddedInstance ("CIM_FileSystemSetting")</i></b>  The FileSystemSetting requirements for the FileSystem. If NULL or the empty string, a default FileSystemSetting shall be specified by the FileSystemCapabilities 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	<b><i>EmbeddedInstance ("CIM_StorageSetting")</i></b>  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.

**Table 63: Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
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.
FileServer	IN, OUT, REF, NULL Allowed	ComputerSystem	<p>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.</p> <p>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.</p> <p>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.</p> <p>On output, this parameter contains a reference to the actual FileServer that has access to the created LocalFileSystem.</p>

**Table 63: Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
LocalAccessPoint	IN, OUT, REF, NULL Allowed	string	<p>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.</p> <p>The parameters for local access are specified by the LocalAccessSetting parameter.</p> <p>If FileSystemConfigurationCapabilities.LocalAccessibilitySupport specifies that local access points are required, then LocalAccessPoint shall not be NULL or an empty string.</p> <p>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.</p> <p>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.</p> <p>On output, this parameter contains the actual LocalAccessPoint used (including any name created by vendor-default).</p>

**Table 63: Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
LocalAccessSetting	IN, EI, OUT, NULL Allowed	string	<p><i>EmbeddedInstance</i> ( "CIM_LocallyAccessibleFileSystemSetting" )</p> <p>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.</p> <p>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.</p> <p>The LocalAccessSetting actually used is returned as the OUT EmbeddedInstance parameter.</p>

**Table 63: Parameters for Extrinsic Method FileSystemConfigurationService.CreateFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
DirectoryServer	IN, OUT, NULL Allowed	ComputerSystem	<p>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.</p> <p>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.</p> <p>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.</p> <p>On output, this parameter contains a reference to the actual DirectoryServer if one was established.</p>
<b>Normal Return</b>			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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.

### 8.5.2.1 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).

### 8.5.3 Signature and Parameters of ModifyFileSystem.

**Table 64: 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	<b><i>EmbeddedInstance ( "CIM_FileSystemSetting" )</i></b> 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	<b><i>EmbeddedInstance ( "CIM_StorageSetting" )</i></b> 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 64: Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
FileServer	IN, OUT, REF, NULL Allowed	REF Computer System	<p>A reference to a ComputerSystem element representing a file server.</p> <p>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:</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>

**Table 64: Parameters for Extrinsic Method FileSystemConfigurationService.ModifyFileSystem**

Parameter Name	Qualifier	Type	Description & Notes
LocalAccess Point	IN, OUT, REF, NULL Allowed	string	<p>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.</p> <p>The parameters for local access are specified by the LocalAccessSetting parameter.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>On output, this parameter contains the actual LocalAccessPoint used (including any name created by vendor-default).</p>
LocalAccess Setting	IN, EI, OUT, NULL Allowed	string	<p><b><i>EmbeddedInstance</i></b> <b><i>("SNIA_LocallyAccessibleFileSystemSetting")</i></b></p> <p>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.</p> <p>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.</p>

**Table 64: 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'.
<b>Normal Return</b>			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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.

**8.5.3.1 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.

#### 8.5.4 Signature and Parameters of DeleteFileSystem.

**Table 65: 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.
<b>Normal Return</b>			
Status		uint32	"Job Completed with No Error", "Failed", "Method Parameters Checked - Job Started"
<b>Error Returns</b>			
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.

#### 8.5.5 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

## 8.6 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.
- We use `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.

**Note:** We need to add at least one recipes that supports local access from a file server.

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.
```

#### 8.6.1 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.
```

## Filesystem Manipulation Subprofile

```
// 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++;
}

```

## Filesystem Manipulation Subprofile

```

}
$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.
//
$dldsetting = 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;
}

```

## 8.6.2 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,

```

## Filesystem Manipulation Subprofile

```

        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;

    //

```

## Filesystem Manipulation Subprofile

```
// 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
$scapabilities->[] = Associators($fsconfigurator,
    "CIM_ElementCapabilities",
    "CIM_FileSystemCapabilities",
    "ManagedElement",
    "Capabilities");

if ($scapabilities->[] == null || $scapabilities->[].length == 0) {
    // No Capabilities found -- error
    $job = NULL;
    return;
}

#j = 0;
while($scapability = $scapabilities->[#j]) {
    if ($scapability.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
//   );
//
}
```

### 8.6.3 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;
}
$fsnewgoal = $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;
    }

```

#### 8.6.4 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) {

```

## Filesystem Manipulation Subprofile

```
// 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;
}
```

**8.6.5 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;
}

```

**8.6.6 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

---

### 8.6.7 Filesystem Manipulation Supported Capabilities Patterns

Table 66, “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 66: 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

## 8.7 Registered Name and Version

Filesystem Manipulation version 1.2.0



## 8.8 CIM Elements

**Table 67: CIM Elements for Filesystem Manipulation**

Element Name	Requirement	Description
SNIA_FileSystemConfigurationService (8.8.1)	Mandatory	The Filesystem Configuration Service provides the methods to manipulate file systems.
CIM_HostedService (8.8.2)	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.
SNIA_FileSystemConfigurationCapabilities (8.8.3)	Mandatory	This element represents the management Capabilities of the Filesystem Configuration Service.
SNIA_ElementCapabilities (FS Configuration Capabilities) (8.8.4)	Mandatory	In this subprofile, associates the Filesystem Configuration Service to the Capabilities element that represents the capabilities that it supports.
SNIA_FileSystemCapabilities (8.8.5)	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 FileSystemConfigurationCapabilities.SupportedActualFileSystemTypes, there will be exactly one corresponding FileSystemCapabilities element with the matching ActualFileSystemType property.
SNIA_ElementCapabilities (At Least One) (8.8.6)	Mandatory	In this subprofile, associates the Filesystem Configuration Service to the FileSystemCapabilities elements that represent all the types of filesystems that it can configure.
SNIA_ElementCapabilities (At Least One) (8.8.7)	Mandatory	This entry represents the single default FileSystemCapabilities element for the Filesystem Configuration Service.

**Table 67: CIM Elements for Filesystem Manipulation**

Element Name	Requirement	Description
SNIA_FileSystemSetting (Pre-defined FS Settings) (8.8.8)	Optional	This element represents sample configuration settings usable for creating or modifying a LocalFileSystem. It represents "pre-defined" 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.
SNIA_SettingsDefineCapabilities (Pre-defined FS Settings) (8.8.9)	Optional	These Setting elements provide detailed information about the FileSystemSettings supported by the associated FileSystemCapabilities element.
SNIA_FileSystemSetting (Attached to FileSystem) (8.8.10)	Optional	<p>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.</p> <p>This profile does not specify how other instances of this class might be created.</p>
CIM_ElementSettingData (Attached to Filesystem) (8.8.11)	Optional	<p>Associates a FileSystemSetting element to a LocalFileSystem. One of these association elements is created by CreateFileSystem when the LocalFileSystem is first created.</p> <p>The profile does not specify how other instances of this association may be surfaced by the implementation.</p>
SNIA_LocalFileSystem (8.8.12)	Mandatory	Represents a LocalFileSystem hosted by and made available through a ComputerSystem (usually the top-level ComputerSystem of a Filesystem Profile).
CIM_HostedFileSystem (8.8.13)	Mandatory	Associates a LocalFileSystem to the ComputerSystem that hosts it.
CIM_Directory (Root Directory) (8.8.14)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>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.</p>

**Table 67: CIM Elements for Filesystem Manipulation**

Element Name	Requirement	Description
CIM_FileStorage (Root Directory) (8.8.15)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>Associates the root Directory to its parent LocalFileSystem.</p>
CIM_LogicalFile (Shared Files and Directories) (8.8.16)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>A LogicalFile (or Directory subclass) that is exported as a FileShare is also visible as a sub-element of the LocalFileSystem.</p> <p>Maybe this class should be defined only in the File Export subprofile.</p>
CIM_FileStorage (Shared Files and Directories) (8.8.17)	Conditional	<p>Conditional requirement: Required if parent profile is backward compatible to SMI Specification v1.1..</p> <p>Associates an exported Logical File or Directory to the LocalFileSystem that contains it.</p>
SNIA_LocallyAccessibleFileSystemSetting (8.8.18)	Conditional	<p>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.</p>
CIM_Dependency (Uses Directory Services From) (8.8.19)	Conditional	<p>Conditional requirement: Required if LocalFileSystem.DirectoryServiceUsage is either Required'or'Optional'.AssociatesaComputerSystemthatindicatesadirectoryservicethatsupportsthe dependentLocalFileSystem.'</p>
SNIA_SettingsDefineCapabilities (Pre-defined Local Access Settings) (8.8.20)	Conditional	<p>Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either LocalAccessRequired,Defaulted'or'LocalAccessRequired,NotDefaulted'.TheSettingelementsthat are associated to this Capabilities element are scoped to the FileServerComputerSystem that provides the operational context for local access.'</p>

**Table 67: CIM Elements for Filesystem Manipulation**

Element Name	Requirement	Description
CIM_ElementCapabilities (Local Access Configuration Capabilities) (8.8.21)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either LocalAccessRequired,Defaulted'or'LocalAccessRequired,NotDefaulted'. In this subprofile, associates the FileSystemConfigurationService to the Capabilities instance that represents the capabilities for Local Access that it supports.'
SNIA_LocallyAccessibleFileSystemCapabilities (8.8.22)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either LocalAccessRequired,Defaulted'or'LocalAccessRequired,NotDefaulted'. The element represents the Local Access configuration Capabilities of the FileSystemConfigurationService. This class provides a CreateGoalSettings method that will return a SNIA_LocallyAccessibleFileSystemSetting element as an Embedded Instance that may be used for making a filesystem locally accessible to a file server ComputerSystem (by the methods CreateFileSystem and ModifyFileSystem). Since the returned Embedded Instance setting element is an instance of a ScopedSetting class, it must be associated with a ComputerSystem via ScopedSettingData when it is instantiated.'
CIM_HostedDependency (Attached to File System) (8.8.23)	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.
CIM_HostedDependency (Pre-Defined) (8.8.24)	Optional	Associates a pre-defined SNIA_LocallyAccessibleFileSystemSetting to the file server ComputerSystem that provides the operational scope for its functionality.
CIM_HostedDependency (Pre-Defined) (8.8.25)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either LocalAccessRequired,Defaulted'or'LocalAccessRequired,NotDefaulted'. Associates a LocalAccessCapabilities to the FileServer that provides the operational scope for its functionality. All of the Settings associated to the referenced Capabilities element must be scoped by the same FileServerComputerSystem. This scoping allows the CreateGoalSetting method of the Capabilities element to know which FileServer provides the scope for any Goal element that it creates.'

**Table 67: CIM Elements for Filesystem Manipulation**

Element Name	Requirement	Description
CIM_ElementSettingData (Local Access Required) (8.8.26)	Conditional	Conditional requirement: Required if FileSystemConfigurationCapabilities.LocalAccessibilitySupport is either LocalAccessRequired,Defaulted'or'LocalAccessRequired,NotDefaulted'.AssociatesaLocalFileSystemandtheLocallyAccessibleFileSystemSettingelements.'
SNIA_LocalAccessAvailable (8.8.27)	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.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA SNIA_LocalFileSystem	Optional	Experimental CQL - Creation of a LocalFileSystem element.
SELECT OBJECTPATH(SourceInstance)AS FSPath, SourceInstance.Name FROM CIM_InstDeletion WHERE SourceInstance ISA SNIA_LocalFileSystem	Optional	Experimental CQL - Deletion of a LocalFileSystem element.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_LocalFileSystem	Optional	Experimental CQL - Modification of a LocalFileSystem element.

**8.8.1 SNIA\_FileSystemConfigurationService**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 68 describes class SNIA\_FileSystemConfigurationService.

**Table 68: SMI Referenced Properties/Methods for SNIA\_FileSystemConfigurationService**

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A user-friendly name for this Service.
SystemCreationClass		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.

**Table 68: SMI Referenced Properties/Methods for SNIA\_FileSystemConfigurationService**

Properties	Flags	Requirement	Description & Notes
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.

**8.8.2 CIM\_HostedService**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 69 describes class CIM\_HostedService.

**Table 69: SMI Referenced Properties/Methods for CIM\_HostedService**

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

**8.8.3 SNIA\_FileSystemConfigurationCapabilities**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 70 describes class SNIA\_FileSystemConfigurationCapabilities.

**Table 70: 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. Note: A supported method shall be listed in this property or in the SupportedAsynchronousMethods property or both.
SupportedAsynchron ousMethods	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 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.
LocalAccessibilitySu pport		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 70: SMI Referenced Properties/Methods for SNIA\_FileSystemConfigurationCapabilities**

Properties	Flags	Requirement	Description & Notes
BlockStorageCreationSupport		Optional	<p>BlockStorageCreationSupport is an ordered array of enumerated values that place a number of restrictions on the use of parameters for CreateFileSystem and ModifyFileSystem.</p> <ol style="list-style-type: none"> <li>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 file system 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.</li> <li>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.</li> <li>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.</li> <li>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).</li> </ol>



**Table 70: SMI Referenced Properties/Methods for SNIA\_FileSystemConfigurationCapabilities**

Properties	Flags	Requirement	Description & Notes
DirectoryServerParameterSupported		Optional	<p>This enumeration indicates support for the DirectoryServer parameter to the extrinsic method FileSystemConfigurationService.CreateFileSystem(). The options are:</p> <p>'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.</p> <p>'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.</p> <p>'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.</p> <p>'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().</p>

**8.8.4 SNIA\_ElementCapabilities (FS Configuration Capabilities)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 71 describes class SNIA\_ElementCapabilities (FS Configuration Capabilities).

**Table 71: SMI Referenced Properties/Methods for SNIA\_ElementCapabilities (FS Configuration Capabilities)**

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The Filesystem Configuration Service
Capabilities		Mandatory	The Filesystem Configuration Capabilities element

### 8.8.5 SNIA\_FileSystemCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 72 describes class SNIA\_FileSystemCapabilities.

**Table 72: 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.
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.
GetRequiredStorageSize()		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.

**8.8.6 SNIA\_ElementCapabilities (At Least One)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 73 describes class SNIA\_ElementCapabilities (At Least One).

**Table 73: SMI Referenced Properties/Methods for SNIA\_ElementCapabilities (At Least One)**

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	
ManagedElement		Mandatory	

**8.8.7 SNIA\_ElementCapabilities (At Least One)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 74 describes class SNIA\_ElementCapabilities (At Least One).

**Table 74: SMI Referenced Properties/Methods for SNIA\_ElementCapabilities (At Least One)**

Properties	Flags	Requirement	Description & Notes
Characteristics		Optional	
Capabilities		Mandatory	
ManagedElement		Mandatory	

**8.8.8 SNIA\_FileSystemSetting (Pre-defined FS Settings)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 75 describes class SNIA\_FileSystemSetting (Pre-defined FS Settings).

**Table 75: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting (Pre-defined 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.
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.

**Table 75: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting (Pre-defined FS Settings)**

Properties	Flags	Requirement	Description & Notes
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.
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.

**8.8.9 SNIA\_SettingsDefineCapabilities (Pre-defined FS Settings)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 76 describes class SNIA\_SettingsDefineCapabilities (Pre-defined FS Settings).

**Table 76: SMI Referenced Properties/Methods for SNIA\_SettingsDefineCapabilities (Pre-defined 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 76: SMI Referenced Properties/Methods for SNIA\_SettingsDefineCapabilities (Pre-defined 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.

**8.8.10 SNIA\_FileSystemSetting (Attached to FileSystem)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: Optional

Table 77 describes class SNIA\_FileSystemSetting (Attached to FileSystem).

**Table 77: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting (Attached to FileSystem)**

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.
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 77: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting (Attached to FileSystem)**

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 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.
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.

**Table 77: SMI Referenced Properties/Methods for SNIA\_FileSystemSetting (Attached to FileSystem)**

Properties	Flags	Requirement	Description & Notes
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.

**8.8.11 CIM\_ElementSettingData (Attached to Filesystem)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: Optional

Table 78 describes class CIM\_ElementSettingData (Attached to Filesystem).

**Table 78: 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.

**8.8.12 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

Class Mandatory: Mandatory



Table 79 describes class SNIA\_LocalFileSystem.

**Table 79: 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	<p>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:</p> <p>'Not Used' indicates that the filesystem will not support security principal information and so will not require support from a directory service.</p> <p>'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.</p> <p>'Required' indicates that the filesystem supports security principal information and will require support from a directory service. The Dependency association described above must exist.</p>
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 79: 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.

**8.8.13 CIM\_HostedFileSystem**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: Mandatory

Table 80 describes class CIM\_HostedFileSystem.

**Table 80: SMI Referenced Properties/Methods for CIM\_HostedFileSystem**

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The ComputerSystem that hosts a LocalFileSystem. The Dedicated property must be one of 24 (NAS Head), 25 (SC NAS), 16 (File Server).
PartComponent		Mandatory	The hosted filesystem.

**8.8.14 CIM\_Directory (Root Directory)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem

Class Mandatory: BC1.1

**8.8.15 CIM\_FileStorage (Root Directory)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem

Class Mandatory: BC1.1

Table 81 describes class CIM\_FileStorage (Root Directory).

**Table 81: SMI Referenced Properties/Methods for CIM\_FileStorage (Root Directory)**

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The LocalFileSystem that contains the associated root Directory.

**Table 81: SMI Referenced Properties/Methods for CIM\_FileStorage (Root Directory)**

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	The Root Directory of the LocalFileSystem.

**8.8.16 CIM\_LogicalFile (Shared Files and Directories)**

Created By: Extrinsic: CreateExportedShare or ModifyExportedShare

Modified By: Extrinsic: CreateExportedShare or ModifyExportedShare

Deleted By: Extrinsic: DeleteExportedShare or ModifyExportedShare

Class Mandatory: BC1.1

Table 82 describes class CIM\_LogicalFile (Shared Files and Directories).

**Table 82: 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.

**8.8.17 CIM\_FileStorage (Shared Files and Directories)**

Created By: Extrinsic: CreateExportedShare or ModifyExportedShare

Modified By: Static

Deleted By: Extrinsic: DeleteExportedShare or ModifyExportedShare

Class Mandatory: BC1.1

Table 83 describes class CIM\_FileStorage (Shared Files and Directories).

**Table 83: 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.

#### 8.8.18 SNIA\_LocallyAccessibleFileSystemSetting

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem or ModifyFileSystem

Class Mandatory: LocalAccessDefinitionRequired

Table 84 describes class SNIA\_LocallyAccessibleFileSystemSetting.

**Table 84: 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.

**Table 84: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
InitialEnabledState		Optional	<p>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:</p> <p>'Enabled' (2) indicates that the element will execute commands, will process any queued commands, and will queue new requests.</p> <p>'Disabled' (3) indicates that the element will not execute commands and will drop any new requests.</p> <p>'In Test' (7) indicates that the element will be in a test state.</p> <p>'Deferred' (8) indicates that the element will not process any commands but will queue new requests.</p> <p>'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.</p>
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.
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.

**Table 84: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
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 84: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
AttributeCaching		Optional	<p>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.</p> <p>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".</p>
AttributeCachingObjects		Optional	<p>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).</p>
AttributeCachingTimeMin		Optional	<p>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.</p>
AttributeCachingTimeMax		Optional	<p>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.</p>



**Table 84: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemServiceSetting**

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.
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.

**Table 84: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemSetting**

Properties	Flags	Requirement	Description & Notes
RootAccessPref		Optional	<p>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.</p> <p>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.</p>

**8.8.19 CIM\_Dependency (Uses Directory Services From)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: DirectoryServiceUsage

Table 85 describes class CIM\_Dependency (Uses Directory Services From).

**Table 85: 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.

**8.8.20 SNIA\_SettingsDefineCapabilities (Pre-defined Local Access Settings)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LocalAccessSupported

Table 86 describes class SNIA\_SettingsDefineCapabilities (Pre-defined Local Access Settings).

**Table 86: SMI Referenced Properties/Methods for SNIA\_SettingsDefineCapabilities (Pre-defined 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.
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.

#### 8.8.21 CIM\_ElementCapabilities (Local Access Configuration Capabilities)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LocalAccessSupported

Table 87 describes class CIM\_ElementCapabilities (Local Access Configuration Capabilities).

**Table 87: 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

#### 8.8.22 SNIA\_LocallyAccessibleFileSystemCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LocalAccessSupported

Table 88 describes class SNIA\_LocallyAccessibleFileSystemCapabilities.

**Table 88: 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 88: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemCapabilities**

Properties	Flags	Requirement	Description & Notes
SupportedProperties		Mandatory	<p>An array of property names of the LocallyAccessibleFileSystemSetting that this SNIA_LocallyAccessibleFileSystemCapabilities element supports.</p> <p>2 'FailurePolicy'</p> <p>3 'RetriesMax'</p> <p>4 'InitialEnabledState'</p> <p>5 'RequestRetryPolicy'</p> <p>6 'TransmissionRetriesMax'</p> <p>7 'RetransmissionTimeout'</p> <p>8 'CachingOptions'</p> <p>9 'ReadBufferSize'</p> <p>10 'WriteBufferSize'</p> <p>11 'AttributeCaching'</p> <p>12 'ReadWritePolicy'</p> <p>13 'LockPolicy'</p> <p>14 'EnableOnSystemStart'</p> <p>15 'ReadWritePref'</p> <p>16 'ExecutePref'</p> <p>17 'RootAccessPref'</p>

**Table 88: SMI Referenced Properties/Methods for SNIA\_LocallyAccessibleFileSystemCapabilities**

Properties	Flags	Requirement	Description & Notes
SupportedObjectsForAttributeCaching		Optional	<p>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.</p> <p>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).</p> <p>0 'Unknown'</p> <p>1 'None'</p> <p>2 'All'</p> <p>3 'Rest'</p> <p>4 'File'</p> <p>5 'Directory'</p>

**8.8.23 CIM\_HostedDependency (Attached to File System)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: LocalAccessDefinitionRequired

Table 89 describes class CIM\_HostedDependency (Attached to File System).

**Table 89: SMI Referenced Properties/Methods for CIM\_HostedDependency (Attached to File System)**

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The Scoping File Server ComputerSystem.

**Table 89: SMI Referenced Properties/Methods for CIM\_HostedDependency (Attached to File System)**

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The Local Access Setting that is scoped by the file server ComputerSystem.

**8.8.24 CIM\_HostedDependency (Pre-Defined)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 90 describes class CIM\_HostedDependency (Pre-Defined).

**Table 90: SMI Referenced Properties/Methods for CIM\_HostedDependency (Pre-Defined)**

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.

**8.8.25 CIM\_HostedDependency (Pre-Defined)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LocalAccessSupported

Table 91 describes class CIM\_HostedDependency (Pre-Defined).

**Table 91: SMI Referenced Properties/Methods for CIM\_HostedDependency (Pre-Defined)**

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.

**8.8.26 CIM\_ElementSettingData (Local Access Required)**

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: LocalAccessSupported

Table 92 describes class CIM\_ElementSettingData (Local Access Required).

**Table 92: 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.

#### 8.8.27 SNIA\_LocalAccessAvailable

Created By: Extrinsic: CreateFileSystem

Modified By: Extrinsic: ModifyFileSystem

Deleted By: Extrinsic: DeleteFileSystem

Class Mandatory: LocalAccessDefinitionRequired

Table 93 describes class SNIA\_LocalAccessAvailable.

**Table 93: 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.



## **EXPERIMENTAL**

---



---

---

## EXPERIMENTAL

### Clause 9: Filesystem Quotas Profile

- Profile Name: Filesystem
- Version: 1.2.0
- Organization: SNIA
- CIM schema version: 2.13
- Central Class: LocalFileSystem
- Scoping Class: ComputerSystem

#### 9.1 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.

##### 9.1.1 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.

### 9.1.2 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.

### 9.1.3 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.

### 9.1.4 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 9.1.2 and 9.1.3.
- 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.

### 9.1.5 Quota types

Quotas are usually limits on disk space. Some systems, however, also support limits on the number of files and/or directories.

### 9.1.6 Class design considerations

#### 9.1.6.1 New Classes

This profile uses several new classes—FSDomainIdentity, FSQuotaCapabilities, FSQuotaReportRecord, FSQuotaConfigEntry, FSQuotaManagementService, and FSQuotaIndication

##### 9.1.6.1.1 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.

#### 9.1.6.1.2 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.

#### 9.1.6.1.3 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.

#### 9.1.6.1.4 FSQuotaConfigEntry

An FSQuotaConfigEntry instance represents a single quota entry supported by the system. For example, one FSQuotaConfigEntry 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.

#### 9.1.6.1.5 FSQuotaManagementService

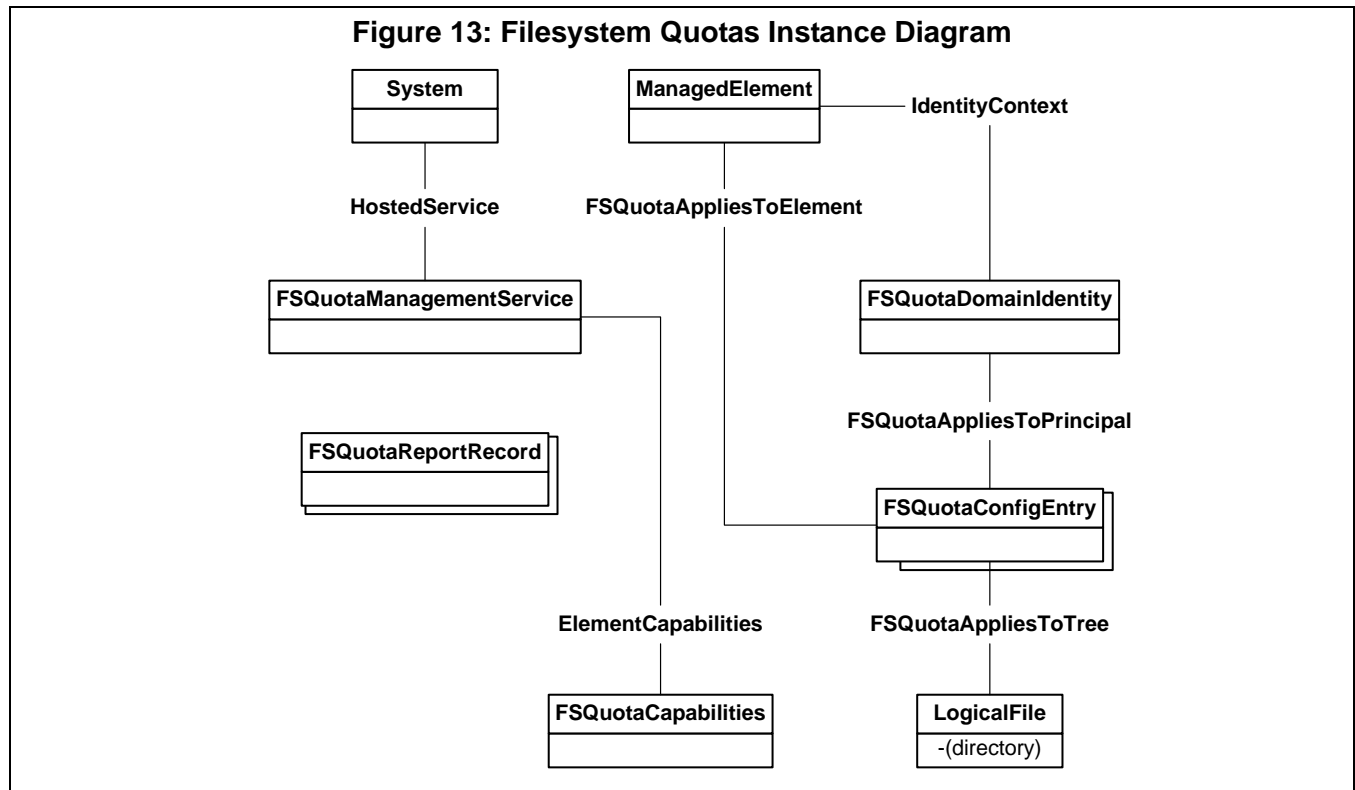
The FSQuotaManagementService provides the interface to the underlying system for most operations which are overtly related to quotas. There shall be at most one instance of a FSQuotaManagementService for each underlying ComputerSystem.

#### 9.1.6.1.6 FSQuotaIndication

The FSQuotaIndication class provides information about threshold crossing events, meaning that a quota has just been exceeded.

### 9.1.7 Instance Diagram

Figure 13 shows the Filesystem Quotas instance diagram.



## 9.2 Health and Fault Management Considerations

None currently applicable.

## 9.3 Supported Profiles, Subprofiles, and Packages

The Filesystem and Indications Profiles are required by this profile.

**Table 94: Supported Profiles for FileSystem Quotas**

Registered Profile Names	Mandatory	Version
Filesystem	Yes	1.2.0
Indication	Yes	1.2.0
Job	No	1.2.0

## 9.4 Methods of the Profile

All profile methods are contained in the **FSQuotaManagementService**.

### 9.4.1 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.

#### 9.4.2 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.

#### 9.4.3 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.

#### 9.4.4 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 9.7) will govern what happens if an entry already exists with the same combination of PrincipalID, ManagedElement, TreeName, ResourceType, QuotaType, and Default.

#### 9.4.5 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.

#### 9.4.6 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").

#### 9.4.7 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".

### 9.5 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 9.5.1.

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

#### 9.5.1 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; ++i) {
        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

### 9.5.2 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

### 9.5.3 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****9.5.4 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

### 9.5.5 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

### 9.5.6 Modify a quota entry

```

//
// There are many ways to modify a quota entry. Here are
// a couple examples
//

```

## Filesystem Quotas Profile

```
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

---

### 9.5.7 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

### 9.5.8 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

## 9.6 Registered Name and Version

Filesystem Quotas version 1.2.0

## 9.7 CIM Elements

**Table 95: CIM Elements for FileSystem Quotas**

Element Name	Requirement	Description
SNIA_FSDomainIdentity (9.7.1)	Mandatory	A small class containing the unique ID of a user or group in a Unix or Windows domain
SNIA_FSQuotaCapabilities (9.7.2)	Mandatory	The supported targets, quota types, resource types and behaviors of the FSQuotaManagementService associated to this class instance.
SNIA_ReportRecord (9.7.3)	Mandatory	An abstract keyless class proposed as the root of a tree of report record classes
SNIA_FSQuotaReportRecord (9.7.4)	Mandatory	A class representing a single line in a quota report generated by a call to the QuotaReport() extrinsic of the FSQuotaManagementService
SNIA_FSQuotaIndication (9.7.5)	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
SNIA_FSQuotaManagementService (9.7.6)	Mandatory	Quota Management Service class.
SNIA_FSQuotaConfigEntry (9.7.7)	Mandatory	A single quota entry in the configuration database.
SNIA_FSQuotaAppliesToElement (9.7.8)	Mandatory	An association between a quota config entry and a managed element
SNIA_FSQuotaAppliesToPrincipal (9.7.9)	Mandatory	An association between a quota config entry and a Filesystem principal entity
SNIA_FSQuotaAppliesToTree (9.7.10)	Mandatory	An association between a quota config entry and a directory
SELECT * FROM SNIA_FSQuotaIndication WHERE WhichLimit = 2	Mandatory	Hard quota threshold crossed
SELECT * FROM SNIA_FSQuotaIndication WHERE WhichLimit = 3	Mandatory	Soft quota threshold crossed

### 9.7.1 SNIA\_FSDomainIdentity

Created By: CreateInstance\_or\_Static\_or\_External

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory



Table 96 describes class SNIA\_FSDomainIdentity.

**Table 96: 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"

### 9.7.2 SNIA\_FSQuotaCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 97 describes class SNIA\_FSQuotaCapabilities.

**Table 97: 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".
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

**Table 97: SMI Referenced Properties/Methods for SNIA\_FSQuotaCapabilities**

Properties	Flags	Requirement	Description & Notes
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.

**9.7.3 SNIA\_ReportRecord**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

**9.7.4 SNIA\_FSQuotaReportRecord**

Created By: Extrinsic

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 98 describes class SNIA\_FSQuotaReportRecord.

**Table 98: 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
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.

**Table 98: SMI Referenced Properties/Methods for SNIA\_FSQuotaReportRecord**

Properties	Flags	Requirement	Description & Notes
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

**9.7.5 SNIA\_FSQuotaIndication**

Created By: External

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 99 describes class SNIA\_FSQuotaIndication.

**Table 99: 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"

**Table 99: SMI Referenced Properties/Methods for SNIA\_FSQuotaIndication**

Properties	Flags	Requirement	Description & Notes
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.

#### 9.7.6 SNIA\_FSQuotaManagementService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 100 describes class SNIA\_FSQuotaManagementService.

**Table 100: SMI Referenced Properties/Methods for SNIA\_FSQuotaManagementService**

Properties	Flags	Requirement	Description & Notes
SystemCreationClass		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.

### 9.7.7 SNIA\_FSQuotaConfigEntry

Created By: Extrinsic\_or\_External

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 101 describes class SNIA\_FSQuotaConfigEntry.

**Table 101: 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.
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.

### 9.7.8 SNIA\_FSQuotaAppliesToElement

Created By: CreateInstance

Modified By: Extrinsic\_or\_External

Deleted By: DeleteInstance

Class Mandatory: Mandatory

Table 102 describes class SNIA\_FSQuotaAppliesToElement.

**Table 102: SMI Referenced Properties/Methods for SNIA\_FSQuotaAppliesToElement**

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The managed element
Dependent		Mandatory	The quota config entry

#### 9.7.9 SNIA\_FSQuotaAppliesToPrincipal

Created By: CreateInstance

Modified By: Extrinsic\_or\_External

Deleted By: DeleteInstance

Class Mandatory: Mandatory

Table 103 describes class SNIA\_FSQuotaAppliesToPrincipal.

**Table 103: SMI Referenced Properties/Methods for SNIA\_FSQuotaAppliesToPrincipal**

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The filesystem principal
Dependent		Mandatory	The quota config entry

#### 9.7.10 SNIA\_FSQuotaAppliesToTree

Created By: CreateInstance

Modified By: Extrinsic\_or\_External

Deleted By: DeleteInstance

Class Mandatory: Mandatory

Table 104 describes class SNIA\_FSQuotaAppliesToTree.

**Table 104: SMI Referenced Properties/Methods for SNIA\_FSQuotaAppliesToTree**

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The filesystem directory tree
Dependent		Mandatory	The quota config entry

## **EXPERIMENTAL**

---





---

---

**STABLE****Clause 10: NAS Head Profile****10.1 Description****10.1.1 Synopsis**

Profile Name: NAS Head

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: ComputerSystem

Scoping Class: ComputerSystem

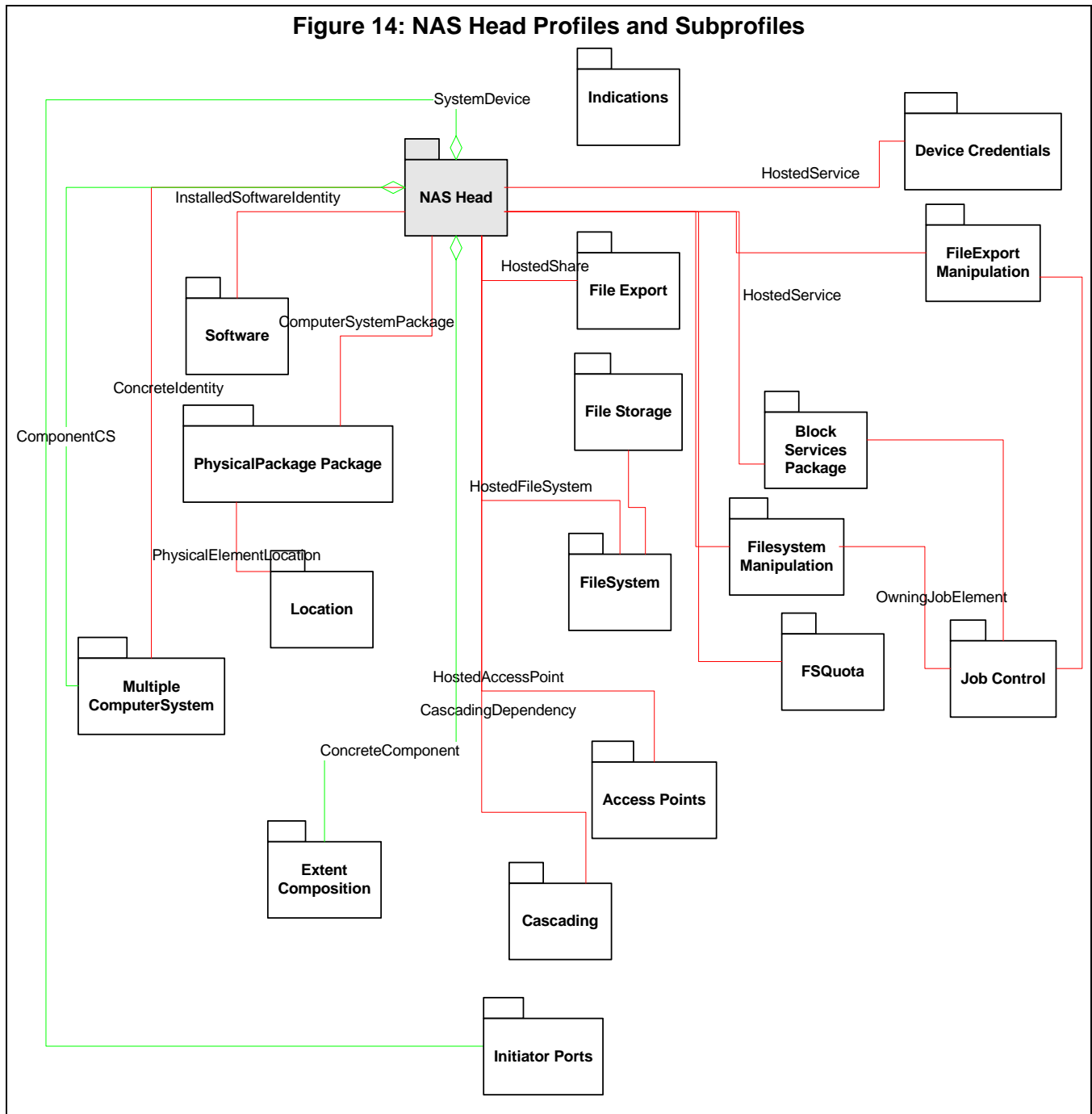
**10.1.2 Overview**

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. The details of how a Storage Array exports storage to the NAS Head is not covered in this profile but is covered by the Array profile.

The NAS Head Profile reuses a significant portion of Clause 23: Storage Virtualizer Profile in *Storage Management Technical Specification, Part 3 Block Devices*.

The NAS Head Profile and its subprofiles and packages are illustrated in Figure 14.

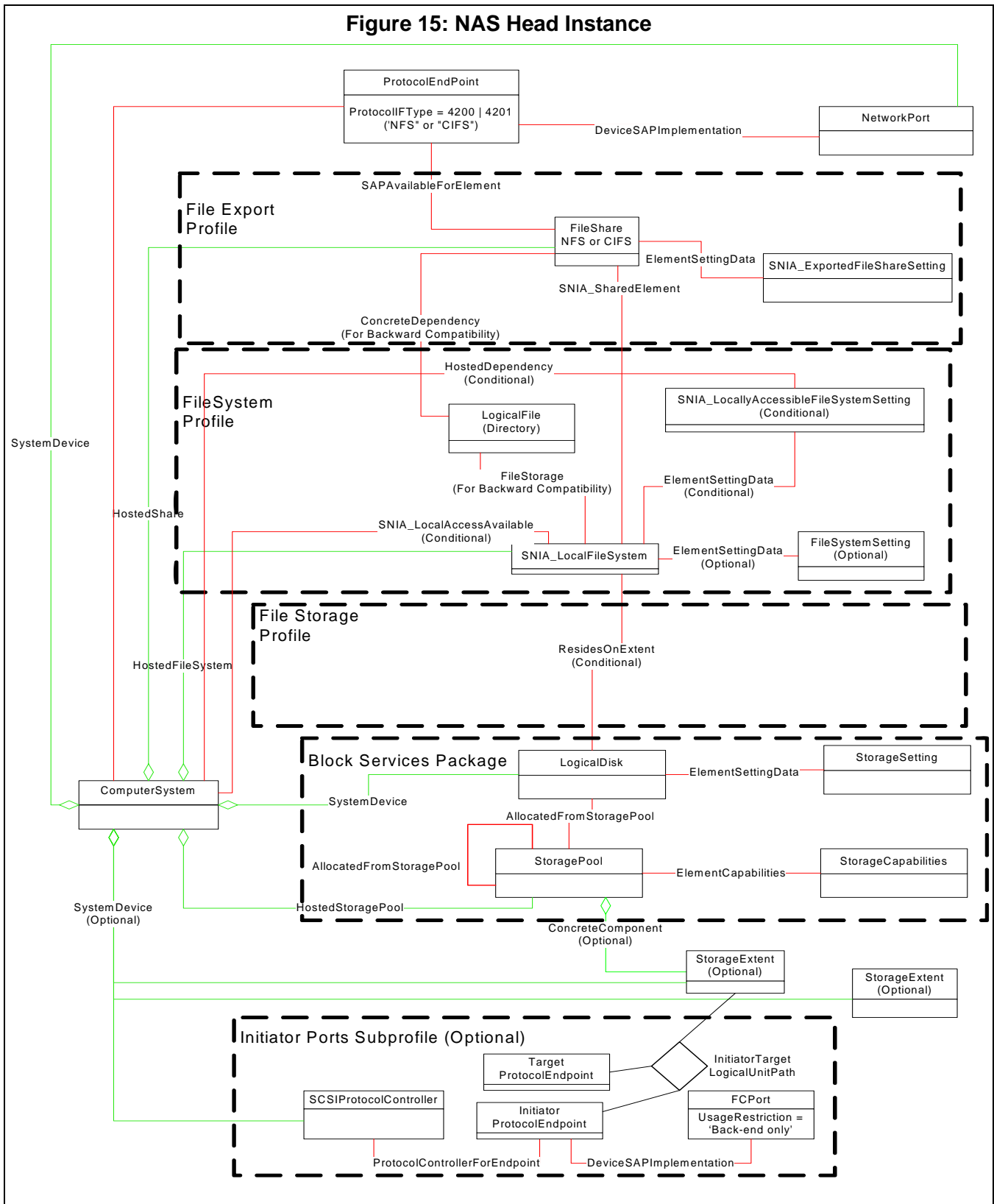


### 10.1.3 Implementation

#### 10.1.3.1 Summary Instance Diagram

Figure 15 illustrates the mandatory classes for the NAS Head Profile. This figure shows all the classes that are mandatory for the NAS Head Profile. Later diagrams will review specific sections of this diagram.

Figure 15: NAS Head Instance



The NAS Head Profile closely parallels the Storage Virtualizer 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 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.

As with 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. As a result, the NAS head shall model the Initiator ports and the StorageExtents that it acquires from the SAN. The NAS Head supports at least one of the Initiator Ports Subprofiles (the dashed box at the bottom of Figure 15) 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 15).

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.

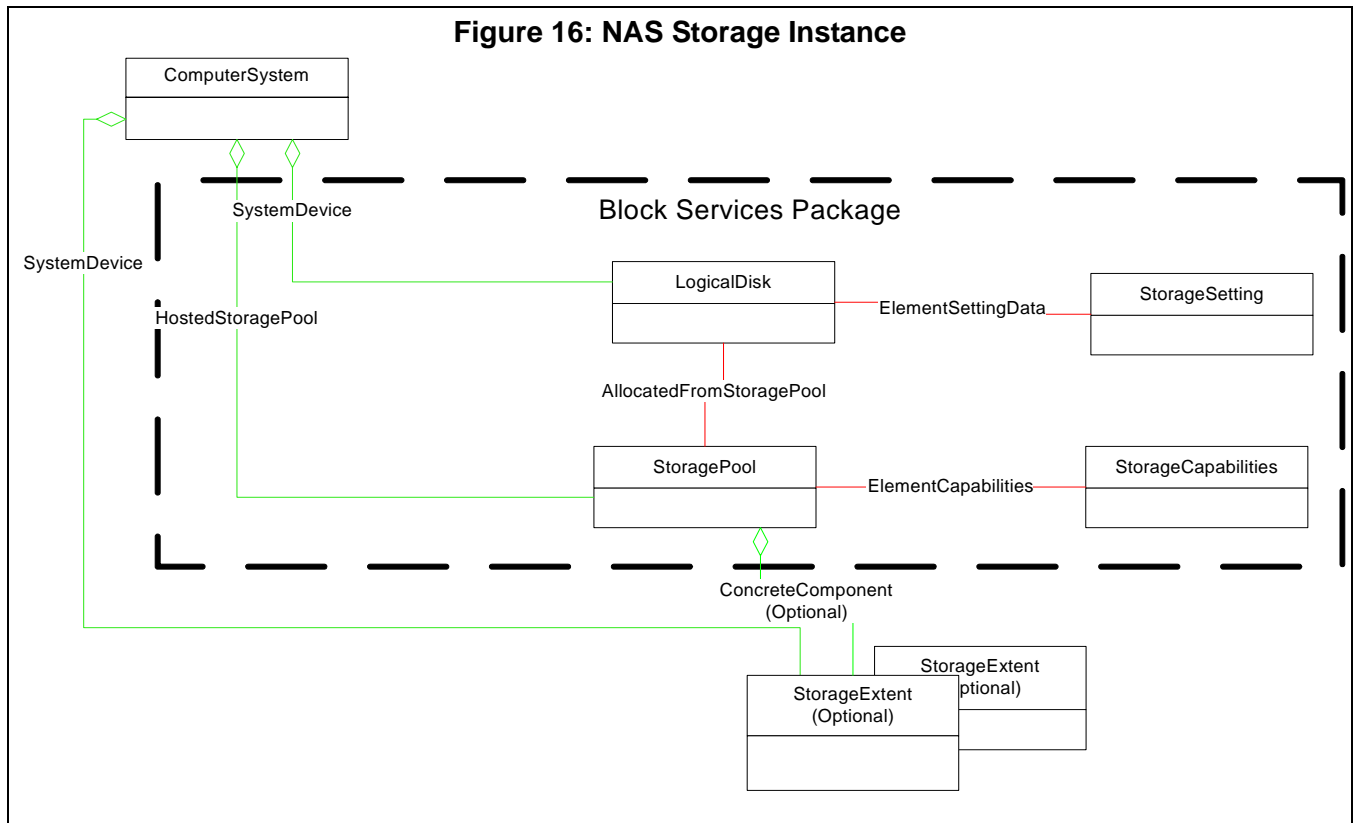
The ResidesOnExtent is conditional on the use by NAS Profiles (NAS Head and Self-contained NAS) in the File Storage Profile. In the NAS Head a LocalFileSystem shall map to a LogicalDisk.

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 15 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.

### 10.1.3.2 NAS Storage Model

Figure 16 illustrates the classes mandatory for modeling of storage for the NAS Head Profile.



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 16 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** shall also be modeled with **ConcreteComponent** associations to the **StoragePool** to which they belong and they will 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*. **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.

### 10.1.3.3 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, the Filesystem Profile shall be supported. See Clause 7: Filesystem Profile for details on this modeling.

#### **10.1.3.4 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, the Filesystem Profile shall be supported. See Clause 6: File Storage Profile for details on the file storage modeling.

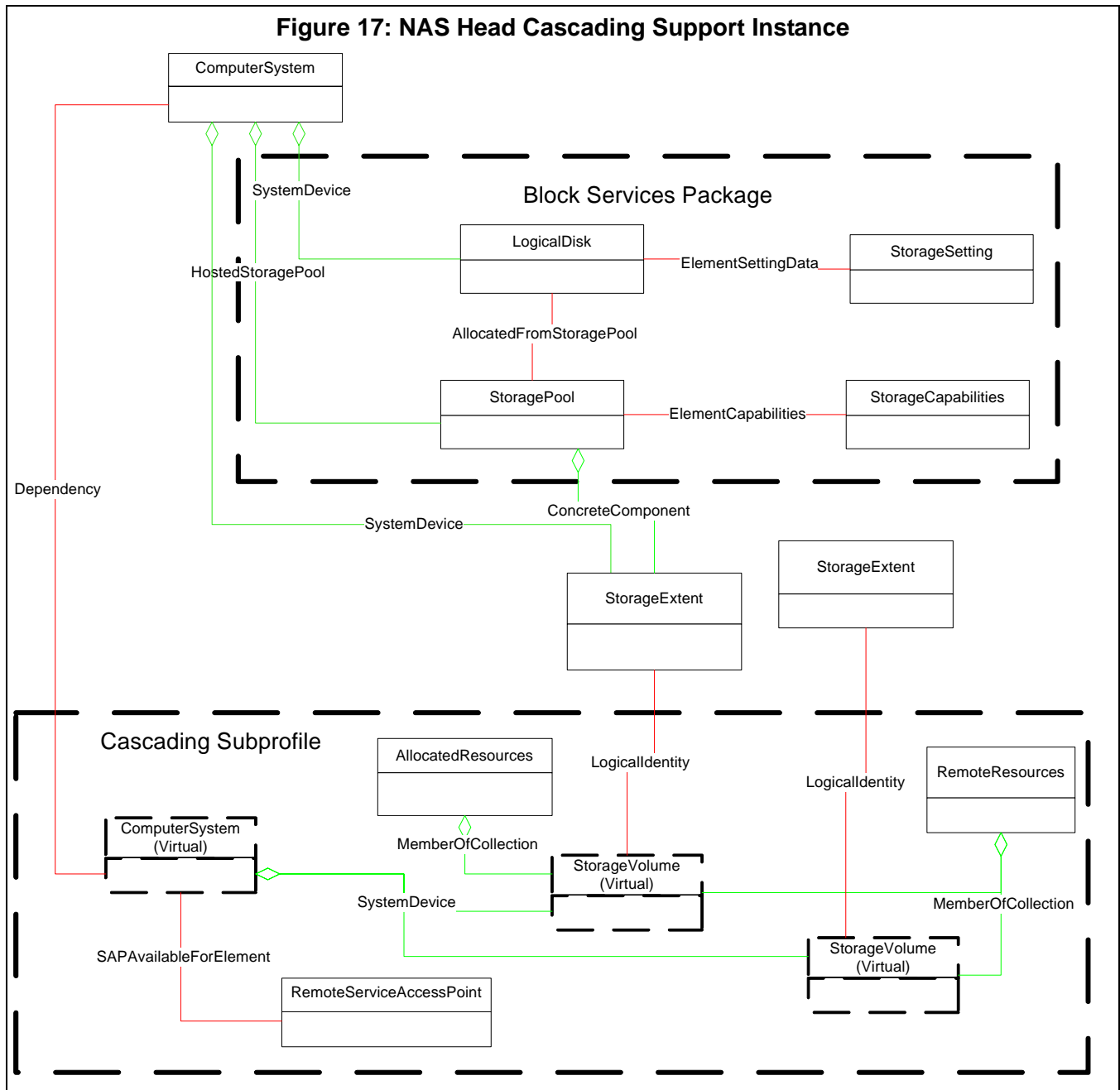
#### **10.1.3.5 NAS Head Use of File Export Profile (Mandatory)**

The NAS Head Profile uses the File Export Profile for modeling of its file export constructs. For the NAS Head, the File Export Profile shall be supported. See Clause 4: File Export Profile for details on this modeling.

## EXPERIMENTAL

### 10.1.3.6 NAS Head Support of Cascading

Figure 17 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.



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).

The RemoteServiceAccessPoint is the URL of the management interface that the NAS Head uses for managing the Array or Storage Virtualizer Profiles. This may or may not be an SMI-S Server URL.

## EXPERIMENTAL

---

### 10.2 Health and Fault Management Considerations

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

- Network Ports (See 10.2.1)
- Back-end Ports (See 17.3.1 Health and Fault Management Considerations in *Storage Management Technical Specification, Part 2 Common Profiles*)
- ComputerSystems (See 28.1.5 Computer System Operational Status in *Storage Management Technical Specification, Part 2 Common Profiles*)
- FileShares that are exported (See 4.2.1)
- LocalFileSystems (See 7.2.1)
- ProtocolEndpoints (See 10.2.2)

#### 10.2.1 OperationalStatus for Network Ports

**Table 105: NetworkPort OperationalStatus**

OperationalStatus	Description
OK	Port is online
Error	Port has a failure
Stopped	Port is disabled
InService	Port is in Self Test
Unknown	

#### 10.2.2 OperationalStatus for ProtocolEndpoints

**Table 106: ProtocolEndpoint OperationalStatus**

OperationalStatus	Description
OK	ProtocolEndpoint is online
Error	ProtocolEndpoint has a failure
Stopped	ProtocolEndpoint is disabled
Unknown	



---



---

## EXPERIMENTAL

### 10.3 Cascading Considerations

The NAS Head is a cascading Profile, but the Cascading Subprofile is Experimental in this release of SMI-S; see Clause 26: Cascading Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles*. 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.

#### 10.3.1 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 17 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 Concretelidentity) to a StorageVolume (from the Array or Storage Virtualizer Profile).

#### 10.3.2 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

#### 10.3.3 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

### 10.4 Supported Subprofiles and Packages

**Table 107: Supported Profiles for NAS Head**

Registered Profile Names	Mandatory	Version
Indication	Yes	1.2.0

**Table 107: Supported Profiles for NAS Head**

Registered Profile Names	Mandatory	Version
Filesystem	Yes	1.2.0
File Storage	Yes	1.2.0
File Export	Yes	1.2.0
Cascading	No	1.2.0
Access Points	No	1.2.0
Multiple Computer System	No	1.2.0
Software	No	1.2.0
Location	No	1.2.0
Extent Composition	No	1.2.0
Filesystem Manipulation	No	1.2.0
File Export Manipulation	No	1.2.0
Job Control	No	1.2.0
SPI Initiator Ports	No	1.2.0
FC Initiator Ports	No	1.2.0
Device Credentials	No	1.2.0
Physical Package	Yes	1.2.0
Block Services	Yes	1.2.0
Health	Yes	1.2.0

## 10.5 Methods of the Profile

### 10.5.1 Extrinsic Methods of the Profile

None.

### 10.5.2 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.

## **10.6 Client Considerations and Recipes**

Not defined in this version of the specification.

## **10.7 Registered Name and Version**

NAS Head version 1.2.0

## 10.8 CIM Elements

**Table 108: CIM Elements for NAS Head**

Element Name	Requirement	Description
CIM_BindsTo (CIFS or NFS) (10.8.1)	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.
CIM_BindsTo (TCP) (10.8.2)	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.
CIM_BindsToLANEndpoint (10.8.3)	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).
CIM_ComputerSystem (Top Level) (10.8.4)	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.
CIM_ComputerSystem (File Server) (10.8.5)	Mandatory	This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top level system. The File Server(s) must be manageable as a computer system and so could be exposed through other profiles and so there must be a way to correlate it with other management clients.
CIM_ConcreteComponent (10.8.6)	Optional	Represents the association between a Primordial StoragePool and the underlying StorageExtents that compose it.
CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort) (10.8.7)	Mandatory	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort) (10.8.8)	Conditional	Conditional requirement: This is required if a LANEndpoint exists. (LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.

**Table 108: CIM Elements for NAS Head**

Element Name	Requirement	Description
CIM_HostedAccessPoint (CIFS or NFS) (10.8.9)	Mandatory	(CIFS or NFS) Represents the association between a CIFS or NFS front end ProtocolEndpoint and the Computer System that hosts it.
CIM_HostedAccessPoint (TCP) (10.8.10)	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.
CIM_HostedAccessPoint (IP) (10.8.11)	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.
CIM_HostedAccessPoint (LAN) (10.8.12)	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.
CIM_IPProtocolEndpoint (10.8.13)	Optional	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
CIM_LANEndpoint (10.8.14)	Optional	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
CIM_LogicalDisk (LD for FS) (10.8.15)	Mandatory	Represents the single Storage Extent on which the NAS Head will build a LocalFileSystem.
CIM_NetworkPort (10.8.16)	Mandatory	Represents the front end logical port that supports access to a local area network.
CIM_ProtocolEndpoint (CIFS or NFS) (10.8.17)	Mandatory	(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
CIM_StorageExtent (Primordial) (10.8.18)	Optional	This StorageExtent represents the LUNs (StorageVolumes) imported from a storage device to the NAS Head.
CIM_SystemDevice (Storage Extents) (10.8.19)	Conditional	Conditional requirement: This is required if primordial StorageExtents exist. This association links all StorageExtents to the scoping system.
CIM_SystemDevice (Logical Disks) (10.8.20)	Mandatory	This association links all LogicalDisks to the scoping system.
CIM_SystemDevice (Network Ports) (10.8.21)	Mandatory	This association links all NetworkPorts to the scoping system. This is used to represent both front end and back end ports.

**Table 108: CIM Elements for NAS Head**

Element Name	Requirement	Description
CIM_TCPProtocolEndpoint (10.8.22)	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	Experimental 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	Experimental 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	Experimental 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	Optional	Experimental 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_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	Experimental CQL - Change of status of a LogicalDisk.  PreviousInstance is optional, but may be supplied by an implementation of the Profile.

**Table 108: 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.

**10.8.1 CIM\_BindsTo (CIFS or NFS)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: TCPProtocolEndpoint

Table 109 describes class CIM\_BindsTo (CIFS or NFS).

**Table 109: 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.

**10.8.2 CIM\_BindsTo (TCP)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: IPPProtocolEndpoint

Table 110 describes class CIM\_BindsTo (TCP).

**Table 110: SMI Referenced Properties/Methods for CIM\_BindsTo (TCP)**

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The TCPProtocolEndpoint that uses an IPPProtocolEndpoint for connectivity.
Antecedent		Mandatory	The IPPProtocolEndpoint that supports a TCPProtocolEndpoint.

### 10.8.3 CIM\_BindsToLANEndpoint

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: LANEndpoint

Table 111 describes class CIM\_BindsToLANEndpoint.

**Table 111: 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.

### 10.8.4 CIM\_ComputerSystem (Top Level)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 112 describes class CIM\_ComputerSystem (Top Level).

**Table 112: 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.
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



**Table 112: SMI Referenced Properties/Methods for CIM\_ComputerSystem (Top Level)**

Properties	Flags	Requirement	Description & Notes
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.

**10.8.5 CIM\_ComputerSystem (File Server)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 113 describes class CIM\_ComputerSystem (File Server).

**Table 113: SMI Referenced Properties/Methods for CIM\_ComputerSystem (File Server)**

Properties	Flags	Requirement	Description & Notes
Dedicated		Mandatory	This is a File Server (Dedicated=16). It could also support other capabilities, so we do not restrict the values that can be in the Dedicated array.
NameFormat		Mandatory	Format for Name property. This shall be "Other".
Name	C	Mandatory	Unique identifier for the NAS Head's File Servers, e.g., Vendor/Model/SerialNo+FS+Number. The Fileserver can have any number of IP addresses, so an IP address does not constitute a single unique id. Also, under various load-balancing or redundancy regimens, the IP address could move around, so it may not even be correlatable. For that reason, the vendor must support a format that will provide a unique ID for the file server.
OperationalStatus		Mandatory	Overall status of the 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.

**Table 113: SMI Referenced Properties/Methods for CIM\_ComputerSystem (File Server)**

Properties	Flags	Requirement	Description & Notes
RequestStateChange ( )		Optional	Not Specified in this version of the Profile.

**10.8.6 CIM\_ConcreteComponent**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Optional

Table 114 describes class CIM\_ConcreteComponent.

**Table 114: 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.

**10.8.7 CIM\_DeviceSAPImplementation (CIFS or NFS to NetworkPort)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 115 describes class CIM\_DeviceSAPImplementation (CIFS or NFS to NetworkPort).

**Table 115: 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.

**10.8.8 CIM\_DeviceSAPImplementation (LANEndpoint to NetworkPort)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LANEndpoint

Table 116 describes class CIM\_DeviceSAPImplementation (LANEndpoint to NetworkPort).

**Table 116: 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.

#### 10.8.9 CIM\_HostedAccessPoint (CIFS or NFS)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 117 describes class CIM\_HostedAccessPoint (CIFS or NFS).

**Table 117: 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).

#### 10.8.10 CIM\_HostedAccessPoint (TCP)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: TCPProtocolEndpoint

Table 118 describes class CIM\_HostedAccessPoint (TCP).

**Table 118: 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.

#### 10.8.11 CIM\_HostedAccessPoint (IP)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: IPProtocolEndpoint

Table 119 describes class CIM\_HostedAccessPoint (IP).

**Table 119: 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.

#### 10.8.12 CIM\_HostedAccessPoint (LAN)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LANEndpoint

Table 120 describes class CIM\_HostedAccessPoint (LAN).

**Table 120: 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.

**10.8.13 CIM\_IPProtocolEndpoint**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 121 describes class CIM\_IPProtocolEndpoint.

**Table 121: 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		Mandatory	An IP v4 address in the format "A.B.C.D".
IPv6Address		Mandatory	
SubnetMask		Mandatory	An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		Mandatory	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.

**Table 121: SMI Referenced Properties/Methods for CIM\_IPProtocolEndpoint**

Properties	Flags	Requirement	Description & Notes
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.

**10.8.14 CIM\_LANEndpoint**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 122 describes class CIM\_LANEndpoint.

**Table 122: 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.
NameFormat		Mandatory	The Format of the Name.
RequestedState		Optional	
OperationalStatus		Mandatory	The operational status of the PEP.
EnabledState		Optional	
OtherEnabledState		Optional	
TimeOfLastStateChange		Optional	

**Table 122: SMI Referenced Properties/Methods for CIM\_LANEndpoint**

Properties	Flags	Requirement	Description & Notes
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.

**10.8.15 CIM\_LogicalDisk (LD for FS)**

Created By: Extrinsic\_or\_External

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory



Table 123 describes class CIM\_LogicalDisk (LD for FS).

**Table 123: 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.
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.

**Table 123: SMI Referenced Properties/Methods for CIM\_LogicalDisk (LD for FS)**

Properties	Flags	Requirement	Description & Notes
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.
OtherNameNamespa ce	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.

**10.8.16 CIM\_NetworkPort**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 124 describes class CIM\_NetworkPort.

**Table 124: 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.
SystemCreationClas sName		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.
DeviceID		Mandatory	A unique ID for the device (in the context of the hosting System).
Speed		Optional	

**Table 124: SMI Referenced Properties/Methods for CIM\_NetworkPort**

Properties	Flags	Requirement	Description & Notes
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.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.

**Table 124: SMI Referenced Properties/Methods for CIM\_NetworkPort**

Properties	Flags	Requirement	Description & Notes
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.

**10.8.17 CIM\_ProtocolEndpoint (CIFS or NFS)**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 125 describes class CIM\_ProtocolEndpoint (CIFS or NFS).

**Table 125: SMI Referenced Properties/Methods for CIM\_ProtocolEndpoint (CIFS or NFS)**

Properties	Flags	Requirement	Description & Notes
SystemCreationClass Name		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 125: SMI Referenced Properties/Methods for CIM\_ProtocolEndpoint (CIFS or NFS)**

Properties	Flags	Requirement	Description & Notes
ProtocolType		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.

**10.8.18 CIM\_StorageExtent (Primordial)**

Created By: Static\_or\_External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 126 describes class CIM\_StorageExtent (Primordial).

**Table 126: SMI Referenced Properties/Methods for CIM\_StorageExtent (Primordial)**

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	

**Table 126: SMI Referenced Properties/Methods for CIM\_StorageExtent (Primordial)**

Properties	Flags	Requirement	Description & Notes
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	
Name		Mandatory	Identifier for a remote LUN on a storage array; possibly, the array ID plus LUN Node WWN. This LUN is imported from a remote storage device, so the NameFormat identifies the remote LUN by identifying the remote array and the unique LUN ID at that array. As an example below, we have specified a 16-character hex format for the Name taken from the Node WWN format.
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.

**Table 126: SMI Referenced Properties/Methods for CIM\_StorageExtent (Primordial)**

Properties	Flags	Requirement	Description & Notes
SequentialAccess	N	Optional	Not Specified in this version of the Profile.
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.

**10.8.19 CIM\_SystemDevice (Storage Extents)**

Created By: Extrinsic\_or\_External\_or\_Static

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: StorageExtents

Table 127 describes class CIM\_SystemDevice (Storage Extents).

**Table 127: 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.

**10.8.20 CIM\_SystemDevice (Logical Disks)**

Created By: Extrinsic\_or\_External\_or\_Static

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 128 describes class CIM\_SystemDevice (Logical Disks).

**Table 128: 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.

#### 10.8.21 CIM\_SystemDevice (Network Ports)

Created By: Extrinsic\_or\_External\_or\_Static

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 129 describes class CIM\_SystemDevice (Network Ports).

**Table 129: 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.

#### 10.8.22 CIM\_TCPProtocolEndpoint

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 130 describes class CIM\_TCPProtocolEndpoint.

**Table 130: 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.



**Table 130: SMI Referenced Properties/Methods for CIM\_TCPProtocolEndpoint**

Properties	Flags	Requirement	Description & Notes
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 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 11: Self-Contained NAS Profile****11.1 Description****11.1.1 Synopsis**

Profile Name: Self-Contained NAS

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: ComputerSystem

Scoping Class: ComputerSystem

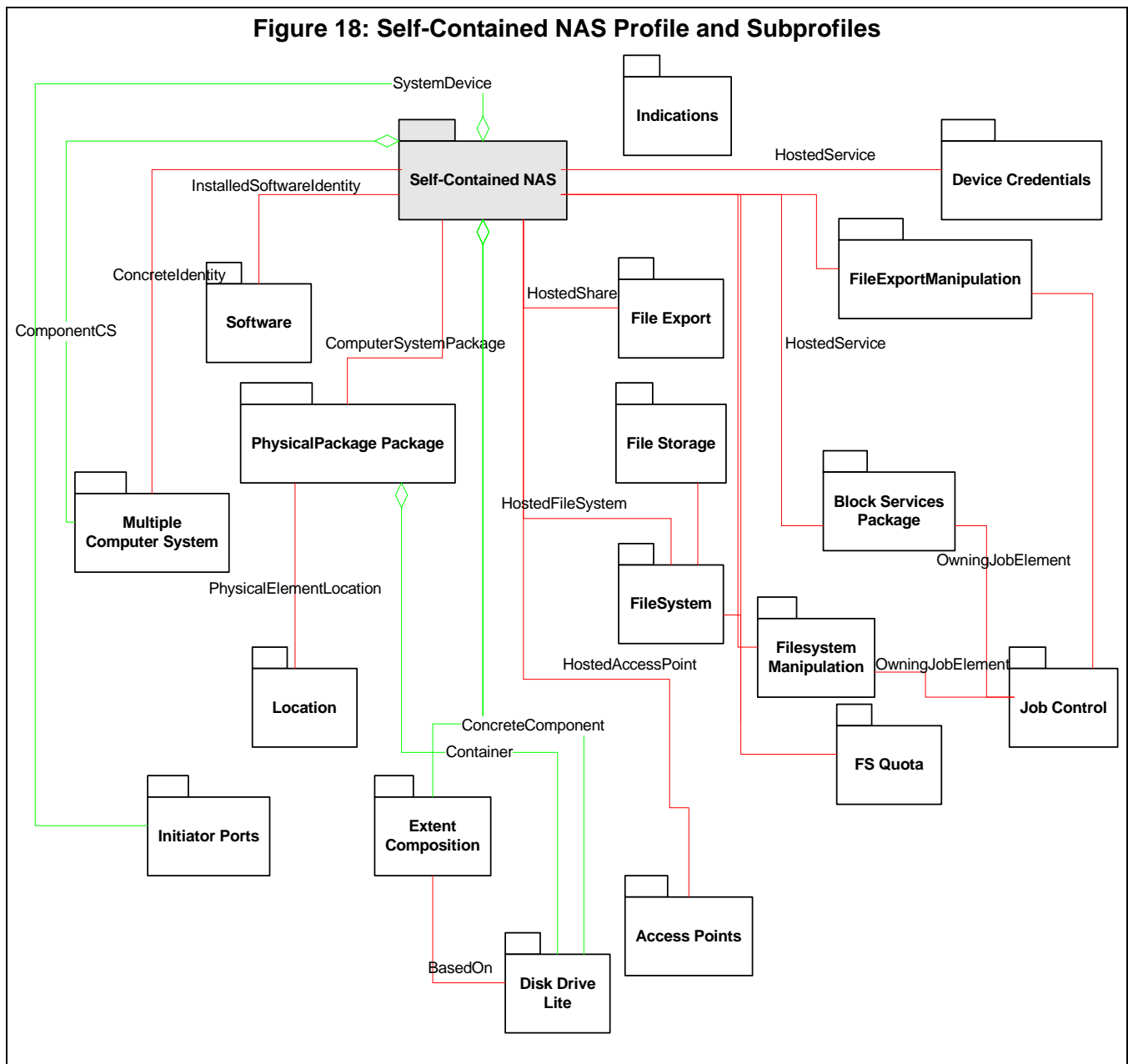
**11.1.2 Overview**

The Self-contained NAS (SC 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 SC 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*.

The Self-contained NAS profile reuses a significant portion of Clause 4: Array Profile in *Storage Management Technical Specification, Part 3 Block Devices*.

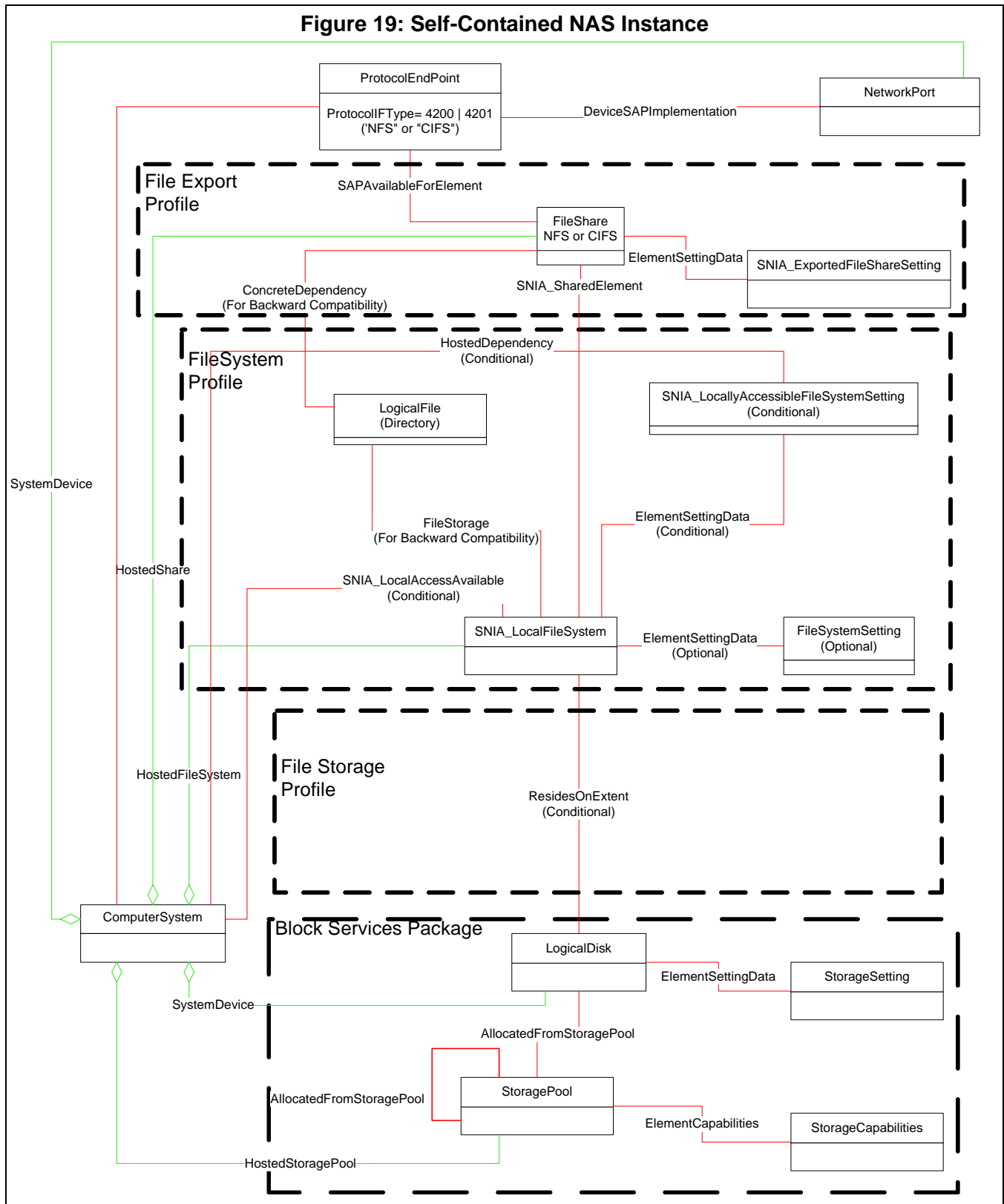
The Self-Contained NAS Profile and its subprofiles and packages are illustrated in Figure 18.



### 11.1.3 Implementation

### 11.1.3.1 Summary Instance Diagram

Figure 19 illustrates the mandatory classes of the Self-Contained NAS Profile. This figure shows all the classes that are mandatory for the Self-contained NAS Profile. Later diagrams will review specific sections of this diagram.



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.

Everything above the LogicalDisk is specific to NAS (and does not appear in the Array 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.

The ResidesOnExtent is conditional on the use by NAS Profiles (NAS Head and Self-contained NAS) in the File Storage Profile. In the Self-contained NAS a LocalFileSystem shall map to a LogicalDisk.

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 19 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

### 11.1.3.2 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 seemingly 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*.

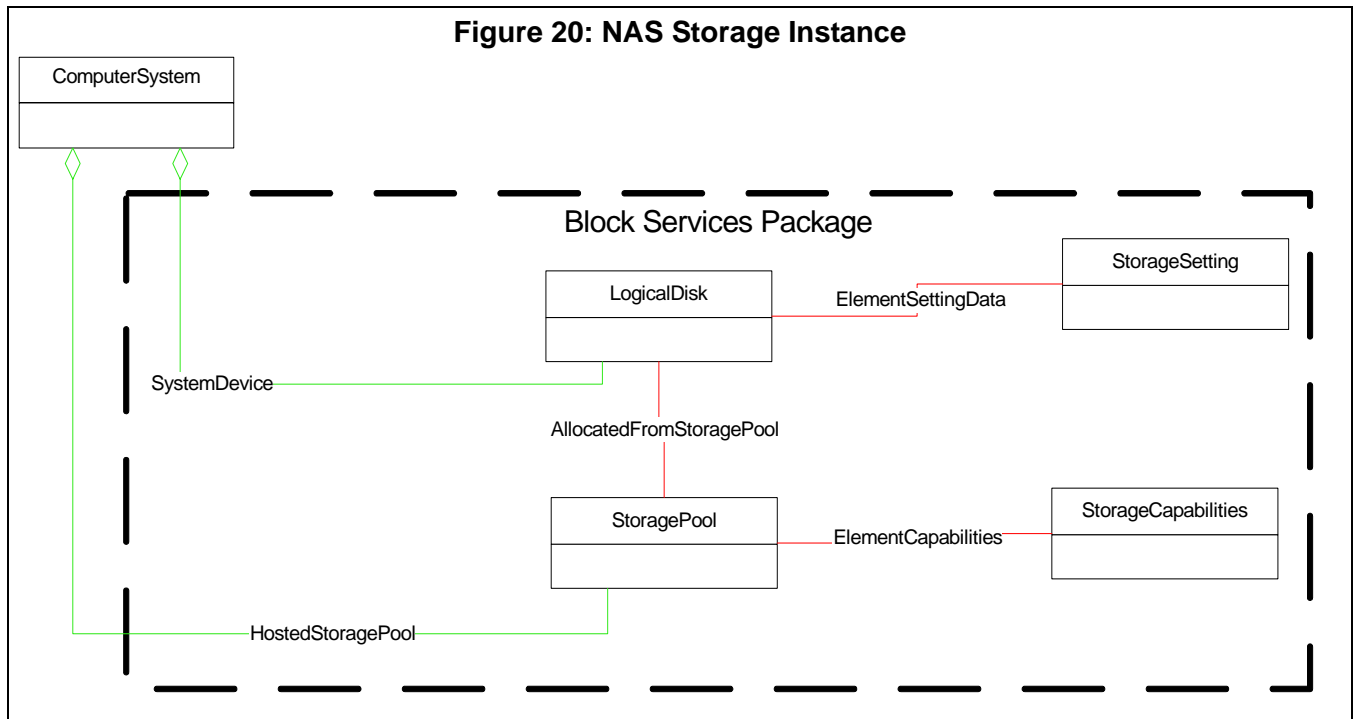
---

---

## EXPERIMENTAL

### 11.1.3.3 NAS Storage Model

Figure 20 illustrates the classes mandatory for modeling of storage for the Self-Contained NAS Profile.



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 20 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*).

For manipulation of Storage, see Clause 5: Block Services Package in the Block Devices Book. 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.

### 11.1.3.4 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, the Filesystem Profile shall be supported. See Clause 7: Filesystem Profile for details on this modeling.

#### 11.1.3.5 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, the Filesystem Profile shall be supported. See Clause 6: File Storage Profile for details on the file storage modeling.

#### 11.1.3.6 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, the File Export Profile shall be supported. See Clause 4: File Export Profile for details on this modeling.

### 11.2 Health and Fault Management Considerations

Self-Contained NAS supports state information (e.g., OperationalStatus) on the following elements of the model:

- Network Ports (See 11.2.1 OperationalStatus for Network Ports)
- Back-end Ports (See 17.3.1 Health and Fault Management Considerations of *Storage Management Technical Specification, Part 2 Common Profiles*)
- ComputerSystems (See 28.1.5 Computer System Operational Status of *Storage Management Technical Specification, Part 2 Common Profiles*)
- FileShares that are exported (See 4.2.1 OperationalStatus for FileShares)
- LocalFileSystems (See 7.2.1 OperationalStatus for Filesystems)
- ProtocolEndpoints (See 11.2.2 OperationalStatus for ProtocolEndpoints)
- DiskDrive (See 11.2 Health and Fault Management Considerations of *Storage Management Technical Specification, Part 3 Block Devices*)



**11.2.1 OperationalStatus for Network Ports****Table 131: NetworkPort OperationalStatus**

OperationalStatus	Description
OK	Port is online
Error	Port has a failure
Stopped	Port is disabled
InService	Port is in Self Test
Unknown	

**11.2.2 OperationalStatus for ProtocolEndpoints****Table 132: ProtocolEndpoint OperationalStatus**

OperationalStatus	Description
OK	ProtocolEndpoint is online
Error	ProtocolEndpoint has a failure
Stopped	ProtocolEndpoint is disabled
Unknown	

**11.3 Cascading Considerations**

Not Applicable.

**11.4 Supported Subprofiles and Packages****Table 133: Supported Profiles for Self-contained NAS System**

Registered Profile Names	Mandatory	Version
Indication	Yes	1.2.0
Filesystem	Yes	1.2.0
File Storage	Yes	1.2.0
File Export	Yes	1.2.0
Access Points	No	1.2.0
Multiple Computer System	No	1.2.0
Software	No	1.2.0
Location	No	1.2.0
Extent Composition	No	1.2.0

**Table 133: Supported Profiles for Self-contained NAS System**

Registered Profile Names	Mandatory	Version
Filesystem Manipulation	No	1.2.0
File Export Manipulation	No	1.2.0
Job Control	No	1.2.0
Disk Drive Lite	No	1.2.0
SPI Initiator Ports	No	1.2.0
FC Initiator Ports	No	1.2.0
iSCSI Initiator Ports	No	1.2.0
Device Credentials	No	1.2.0
Physical Package	Yes	1.2.0
Block Services	Yes	1.2.0
Health	Yes	1.2.0

## 11.5 Methods of the Profile

### 11.5.1 Extrinsic Methods of the Profile

None.

### 11.5.2 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.

## 11.6 Client Considerations and Recipes

Not defined in this version of the specification

## **11.7 Registered Name and Version**

Self-contained NAS System version 1.2.0

## 11.8 CIM Elements

**Table 134: CIM Elements for Self-contained NAS System**

Element Name	Requirement	Description
CIM_BindsTo (CIFS or NFS) (11.8.1)	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.
CIM_BindsTo (TCP) (11.8.2)	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.
CIM_BindsToLANEndpoint (11.8.3)	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).
CIM_ComputerSystem (Top Level) (11.8.4)	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.
CIM_ComputerSystem (File Server) (11.8.5)	Mandatory	This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top-level system. The File Server(s) shall be manageable as a computer system and so could be exposed through other profiles and so there shall be a way to correlate it with other management clients.
CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort) (11.8.6)	Mandatory	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort) (11.8.7)	Conditional	Conditional requirement: This is required if a LANEndpoint exists. (LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
CIM_HostedAccessPoint (CIFS or NFS) (11.8.8)	Mandatory	(CIFS or NFS) Represents the association between a front end ProtocolEndpoint and the Computer System that hosts it.

**Table 134: CIM Elements for Self-contained NAS System**

Element Name	Requirement	Description
CIM_HostedAccessPoint (TCP) (11.8.9)	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.
CIM_HostedAccessPoint (IP) (11.8.10)	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.
CIM_HostedAccessPoint (LAN) (11.8.11)	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.
CIM_IPProtocolEndpoint (11.8.12)	Optional	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
CIM_LANEndpoint (11.8.13)	Optional	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
CIM_LogicalDisk (Disk for FS) (11.8.14)	Mandatory	Represents LogicalDisks used for building LocalFileSystems.
CIM_NetworkPort (11.8.15)	Mandatory	Represents the front end logical port that supports access to a local area network.
CIM_ProtocolEndpoint (CIFS or NFS) (11.8.16)	Mandatory	(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
CIM_SystemDevice (Logical Disks) (11.8.17)	Mandatory	This association links all LogicalDisks to the scoping system.
CIM_SystemDevice (Network Ports) (11.8.18)	Mandatory	This association links all NetworkPorts to the scoping system.
CIM_TCPProtocolEndpoint (11.8.19)	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	Experimental CQL - Change of Status of a NAS ComputerSystem (controller).  PreviousInstance is optional, but may be supplied by an implementation of the Profile.

**Table 134: CIM Elements for Self-contained NAS System**

Element Name	Requirement	Description
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	Experimental 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	Experimental 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_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.

**11.8.1 CIM\_BindsTo (CIFS or NFS)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: TCPProtocolEndpoint

Table 135 describes class CIM\_BindsTo (CIFS or NFS).

**Table 135: 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.

### 11.8.2 CIM\_BindsTo (TCP)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: IPProtocolEndpoint

Table 136 describes class CIM\_BindsTo (TCP).

**Table 136: 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.

### 11.8.3 CIM\_BindsToLANEndpoint

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: LANEndpoint

Table 137 describes class CIM\_BindsToLANEndpoint.

**Table 137: 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.

**Table 137: SMI Referenced Properties/Methods for CIM\_BindsToLANEndpoint**

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	A LANEndpoint.

**11.8.4 CIM\_ComputerSystem (Top Level)**

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 138 describes class CIM\_ComputerSystem (Top Level).

**Table 138: 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.
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.



**Table 138: SMI Referenced Properties/Methods for CIM\_ComputerSystem (Top Level)**

Properties	Flags	Requirement	Description & Notes
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.

**11.8.5 CIM\_ComputerSystem (File Server)**

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 139 describes class CIM\_ComputerSystem (File Server).

**Table 139: SMI Referenced Properties/Methods for CIM\_ComputerSystem (File Server)**

Properties	Flags	Requirement	Description & Notes
Dedicated		Mandatory	This 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 File Servers. Eg Vendor/Model/SerialNo+FS+Number. The Fileserver can have any number of IP addresses, so an IP address does not constitute a single unique id. Also, under various load-balancing or redundancy regimens, the IP address could move around, so it may not even be correlatable. For that reason, the vendor shall support a format that will provide a unique id for the file server.
OperationalStatus		Mandatory	Overall status of the File Server.
Caption	N	Optional	Not Specified in this version of the Profile.

**Table 139: SMI Referenced Properties/Methods for CIM\_ComputerSystem (File Server)**

Properties	Flags	Requirement	Description & Notes
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.

**11.8.6 CIM\_DeviceSAPImplementation (CIFS or NFS to NetworkPort)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 140 describes class CIM\_DeviceSAPImplementation (CIFS or NFS to NetworkPort).

**Table 140: 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.

#### 11.8.7 CIM\_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LANEndpoint

Table 141 describes class CIM\_DeviceSAPImplementation (LANEndpoint to NetworkPort).

**Table 141: 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.

#### 11.8.8 CIM\_HostedAccessPoint (CIFS or NFS)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 142 describes class CIM\_HostedAccessPoint (CIFS or NFS).

**Table 142: 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 always FileServers (Dedicated=16).

#### 11.8.9 CIM\_HostedAccessPoint (TCP)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: TCPProtocolEndpoint

Table 143 describes class CIM\_HostedAccessPoint (TCP).

**Table 143: 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.

#### 11.8.10 CIM\_HostedAccessPoint (IP)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: IPProtocolEndpoint

Table 144 describes class CIM\_HostedAccessPoint (IP).

**Table 144: 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.

**11.8.11 CIM\_HostedAccessPoint (LAN)**

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: LANEndpoint

Table 145 describes class CIM\_HostedAccessPoint (LAN).

**Table 145: 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.

**11.8.12 CIM\_IPProtocolEndpoint**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 146 describes class CIM\_IPProtocolEndpoint.

**Table 146: SMI Referenced Properties/Methods for CIM\_IPProtocolEndpoint**

Properties	Flags	Requirement	Description & Notes
SystemCreationClass		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.
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		Mandatory	An IP v4 address in the format "A.B.C.D".
IPv6Address		Mandatory	

**Table 146: SMI Referenced Properties/Methods for CIM\_IPProtocolEndpoint**

Properties	Flags	Requirement	Description & Notes
SubnetMask		Mandatory	An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		Mandatory	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.

**11.8.13 CIM\_LANEndpoint**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 147 describes class CIM\_LANEndpoint.

**Table 147: 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.

**Table 147: SMI Referenced Properties/Methods for CIM\_LANEndpoint**

Properties	Flags	Requirement	Description & Notes
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.

**11.8.14 CIM\_LogicalDisk (Disk for FS)**

Created By: Extrinsic\_or\_External

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 148 describes class CIM\_LogicalDisk (Disk for FS).

**Table 148: 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.
IdentifyingDescriptions	N	Optional	Not Specified in this version of the Profile.
AdditionalAvailability	N	Optional	Not Specified in this version of the Profile.



**Table 148: SMI Referenced Properties/Methods for CIM\_LogicalDisk (Disk for FS)**

Properties	Flags	Requirement	Description & Notes
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.
OtherNameNamespa ce	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.

**11.8.15 CIM\_NetworkPort**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 149 describes class CIM\_NetworkPort.

**Table 149: 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.
SystemCreationClas sName		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.
DeviceID		Mandatory	A unique ID for the device (in the context of the hosting System).

**Table 149: SMI Referenced Properties/Methods for CIM\_NetworkPort**

Properties	Flags	Requirement	Description & Notes
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.
OtherIdentifyingInfo	N	Optional	Not Specified in this version of the Profile.

**Table 149: SMI Referenced Properties/Methods for CIM\_NetworkPort**

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.
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.

**11.8.16 CIM\_ProtocolEndpoint (CIFS or NFS)**

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 150 describes class CIM\_ProtocolEndpoint (CIFS or NFS).

**Table 150: 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 150: SMI Referenced Properties/Methods for CIM\_ProtocolEndpoint (CIFS or NFS)**

Properties	Flags	Requirement	Description & Notes
ProtocolType		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.

**11.8.17 CIM\_SystemDevice (Logical Disks)**

Created By: Extrinsic\_or\_External\_or\_Static

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 151 describes class CIM\_SystemDevice (Logical Disks).

**Table 151: 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. These include NetworkPorts.

**11.8.18 CIM\_SystemDevice (Network Ports)**

Created By: Extrinsic\_or\_External\_or\_Static

Modified By: Extrinsic\_or\_External

Deleted By: Extrinsic\_or\_External

Class Mandatory: Mandatory

Table 152 describes class CIM\_SystemDevice (Network Ports).

**Table 152: 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.

#### 11.8.19 CIM\_TCPProtocolEndpoint

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Optional

Table 153 describes class CIM\_TCPProtocolEndpoint.

**Table 153: 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.

**Table 153: SMI Referenced Properties/Methods for CIM\_TCPProtocolEndpoint**

Properties	Flags	Requirement	Description & Notes
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**


---



---

## 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).

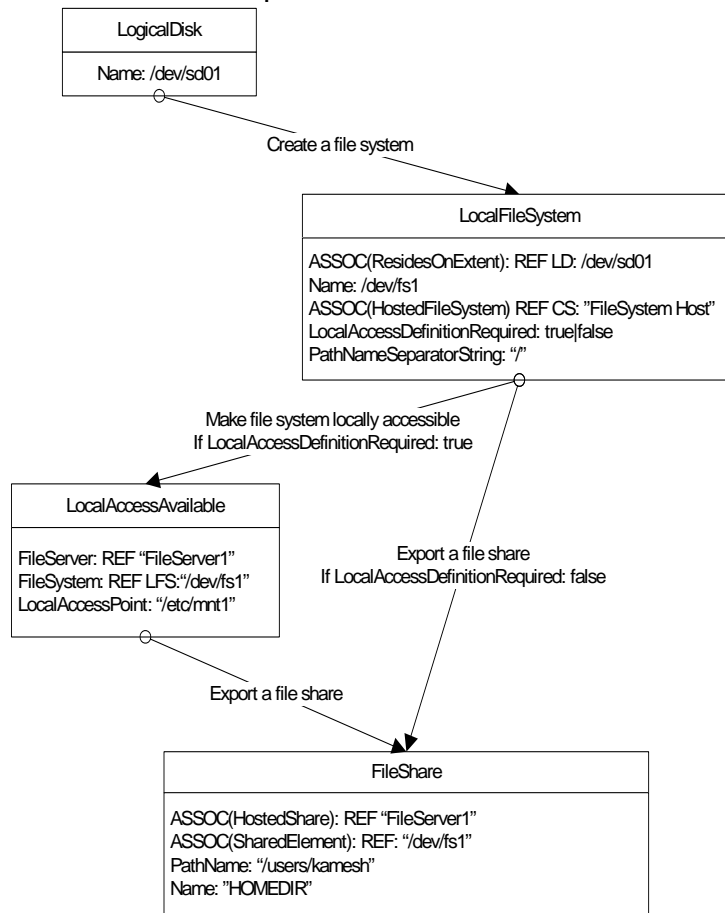
In filesystem-related profiles, the collection of storage elements used by a *LocalFileSystem* is called a *LogicalDisk(s)*. There are multiple formats (both open and proprietary) for structuring a *LogicalDisk* into a *LocalFileSystem*—how the metadata, attributes, and content are stored and so on—that 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*. We represent the type of the *LocalFileSystem* in this and related profiles 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*. We represent this 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 we require 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 21. We begin 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*—in Figure 21 we show the name and the *ComputerSystem* that has a *HostedFileSystem* association to the *LocalFileSystem*. We skip other details of the *LocalFileSystem*.

**Figure 21: 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 we will be using “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 21, 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 FileSystemName 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 LocalFileSystemName 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 21 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).