



Storage Management Technical Specification, Part 3 Block Devices

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."

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 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

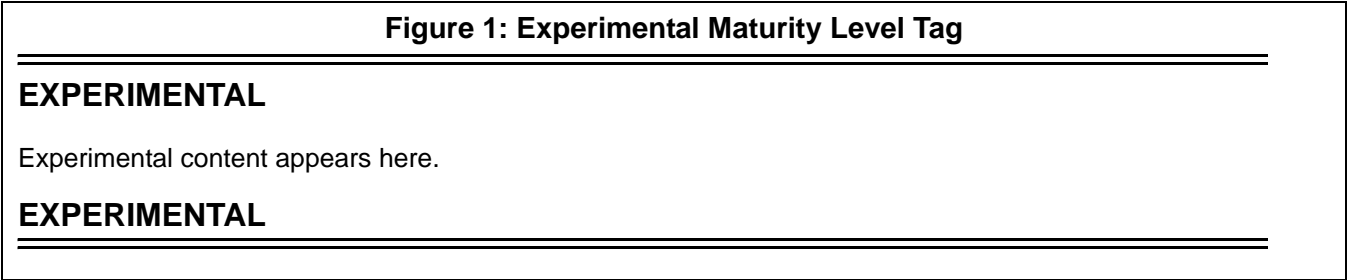
This specification has been structured to convey both the formal requirements and assumptions of the SMI-S API and its emerging implementation and deployment lifecycle. Over time, the intent is that all content in the specification will represent a mature and stable design, be verified by extensive implementation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

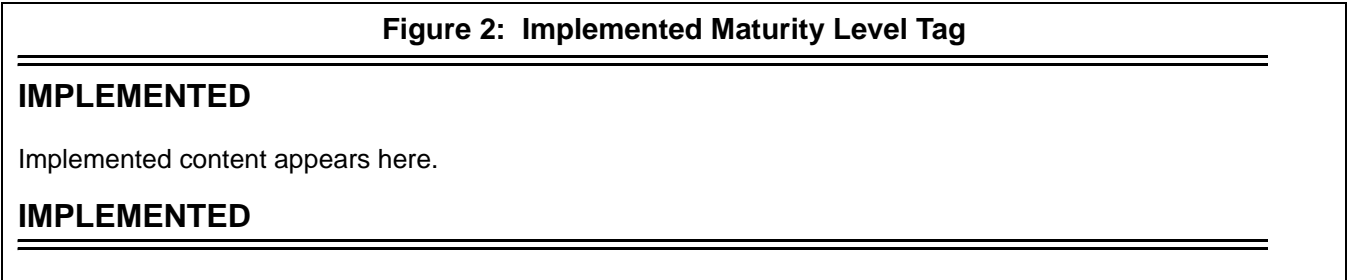
Experimental Maturity Level

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

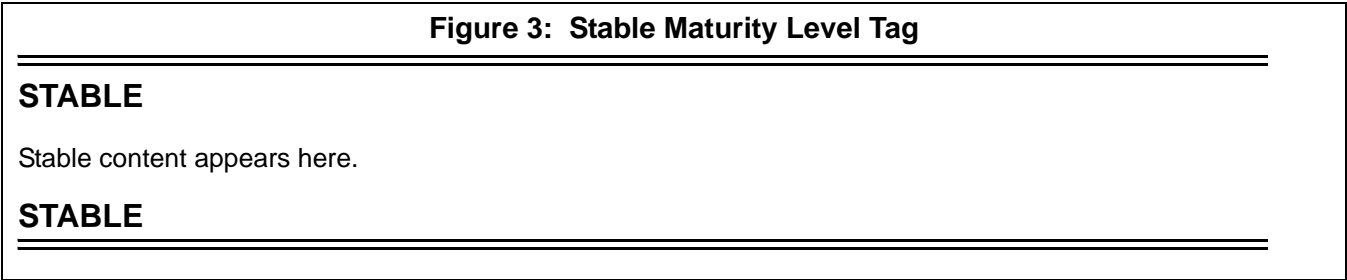
Profiles for which initial implementations have been completed are classified as “Implemented”. This indicates that at least two different vendors have implemented the profile, including at least one provider implementation. At this maturity level, the underlying architecture and modeling are stable,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



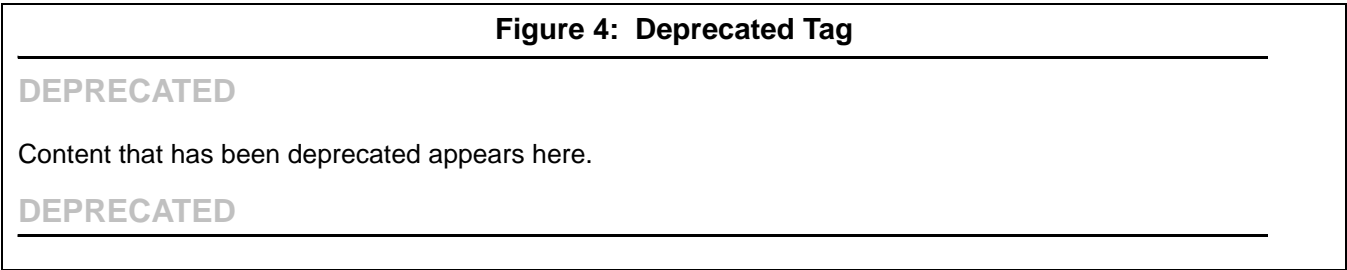
Finalized Maturity Level

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 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 Figures	xiii
List of Tables	xvii
Foreword.....	xxvii
1. Scope	1
2. Normative References	3
2.1 Approved references	3
2.2 References under development	3
2.3 Other references.....	3
3. Terms and definitions	5
3.1 Definitions.....	5
4. Array Profile	7
4.1 Description.....	7
4.2 Health and Fault Management	9
4.3 Cascading Considerations.....	9
4.4 Supported Subprofiles and Packages	9
4.5 Methods of the Profile.....	10
4.6 Client Considerations and Recipes	10
4.7 Registered Name and Version	10
4.8 CIM Elements	11
5. Block Services Package.....	15
5.1 Description.....	15
5.2 Health and Fault Management Considerations	33
5.3 Cascading Considerations.....	34
5.4 Supported Profile, Subprofiles and Packages	34
5.5 Methods of this Profile	34
5.6 Client Considerations and Recipes	48
5.7 Registered Name and Version	75
5.8 CIM Elements	76
6. Block Storage Views Profile	101
6.1 Description.....	101
6.2 Health and Fault Management Consideration	111
6.3 Cascading Considerations.....	111
6.4 Supported Profiles, Subprofiles, and Packages	111
6.5 Methods of the Profile.....	112
6.6 Client Considerations and Recipes	112
6.7 Registered Name and Version	112
6.8 CIM Elements	113
7. Block Server Performance Subprofile	133
7.1 Description.....	133
7.2 Implementation	135
7.3 Health and Fault Management Considerations	153
7.4 Cascading Considerations.....	154
7.5 Supported Subprofiles and Packages	154
7.6 Methods of the Profile.....	154
7.7 Client Considerations and Recipes	161
7.8 Registered Name and Version	186
7.9 CIM Elements	187
8. CKD Block Services Profile	213
8.1 Description.....	213
8.2 Health and Fault Management Consideration	216
8.3 Cascading Considerations.....	216

8.4	Supported Profiles, Subprofiles, and Packages	216
8.5	Methods of the Profile.....	216
8.6	Client Considerations and Recipes	216
8.7	Registered Name and Version	216
8.8	CIM Elements	217
9.	Copy Services Subprofile	243
9.1	Description.....	243
9.2	Health and Fault Management Considerations	262
9.3	Cascading Considerations.....	263
9.4	Supported Subprofiles and Packages	263
9.5	Methods of the Profile.....	265
9.6	Client Considerations and Recipes	272
9.7	Registered Name and Version	297
9.8	CIM Elements	298
10.	Disk Drive Subprofile	319
11.	Disk Drive Lite Subprofile	321
11.1	Description.....	321
11.2	Health and Fault Management Considerations	323
11.3	Cascading Considerations.....	323
11.4	Supported Subprofiles and Packages	323
11.5	Methods of this Profile	323
11.6	Registered Name and Version	323
11.7	CIM Elements	324
12.	Disk Sparring Subprofile	333
12.1	Description.....	333
12.2	Health and Fault Management Considerations	338
12.3	Cascading Conjurations	339
12.4	Supported Subprofiles and Packages	339
12.5	Methods of the Profile.....	339
12.6	Client Considerations and Recipes	341
12.7	Registered Name and Version	342
12.8	CIM Elements	343
13.	Erase Profile	355
13.1	Description.....	355
13.2	Health and Fault Management Considerations	357
13.3	Cascading Considerations.....	357
13.4	Supported Profiles, Subprofiles, and Packages	357
13.5	Methods of the Profile.....	357
13.6	Client Considerations and Recipes	358
13.7	Registered Name and Version	362
13.8	CIM Elements	362
14.	Extent Mapping Subprofile	367
15.	Extent Composition Subprofile	369
15.1	Description.....	369
15.2	Health and Fault Management Considerations	383
15.3	Cascading Considerations.....	383
15.4	Supported Subprofiles and Packages	383
15.5	Methods of the Profile.....	383
15.6	Client Considerations and Recipes	384
15.7	Registered Name and Version	389
15.8	CIM Elements	390
16.	LUN Creation Subprofile	401
17.	LUN Mapping and Masking Subprofile	403
17.1	Compatibility with SMI-S 1.0 clients.	403

18. Masking and Mapping Subprofile	405
18.1 Description.....	405
18.2 Health and Fault Management Considerations	414
18.3 Cascading Considerations.....	414
18.4 Supported Subprofiles, and Packages	414
18.5 Methods of the Profile.....	414
18.6 Client Considerations and Recipes	425
18.7 Registered Name and Version	436
18.8 CIM Elements	437
19. Pool Management Policy Subprofile	457
19.1 Description.....	457
19.2 Health and Fault Management Considerations	460
19.3 Cascading Considerations.....	460
19.4 Supported Subprofiles and Packages	460
19.5 Methods of the Profile.....	460
19.6 Client Considerations and Recipes	462
19.7 Required CIM Elements	463
19.8 Classes Used in the Profile	464
19.9 Dependencies on Other Standards	475
20. Pool Manipulation Capabilities, and Settings Subprofile	477
21. Storage Server Asymmetry Profile	479
21.1 Description.....	479
21.2 Health and Fault Management Consideration	488
21.3 Cascading Considerations.....	488
21.4 Supported Profiles, Subprofiles, and Packages	488
21.5 Methods of the Profile.....	488
21.6 Client Considerations and Recipes	489
21.7 Registered Name and Version	490
21.8 CIM Elements	491
22. Resource Ownership Subprofile	503
22.1 Description.....	503
22.2 Client Considerations and Recipes	508
23. Storage Virtualizer Profile.....	511
23.1 Description.....	511
23.2 Health and Fault Management	513
23.3 Cascading Considerations.....	514
23.4 Supported Subprofiles and Packages	515
23.5 Methods of the Profile.....	516
23.6 Client Considerations and Recipes	517
23.7 Registered Name and Version	517
23.8 CIM Elements	518
24. Volume Composition Profile.....	523
24.1 Description.....	523
24.2 Health and Fault Management Consideration	529
24.3 Cascading Considerations.....	530
24.4 Supported Profiles, Subprofiles, and Packages	530
24.5 Methods of the Profile.....	530
24.6 Client Considerations and Recipes	536
24.7 Registered Name and Version	541
24.8 CIM Elements	542
25. Volume Management Profile.....	551
25.1 Description.....	551
25.2 Health and Fault Management Considerations	552
25.3 Cascading Considerations.....	552

25.4	Supported Subprofiles and Packages	553
25.5	Methods of the Profile.....	553
25.6	Client Considerations and Recipes	553
25.7	Registered Name and Version	554
25.8	CIM Elements	555
26.	Storage Element Protection SubProfile.....	565
26.1	Description.....	565
26.2	Health and Fault Management Consideration	576
26.3	Cascading Considerations.....	576
26.4	Supported Profiles, Subprofiles, and Packages	576
26.5	Methods of the Profile.....	577
26.6	Client Considerations and Recipes	578
26.7	Registered Name and Version	582
26.8	CIM Elements	583

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.	Array Profile Instance Diagram	7
Figure 6.	Array Package Diagram.....	8
Figure 7.	Storage Capacity State	15
Figure 8.	StoragePool Manipulation Instance Diagram.....	17
Figure 9.	StorageVolume Creation Instance Diagram	21
Figure 10.	Storage Configuration	23
Figure 11.	StorageExtent Conservation - Step 1	29
Figure 12.	StorageExtent Conservation - Step 2	30
Figure 13.	StorageExtent Conservation - Step 3	31
Figure 14.	Representative Block Service Instance Diagram.....	48
Figure 15.	StoragePool Creation - Initial State.....	49
Figure 16.	StoragePool Creation - Step 1	50
Figure 17.	StoragePool Creation - Step 2	50
Figure 18.	StoragePool Creation - Step 3	51
Figure 19.	StorageVolume Creation - Initial State.....	52
Figure 20.	StorageVolume Creation - Step 1	52
Figure 21.	StorageVolume Creation - Step 2	53
Figure 22.	StorageVolume Creation - Step 3	54
Figure 23.	Object Diagram for SNIA_ View Classes.....	103
Figure 24.	Block Storage View Class Capabilities	104
Figure 25.	SNIA_ VolumeView and related associations.....	105
Figure 26.	SNIA_ DiskDriveView and related associations	107
Figure 27.	SNIA_ ExposedView Association	109
Figure 28.	SNIA_ MaskingMapView Association	110
Figure 29.	Block Server Performance Subprofile Summary Instance Diagram	136
Figure 30.	Base Array Profile Block Server Performance Instance Diagram.....	139
Figure 31.	Base Storage Virtualizer Profile Block Server Performance Instance Diagram.....	141
Figure 32.	Base Volume Management Profile Block Server Performance Instance Diagram	143
Figure 33.	Multiple Computer System Subprofile Block Server Performance Instance Diagram	145
Figure 34.	Fibre Channel Initiator Port Subprofile Block Server Performance Instance Diagram.....	146
Figure 35.	Extent Composition Subprofile Block Server Performance Instance Diagram	147
Figure 36.	Disk Drive Lite Subprofile Block Server Performance Instance Diagram	148
Figure 37.	SCSIArbitraryLogicalUnit Block Server Performance Instance Diagram	149
Figure 38.	Remote Mirrors Block Server Performance Instance Diagram	150
Figure 39.	Block Server Performance Manifest Collections.....	152
Figure 40.	Block Services Support for Count Key Data Storage.....	214
Figure 41.	Copy Services Discovery	245
Figure 42.	Local Replica	246

Figure 43. Remote Mirror Replica	247
Figure 44. Cascading the Copy Services Subprofile.....	248
Figure 45. Peer-to-Peer Connection	249
Figure 46. Multi-Level Local Replication	250
Figure 47. Multi-Level Remote Replication	251
Figure 48. Multiple Snapshots Per Source Element	252
Figure 49. State Transitions for Mirrors and Clones	258
Figure 50. State Transitions for Snapshots and Migration	259
Figure 51. Remote Replication Buffer	275
Figure 52. Fixed Space Consumption.....	280
Figure 53. Variable Space Consumption	281
Figure 54. Disk Drive Lite Instance Model	322
Figure 55. Sparing Instance Diagram	333
Figure 56. Variations of RS per Storage Element.....	335
Figure 57. Before Failure	336
Figure 58. During Failure	337
Figure 59. After Failure	338
Figure 60. Model Elements	357
Figure 61. Volume Composition from General QOS Pool.....	371
Figure 62. Single QOS Pool Composition (RAID Groups).....	372
Figure 63. Single QOS Pool Composition - Two Concretes	373
Figure 64. Concatenation Composition.....	375
Figure 65. RAID 0 Composition	375
Figure 66. RAID 1 Composition	376
Figure 67. RAID 10 Composition	377
Figure 68. RAID 0+1 Composition	378
Figure 69. RAID 4, 5 Composition	379
Figure 70. RAID 6, 5DP, 4DP	380
Figure 71. RAID 15 Composition	381
Figure 72. RAID 50 Composition	382
Figure 73. RAID 51 Composition	383
Figure 74. Generic System with no Configuration Service.....	406
Figure 75. Generic System with ControllerConfigurationService	407
Figure 76. Relationship of Initiator IDs, Endpoints, and Logical Units	408
Figure 77. StorageClientSettingData Model.....	411
Figure 78. Entire Model.....	412
Figure 79. Basic Pool Management Instance Diagram.....	458
Figure 80. Storage Asymmetry Class Hierarchy	481
Figure 81. Asymmetry with MCS.....	483
Figure 82. Ports Do Not Failover, Healthy	484
Figure 83. Ports Do Not Failover, Failed Controller	485
Figure 84. Ports Failover, Healthy.....	486
Figure 85. Ports Failover, Failed Controller	487
Figure 86. Resource Ownership for Block Services.....	503
Figure 87. ServiceAffectsElement Associations for ResourceOwnership.....	507
Figure 88. AuthorizedPrivilege Associations for ResourceOwnership	507

Figure 89. Storage Virtualizer Package Diagram.....	511
Figure 90. Storage Virtualizer System Instance.....	512
Figure 91. Virtualizer, Cascading and Initiator Ports.....	515
Figure 92. Volume Composition Class Mode.....	524
Figure 93. Example 1 Step 1.....	526
Figure 94. Example 1 Step 2.....	527
Figure 95. Example 2 Step 1.....	528
Figure 96. Example 2 Step 2.....	529
Figure 97. Volume Management Instance Diagram.....	551
Figure 98. Storage Element Protection Class Model	566
Figure 99. Retention Time Line.....	570
Figure 100. Protection State Transition Diagram.....	571
Figure 101. Step 1 - Initial State	572
Figure 102. Step 2 - Volume Set to Read-only	573
Figure 103. Step 3 - Second Volume Set to Read-only	574
Figure 104. Step 4 - Volume Set to Read/Write Disabled.....	575
Figure 105. Step 5 Volume Access Changed	576

List of Tables

Table 1.	Supported Profiles for Array	9
Table 2.	CIM Elements for Array	11
Table 3.	SMI Referenced Properties/Methods for CIM_ComputerSystem	11
Table 4.	SMI Referenced Properties/Methods for CIM_SystemDevice (System to SCSIProtocolController)	12
Table 5.	SMI Referenced Properties/Methods for CIM_SystemDevice (System to SCSIArbitraryLogicalUnit)	12
Table 6.	SMI Referenced Properties/Methods for CIM_SCSIArbitraryLogicalUnit	13
Table 7.	SMI Referenced Properties/Methods for CIM_SCSIProtocolController	14
Table 8.	Mapping: Supported Actions to Methods.....	19
Table 9.	SupportedStoragePoolFeatures Array - Behavior	20
Table 10.	SupportedStoragePoolFeatures Array - Special Features	20
Table 11.	RAID Mapping	24
Table 12.	Meaning of Usage values	26
Table 13.	Classes Required In Read-Only Implementation	27
Table 14.	Standard Messages for Block Services Package	34
Table 15.	Supported Profiles for Block Services	34
Table 16.	CIM Elements for Block Services	76
Table 17.	SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)	79
Table 18.	SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)	80
Table 19.	SMI Referenced Properties/Methods for CIM_HostedStoragePool	81
Table 20.	SMI Referenced Properties/Methods for CIM_HostedService	82
Table 21.	SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities	82
Table 22.	SMI Referenced Properties/Methods for CIM_StorageConfigurationService	83
Table 23.	SMI Referenced Properties/Methods for CIM_OwningJobElement	84
Table 24.	SMI Referenced Properties/Methods for CIM_StorageCapabilities	84
Table 25.	SMI Referenced Properties/Methods for CIM_StorageSetting	87
Table 26.	SMI Referenced Properties/Methods for CIM_StorageSettingWithHints	88
Table 27.	SMI Referenced Properties/Methods for CIM_ElementSettingData	90
Table 28.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageCapabilities to StoragePool)	90
Table 29.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)	91
Table 30.	SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool)	91
Table 31.	SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)	92
Table 32.	SMI Referenced Properties/Methods for CIM_StorageVolume	92
Table 33.	SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageVolume or LogicalDisk)	93
Table 34.	SMI Referenced Properties/Methods for CIM_BasedOn (StorageVolume to extent from Extent Composition)	94
Table 35.	SMI Referenced Properties/Methods for CIM_LogicalDisk	94
Table 36.	SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to extent from Extent Composition)	95
Table 37.	SMI Referenced Properties/Methods for CIM_StorageSettingsAssociatedToCapabilities	96
Table 38.	SMI Referenced Properties/Methods for CIM_StorageSettingsGeneratedFromCapabilities	96
Table 39.	SMI Referenced Properties/Methods for CIM_ConcreteComponent (storage element to extent from Extent Composition)	97
Table 40.	SMI Referenced Properties/Methods for CIM_AssociatedComponentExtent (StoragePool to extent from Composition)	97
Table 41.	SMI Referenced Properties/Methods for CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)	97
Table 42.	SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities	98
Table 43.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)	99

Table 44.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)	99
Table 45.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool).....	99
Table 46.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)	100
Table 47.	Supported Profiles for Block Storage Views.....	111
Table 48.	CIM Elements for Block Storage Views.....	113
Table 49.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (View Capabilities)	116
Table 50.	SMI Referenced Properties/Methods for SNIA_AllocatedFromStoragePoolView	117
Table 51.	SMI Referenced Properties/Methods for SNIA_BaseInstance (StorageSetting).....	117
Table 52.	SMI Referenced Properties/Methods for SNIA_BaseInstance (Volume)	118
Table 53.	SMI Referenced Properties/Methods for SNIA_BaseInstance (DiskDrive)	118
Table 54.	SMI Referenced Properties/Methods for SNIA_BasedOnView (ExtentOnDriveExtent)	119
Table 55.	SMI Referenced Properties/Methods for SNIA_BasedOnView (VolumeOnExtent)	119
Table 56.	SMI Referenced Properties/Methods for SNIA_ConcreteComponentView	120
Table 57.	SMI Referenced Properties/Methods for SNIA_ContainerView	120
Table 58.	SMI Referenced Properties/Methods for SNIA_DiskDriveView.....	121
Table 59.	SMI Referenced Properties/Methods for SNIA_ElementStatisticalDataView (VolumeView)	123
Table 60.	SMI Referenced Properties/Methods for SNIA_ElementStatisticalDataView (DiskDriveView)	124
Table 61.	SMI Referenced Properties/Methods for SNIA_ExposedView	124
Table 62.	SMI Referenced Properties/Methods for SNIA_MaskingMapView.....	126
Table 63.	SMI Referenced Properties/Methods for SNIA_ViewCapabilities	127
Table 64.	SMI Referenced Properties/Methods for SNIA_VolumeView.....	128
Table 65.	SMI Referenced Properties/Methods for SNIA_SystemDeviceView (DiskDriveViews).....	131
Table 66.	SMI Referenced Properties/Methods for SNIA_SystemDeviceView (VolumeViews).....	131
Table 67.	Summary of Element Types by Profile	144
Table 68.	Supported Profiles for Block Server Performance.....	154
Table 69.	Creation, Deletion and Modification Methods in Block Server Performance Subprofile.....	154
Table 70.	Summary of Statistics Support by Element	183
Table 71.	Formulas and Calculations	185
Table 72.	Block Server Performance Subprofile Supported Capabilities Patterns	186
Table 73.	CIM Elements for Block Server Performance.....	187
Table 74.	SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Provider defined collection)	190
Table 75.	SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Client defined collection)	191
Table 76.	SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Provider Support)	191
Table 77.	SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Client Defined).....	193
Table 78.	SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Provider Defined).....	195
Table 79.	SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Client Defined)	195
Table 80.	SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData	196
Table 81.	SMI Referenced Properties/Methods for CIM_BlockStatisticsCapabilities	200
Table 82.	SMI Referenced Properties/Methods for CIM_BlockStatisticsService	201
Table 83.	SMI Referenced Properties/Methods for CIM_StatisticsCollection	203
Table 84.	SMI Referenced Properties/Methods for CIM_ElementCapabilities.....	203
Table 85.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Top Level System Stats)	204
Table 86.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Component System Stats)	204
Table 87.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Volume Stats).....	205
Table 88.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Logical Disk Stats)	206

Table 89.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Front end Port Stats)	206
Table 90.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Back end Port Stats)	207
Table 91.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Remote Copy Stats)	207
Table 92.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Extent Stats)	208
Table 93.	SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Disk Stats)	208
Table 94.	SMI Referenced Properties/Methods for CIM_HostedService	209
Table 95.	SMI Referenced Properties/Methods for CIM_HostedCollection (Provider Supplied)	209
Table 96.	SMI Referenced Properties/Methods for CIM_HostedCollection (Default)	210
Table 97.	SMI Referenced Properties/Methods for CIM_HostedCollection (Client Defined)	210
Table 98.	SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of statistics collection)	211
Table 99.	SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of pre-defined collection)	211
Table 100.	SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of client defined collection)	211
Table 101.	Supported Profiles for CKD Block Services	216
Table 102.	CIM Elements for CKD Block Services	217
Table 103.	SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)	222
Table 104.	SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)	222
Table 105.	SMI Referenced Properties/Methods for CIM_HostedStoragePool	223
Table 106.	SMI Referenced Properties/Methods for CIM_HostedService	224
Table 107.	SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities	224
Table 108.	SMI Referenced Properties/Methods for CIM_StorageConfigurationService	225
Table 109.	SMI Referenced Properties/Methods for CIM_OwningJobElement	226
Table 110.	SMI Referenced Properties/Methods for SNIA_StorageCapabilities	227
Table 111.	SMI Referenced Properties/Methods for SNIA_StorageSetting	228
Table 112.	SMI Referenced Properties/Methods for CIM_StorageSettingWithHints	229
Table 113.	SMI Referenced Properties/Methods for CIM_ElementSettingData	231
Table 114.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageCapabilities to StoragePool)	231
Table 115.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)	232
Table 116.	SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool)	232
Table 117.	SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)	233
Table 118.	SMI Referenced Properties/Methods for SNIA_StorageVolume	233
Table 119.	SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageVolume or LogicalDisk)	235
Table 120.	SMI Referenced Properties/Methods for CIM_BasedOn (StorageVolume to extent from Extent Composition)	235
Table 121.	SMI Referenced Properties/Methods for CIM_LogicalDisk	235
Table 122.	SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to extent from Extent Composition)	237
Table 123.	SMI Referenced Properties/Methods for CIM_StorageSettingsAssociatedToCapabilities	237
Table 124.	SMI Referenced Properties/Methods for CIM_StorageSettingsGeneratedFromCapabilities	238
Table 125.	SMI Referenced Properties/Methods for CIM_ConcreteComponent (storage element to extent from Extent Composition)	238
Table 126.	SMI Referenced Properties/Methods for CIM_AssociatedComponentExtent (StoragePool to extent from Composition)	238
Table 127.	SMI Referenced Properties/Methods for CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)	239
Table 128.	SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities	239
Table 129.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)	240
Table 130.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)	241
Table 131.	SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)	241

Table 132. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)	241
Table 133. Synchronization Operation Support Requirements	254
Table 134. SyncState Values	255
Table 135. SyncMaintained and WhenSynced Properties	256
Table 136. OperationalStatus Values for ReplicationPipe.....	262
Table 137. Copy Services Alert Indications.....	263
Table 138. Copy Services Error Responses	263
Table 139. Extrinsic Methods of ReplicationServices Subprofile	265
Table 140. ModifySynchronization	265
Table 141. CreateReplica Method.....	266
Table 142. TargetPool Parameter for Delta Replicas.....	267
Table 143. AttachOrModifyReplica Method.....	269
Table 144. CreateReplicationBuffer Method	270
Table 145. CreateOrModifyReplicationPipe Method	271
Table 146. Replica Specialization by CopyType	272
Table 147. Patterns Supported for StorageReplicationCapabilities	281
Table 148. Space Consumption Properties.....	283
Table 149. Space Consumption Properties, Fixed Pattern	283
Table 150. CIM Elements for Copy Services.....	298
Table 151. SMI Referenced Properties/Methods for CIM_AssociatedMemory	300
Table 152. SMI Referenced Properties/Methods for CIM_BasedOn ()	300
Table 153. SMI Referenced Properties/Methods for CIM_ConcreteDependency	301
Table 154. SMI Referenced Properties/Methods for CIM_ElementCapabilities (Associates StorageReplicationCapabilities and StorageConfigurationService)	301
Table 155. SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe	302
Table 156. SMI Referenced Properties/Methods for CIM_HostedNetworkPipe.....	302
Table 157. SMI Referenced Properties/Methods for CIM_Memory.....	303
Table 158. SMI Referenced Properties/Methods for CIM_Network	303
Table 159. SMI Referenced Properties/Methods for SNIA_ReplicationPipe.....	304
Table 160. SMI Referenced Properties/Methods for CIM_NetworkPipeComposition	304
Table 161. SMI Referenced Properties/Methods for CIM_ProtocolEndpoint	305
Table 162. SMI Referenced Properties/Methods for CIM_ReplicaPoolForStorage	305
Table 163. SMI Referenced Properties/Methods for CIM_StorageCapabilities	306
Table 164. SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities	306
Table 165. SMI Referenced Properties/Methods for CIM_StorageConfigurationService.....	307
Table 166. SMI Referenced Properties/Methods for CIM_StoragePool.....	308
Table 167. SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities.....	309
Table 168. SMI Referenced Properties/Methods for CIM_StorageSetting.....	314
Table 169. SMI Referenced Properties/Methods for CIM_StorageSynchronized (Between LogicalDisk elements).....	315
Table 170. SMI Referenced Properties/Methods for CIM_StorageSynchronized (Between StorageVolume elements)	317
Table 171. SMI Referenced Properties/Methods for CIM_SystemComponent	317
Table 172. OperationalStatus For DiskDrive	323
Table 173. CIM Elements for Disk Drive Lite.....	324
Table 174. SMI Referenced Properties/Methods for CIM_ConcreteComponent	324
Table 175. SMI Referenced Properties/Methods for CIM_Container	325
Table 176. SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation	325
Table 177. SMI Referenced Properties/Methods for CIM_DiskDrive	326
Table 178. SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity.....	326
Table 179. SMI Referenced Properties/Methods for CIM_MediaPresent	327

Table 180. SMI Referenced Properties/Methods for CIM_ProtocolControllerAccessesUnit	327
Table 181. SMI Referenced Properties/Methods for CIM_PhysicalPackage	328
Table 182. SMI Referenced Properties/Methods for CIM_Realizes	328
Table 183. SMI Referenced Properties/Methods for CIM_SAPAvailableForElement	329
Table 184. SMI Referenced Properties/Methods for CIM_SCSIInitiatorTargetLogicalUnitPath	329
Table 185. SMI Referenced Properties/Methods for CIM_SoftwareIdentity	329
Table 186. SMI Referenced Properties/Methods for CIM_StorageExtent	330
Table 187. SMI Referenced Properties/Methods for CIM_SystemDevice	331
Table 188. Supported Methods to Method Mapping	336
Table 189. CIM Elements for Disk Sparing	343
Table 190. SMI Referenced Properties/Methods for CIM_StorageRedundancySet	344
Table 191. SMI Referenced Properties/Methods for CIM_IsSpare	345
Table 192. SMI Referenced Properties/Methods for CIM_Spared	345
Table 193. SMI Referenced Properties/Methods for CIM_MemberOfCollection	346
Table 194. SMI Referenced Properties/Methods for CIM_StorageExtent	346
Table 195. SMI Referenced Properties/Methods for CIM_StorageVolume	347
Table 196. SMI Referenced Properties/Methods for CIM_StoragePool	347
Table 197. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to StorageVolume)	348
Table 198. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to LogicalDisk)	348
Table 199. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to Pool)	348
Table 200. SMI Referenced Properties/Methods for CIM_LogicalDisk	349
Table 201. SMI Referenced Properties/Methods for CIM_HostedCollection	350
Table 202. SMI Referenced Properties/Methods for CIM_HostedCollection	350
Table 203. SMI Referenced Properties/Methods for SNIA_SpareConfigurationCapabilities	351
Table 204. SMI Referenced Properties/Methods for SNIA_SpareConfigurationService	352
Table 205. SMI Referenced Properties/Methods for CIM_ElementCapabilities	352
Table 206. SMI Referenced Properties/Methods for CIM_HostedCollection	353
Table 207. Erase Method	358
Table 208. CIM Elements for Erasure	362
Table 209. SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool	363
Table 210. SMI Referenced Properties/Methods for SNIA_ErasureCapabilities	363
Table 211. SMI Referenced Properties/Methods for SNIA_ErasureService	364
Table 212. SMI Referenced Properties/Methods for CIM_StoragePool	364
Table 213. SMI Referenced Properties/Methods for CIM_StorageVolume	365
Table 214. SMI Referenced Properties/Methods for CIM_LogicalDisk	365
Table 215. SMI Referenced Properties/Methods for SNIA_ErasureSetting	366
Table 216. Supported Common RAID Levels	374
Table 217. CIM Elements for Extent Composition	390
Table 218. SMI Referenced Properties/Methods for CIM_StorageExtent (Allocated Extent)	391
Table 219. SMI Referenced Properties/Methods for CIM_CompositeExtent (Composite Extent)	392
Table 220. SMI Referenced Properties/Methods for CIM_BasedOn (Volume to Composite)	393
Table 221. SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to Composite)	394
Table 222. SMI Referenced Properties/Methods for CIM_BasedOn (Non-striping Composite to Composite Extent)	394
Table 223. SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Striping Composite to Composite Extent)	395
Table 224. SMI Referenced Properties/Methods for CIM_BasedOn (Non-striping Composite to Allocated Extent)	395
Table 225. SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Striping Composite to Allocated Extent)	396
Table 226. SMI Referenced Properties/Methods for CIM_BasedOn (Allocated Extent to Composite Extent)	396
Table 227. SMI Referenced Properties/Methods for CIM_BasedOn (Allocated Extent to Allocated Extent)	397

Table 228. SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent)	397
Table 229. SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent)	398
Table 230. SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or CompositeExtent To Logical Disk) ...	398
Table 231. SMI Referenced Properties/Methods for CIM_ConcreteComponent	399
Table 232. SCSIProtocolController Property Description.....	410
Table 233. Element to Service Mapping.....	413
Table 234. Element to Element Name Mapping.....	413
Table 235. Supported Profiles for Masking and Mapping.....	414
Table 236. ExposePath Use Cases.....	415
Table 237. HidePaths Use Cases	417
Table 238. Use Cases for ExposeDefaultLUs	420
Table 239. Use Cases for HideDefaultLUs.....	422
Table 240. CIM Elements for Masking and Mapping.....	437
Table 241. SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege.....	439
Table 242. SMI Referenced Properties/Methods for CIM_AuthorizedSubject	440
Table 243. SMI Referenced Properties/Methods for CIM_AuthorizedTarget	440
Table 244. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates ControllerConfigurationService and ProtocolController).....	441
Table 245. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and StorageHardwareID).....	441
Table 246. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates PrivilegeManagementService and AuthorizedPrivilege)	442
Table 247. SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and SystemSpecificCollection)	442
Table 248. SMI Referenced Properties/Methods for CIM_ControllerConfigurationService.....	443
Table 249. SMI Referenced Properties/Methods for CIM_ElementCapabilities.....	443
Table 250. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareIDManagementService).....	444
Table 251. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ControllerConfigurationService)	444
Table 252. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareID).....	445
Table 253. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to SystemSpecificCollection)	445
Table 254. SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ProtocolController).....	445
Table 255. SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates ComputerSystem and StorageClientSettingData).....	446
Table 256. SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates StorageHardwareID and StorageClientSettingData).....	446
Table 257. SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates ProtocolController and StorageClientSettingData).....	447
Table 258. SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates Port and StorageClientSetting-Data).....	447
Table 259. SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities	447
Table 260. SMI Referenced Properties/Methods for CIM_HostedCollection	448
Table 261. SMI Referenced Properties/Methods for CIM_HostedService (Associates ComputerSystem and ControllerConfigurationService)	449
Table 262. SMI Referenced Properties/Methods for CIM_HostedService (Associates ComputerSystem and PrivilegeManagementService).....	449

Table 263. SMI Referenced Properties/Methods for CIM_HostedService (Associates ComputerSystem and StorageHardwareIDManagementService).....	449
Table 264. SMI Referenced Properties/Methods for CIM_MemberOfCollection	450
Table 265. SMI Referenced Properties/Methods for CIM_PrivilegeManagementService	450
Table 266. SMI Referenced Properties/Methods for CIM_ProtocolController	451
Table 267. SMI Referenced Properties/Methods for CIM_ProtocolControllerForUnit	451
Table 268. SMI Referenced Properties/Methods for CIM_ProtocolControllerMaskingCapabilities	452
Table 269. SMI Referenced Properties/Methods for SNIA_ProtocolControllerMaskingCapabilities	453
Table 270. SMI Referenced Properties/Methods for CIM_SAPAvailableForElement	454
Table 271. SMI Referenced Properties/Methods for CIM_StorageClientSettingData	454
Table 272. SMI Referenced Properties/Methods for CIM_StorageHardwareID	455
Table 273. SMI Referenced Properties/Methods for CIM_StorageHardwareIDManagementService	455
Table 274. SMI Referenced Properties/Methods for CIM_SystemSpecificCollection	456
Table 275. Pool Management Policy Subprofiles	460
Table 276. Static Policy Instance Methods	460
Table 277. Instance Methods of Dynamic Rules and Static Conditions and Actions	461
Table 278. Dynamic Policy Instance Methods	461
Table 279. Table 8: Pool Management Policy Subprofile Required Classes, Associations, Methods and Indications	463
Table 280. Table 9: Instance Creation, Deletion or Modification for Pool Management Policy Subprofile Classes	464
Table 281. Valid Flag Values	464
Table 282. Properties for MethodAction	465
Table 283. Properties for MethodActionResult	466
Table 284. Properties for PolicyActionInPolicyRule	467
Table 285. Properties for PolicyCapabilities	468
Table 286. Properties for PolicyConditionInPolicyRule	468
Table 287. Properties for PolicyRule	469
Table 288. Properties for PolicyRuleInSystem	471
Table 289. Properties for PolicySetAppliesToElement	472
Table 290. Properties for QueryCondition	472
Table 291. Properties for QueryConditionResult	474
Table 292. Pool Management Policy Subprofile Standard Dependencies	475
Table 293. Pool Management Policy Subprofile Functional Profile Requirements	475
Table 294. CIM Elements for Storage Server Asymmetry	491
Table 295. SMI Referenced Properties/Methods for SNIA_StorageServerAsymmetryCapabilities	494
Table 296. SMI Referenced Properties/Methods for CIM_ElementCapabilities (To Top-level ComputerSystem)	496
Table 297. SMI Referenced Properties/Methods for SNIA_StorageProcessorAffinity (Target Port Group)	496
Table 298. SMI Referenced Properties/Methods for SNIA_StorageProcessorAffinity (StorageResourceLoadGroup)	497
Table 299. SMI Referenced Properties/Methods for SNIA_AsymmetricAccessibility	498
Table 300. SMI Referenced Properties/Methods for CIM_MemberOfCollection (iSCSI Target Port Group)	498
Table 301. SMI Referenced Properties/Methods for CIM_MemberOfCollection (SCSI Target Port Group)	499
Table 302. SMI Referenced Properties/Methods for CIM_MemberOfCollection (SB Target Port Group)	499
Table 303. SMI Referenced Properties/Methods for CIM_MemberOfCollection (SATA Target Port Group)	500
Table 304. SMI Referenced Properties/Methods for CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Volumes)	500
Table 305. SMI Referenced Properties/Methods for CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools)	500
Table 306. SMI Referenced Properties/Methods for SNIA_StorageConfigurationService	501
Table 307. Block Service Management Rights	504
Table 308. Supported Profiles for Storage Virtualizer	516
Table 309. CIM Elements for Storage Virtualizer	518

Table 310. SMI Referenced Properties/Methods for CIM_ComputerSystem.....	519
Table 311. SMI Referenced Properties/Methods for CIM_ConcreteComponent	520
Table 312. SMI Referenced Properties/Methods for CIM_StorageExtent.....	520
Table 313. SMI Referenced Properties/Methods for CIM_LogicalPort (Target Port)	521
Table 314. SMI Referenced Properties/Methods for CIM_LogicalPort (Initiator Port).....	521
Table 315. SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageExtent)	522
Table 316. CompositionCharacteristics Property	524
Table 317. Supported Profiles for Volume Composition.....	530
Table 318. CreateOrModifyCompositeElement.....	532
Table 319. ReturnElementToElements	534
Table 320. GetAvailableElements	535
Table 321. GetCompositeElements.....	536
Table 322. CIM Elements for Volume Composition.....	542
Table 323. SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool.....	543
Table 324. SMI Referenced Properties/Methods for CIM_BasedOn (Volume Composition)	543
Table 325. SMI Referenced Properties/Methods for CIM_BasedOn (Extent Composition)	543
Table 326. SMI Referenced Properties/Methods for CIM_ConcreteComponent	544
Table 327. SMI Referenced Properties/Methods for CIM_CompositeExtent	544
Table 328. SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Volume Composition).....	545
Table 329. SMI Referenced Properties/Methods for CIM_ElementCapabilities.....	545
Table 330. SMI Referenced Properties/Methods for CIM_HostedService (Associates ComputerSystem and the ElementCompositionService)	546
Table 331. SMI Referenced Properties/Methods for CIM_StorageExtent.....	546
Table 332. SMI Referenced Properties/Methods for CIM_StoragePool.....	547
Table 333. SMI Referenced Properties/Methods for CIM_StorageVolume	547
Table 334. SMI Referenced Properties/Methods for SNIA_ElementCompositionCapabilities	548
Table 335. SMI Referenced Properties/Methods for SNIA_ElementCompositionService.....	549
Table 336. Supported Profiles for Volume Management.....	553
Table 337. CIM Elements for Volume Management.....	555
Table 338. SMI Referenced Properties/Methods for CIM_ComputerSystem.....	556
Table 339. SMI Referenced Properties/Methods for CIM_SystemDevice.....	557
Table 340. SMI Referenced Properties/Methods for CIM_HostedStoragePool	557
Table 341. SMI Referenced Properties/Methods for CIM_LogicalDisk	558
Table 342. SMI Referenced Properties/Methods for CIM_ElementSettingData.....	558
Table 343. SMI Referenced Properties/Methods for CIM_StorageSetting.....	559
Table 344. SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (LogicalDisk from Pool)	560
Table 345. SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool).....	560
Table 346. SMI Referenced Properties/Methods for CIM_StoragePool (Concrete).....	561
Table 347. SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)	561
Table 348. SMI Referenced Properties/Methods for CIM_ElementCapabilities.....	562
Table 349. SMI Referenced Properties/Methods for CIM_StorageCapabilities	562
Table 350. Properties for StorageProtectionCapabilities.....	566
Table 351. Properties for StorageProtectionSetting	567
Table 352. Values for ProtectionControlled.....	568
Table 353. Values for Access.....	568
Table 354. Values for InquiryProtection	568
Table 355. Values for DenyAsCopyTarget	569
Table 356. Values for LUNMappingConfigurable	569
Table 357. Values for ProtectExpirationSpecified	569

Table 358. Values for RemainingProtectionTime	569
Table 359. Methods of the Storage Element Protection Profile.....	577
Table 360. CIM Elements for Storage Element Protection.....	583
Table 361. SMI Referenced Properties/Methods for SNIA_StorageProtectionSetting	583
Table 362. SMI Referenced Properties/Methods for SNIA_ElementProtectionSettingData.....	584
Table 363. SMI Referenced Properties/Methods for SNIA_StorageProtectionCapabilities	585
Table 364. SMI Referenced Properties/Methods for SNIA_StorageProtectionService	585

Foreword

The Block Devices portion of the Storage Management Technical Specification contains the profiles for devices that server block storage. These devices include RAID arrays, Storage Virtualizers, host volume managers, and disk drives. The book also contains supporting profiles like the Block Services package.

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 RepresentedName of Representative

Brocade Communications SystemsJohn Crandall

EMC CorporationKamesh 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 ApplianceAlan Yoder

Sun Microsystems.....Mark Carlson

SymantecSteve 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

This Technical Specification defines an interface for the secure, extensible, and interoperable management of a distributed and heterogeneous storage system. This interface uses an object-oriented, XML-based, messaging-based protocol designed to support the specific requirements of managing devices and subsystems in this storage environment. Using this protocol, this Technical Specification describes the information available to a WBEM Client from an SMI-S compliant CIM WBEM Server.

Clause 2: Normative References

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.

2.1 Approved references

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

ISO/IEC 24775 Storage Management

2.2 References under development

Storage Management Technical Specification, Part 1 Common Architecture

Storage Management Technical Specification, Part 2 Common Profiles

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

DMTF WBEM URI Mapping Specification (DSP0207) 1.0.01 (preliminary)

2.3 Other references

DMTF DSP0214:2004 CIM Operations over HTTP

Normative References

Clause 3: Terms and definitions

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

3.1 Definitions

3.1.1 path

combination of initiator and target ports and logical unit

3.1.2 partition

collection of contiguous block on a disk or virtual disk

STABLE

Clause 4: Array Profile

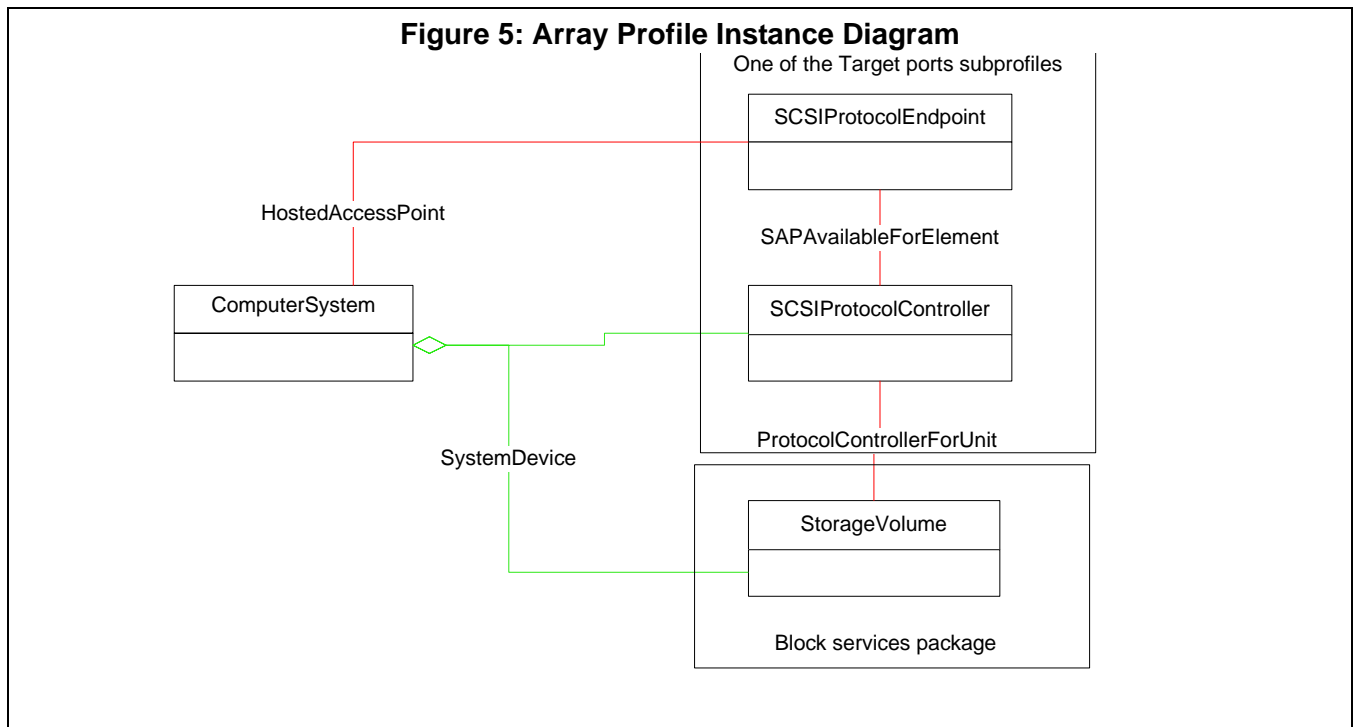
4.1 Description

The Array profile describes RAID array systems. The RAID systems supported by this profile are standalone and use local disks to store the data. Systems that use external storage or a combination of local and external storage are “Storage Virtualizers”. Systems that plug into backplanes or are on mother boards should use the “Host Hardware RAID” profile.

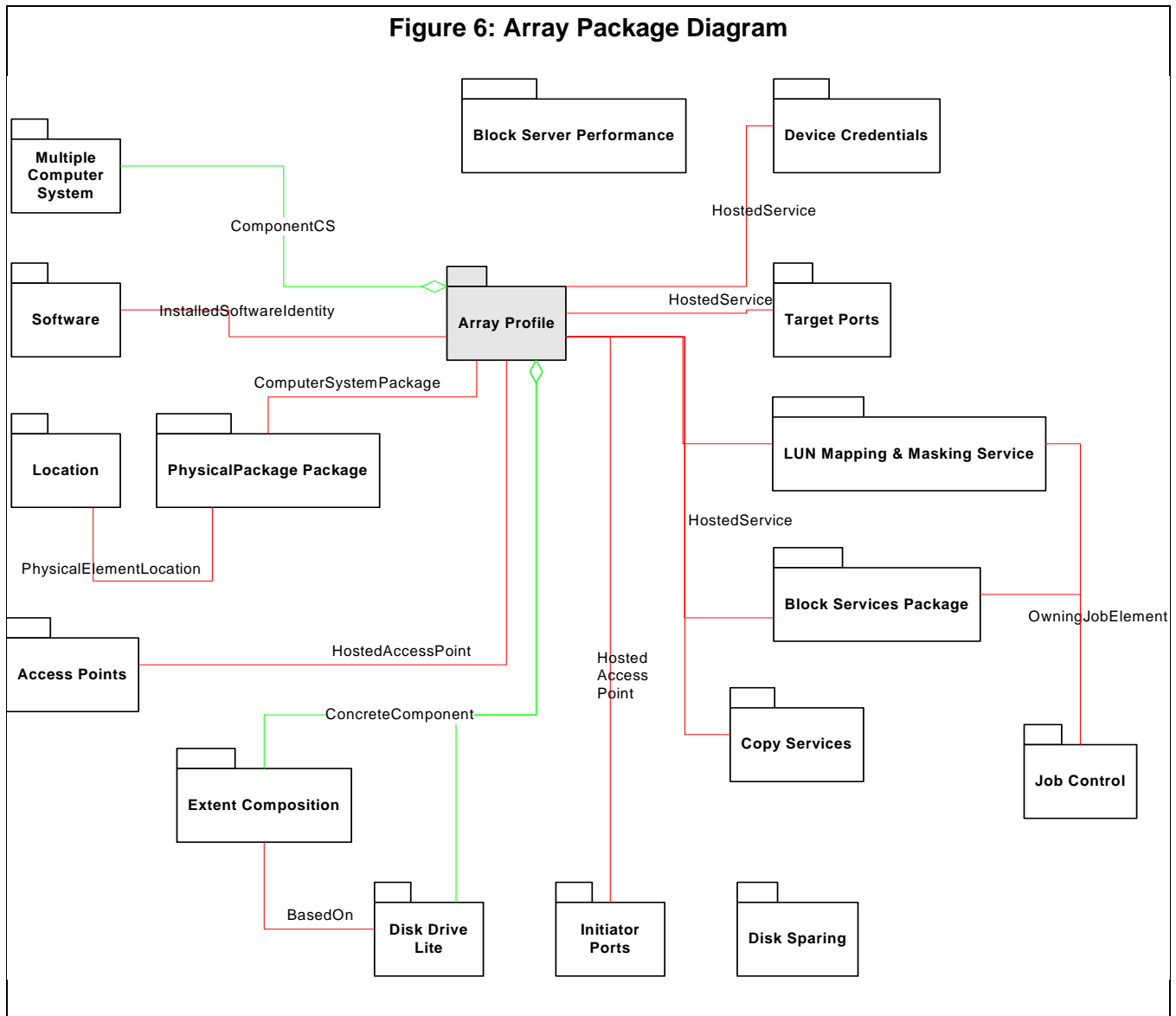
The model consists of multiple subprofiles and packages. The main sub-profiles are:

- The Array profile contains a CIM_ComputerSystems object that represents the array as a whole. It is the top level object for the profile.
- Block Services Package is the main part of the model. It contains the StorageExtents that represent the physical storage, StoragePools that gather together the extents and supports allocation and QoS (Quality of Service) settings, and StorageVolumes that represent the logical devices allocated from the pools.
- Target Ports sub-profile model the ports (e.g., Fibre Channel or iSCSI) through which the LUNs are made available to hosts.

Below is a simplified instance diagram of an array



At the minimum, the array profile provides a high level read-only ‘view’ of an array. The Block Services Package (Clause 5:) includes the basic description of how storage is managed.



The various subprofiles indicated in Figure 6 cover other areas of functionality like location, software/firmware versions, and access to the management interfaces of the array.

The base “Array” profile only contains the CIM_ComputerSystem object representing the array. This object is attached to the other sub profiles and packages through a set of associations.

The Block Services package” supports configuration of the storage using a QoS (Quality of Service) model. The model is further extended by the “Extent Composition sub-profile” to model the details of how the RAID sets are composed. This sub-profile supports the detailed configuration of storage by the selection of disk drives and partitions that make-up the RAID sets.

Target Ports model the array ports that provide block data service to the host systems. These ports shall be modeled.

The “initiator Ports” sub-profile and the “Disk Drive Lite” sub-profile are used to model the physical disk drives and how they are attached to the array system. This part of the model is optional.

The multiple computer system subprofile models multiple controllers in a single array system. The model provides a way to model failover and other redundant behavior of a multiple controller system. This sub profile is optional.

The array profile includes the “Copy Services” sub-profile to model and configure local and remote snapshots, clones, mirrors, and other array based copying. The copy services will be enhanced in the future to model remote replication. The enhancement is included as experimental in this version of SMI-S. This part of the model is optional.

Storage Management Technical Specification, Part 2 Common Profiles Clause 33: Physical Package Package describes the physical layout of the array and includes product identification information.

4.2 Health and Fault Management

Health and Fault management is described in the referenced subprofiles and packages.

4.3 Cascading Considerations

Not defined in this standard.

4.4 Supported Subprofiles and Packages

Table 1: Supported Profiles for Array

Registered Profile Names	Mandatory	Version
Access Points	No	1.2.0
Block Server Performance	No	1.2.0
Cluster	No	1.0.2
Extra Capacity Set	No	1.0.2
Disk Drive	No	1.0.2
Disk Drive Lite	No	1.2.0
Extent Mapping	No	1.0.2
Extent Composition	No	1.2.0
Location	No	1.2.0
Software	No	1.2.0
Copy Services	No	1.2.0
Pool Manipulation Capabilities and Settings	No	1.0.2
LUN Creation	No	1.0.2
Device Credentials	No	1.2.0
LUN Mapping and Masking	No	1.0.2
Masking and Mapping	No	1.2.0
Generic Target Ports	Yes	121.0

Table 1: Supported Profiles for Array

Registered Profile Names	Mandatory	Version
Backend Ports	No	1.0.2
Disk Sparing	No	1.2.0
Generic Initiator Ports	No	1.2.0
Block Services	Yes	1.2.0
Physical Package	Yes	1.2.0
Health	Yes	1.2.0
Multiple Computer System	No	1.2.0
Block Storage Views	No	1.2.0

4.5 Methods of the Profile

None.

4.6 Client Considerations and Recipes

None.

4.7 Registered Name and Version

Array version 1.2.0

4.8 CIM Elements

Table 2: CIM Elements for Array

Element Name	Requirement	Description
CIM_ComputerSystem (4.8.1)	Mandatory	'Top level' system that represents the whole array.
CIM_SystemDevice (System to SCSIProtocolController) (4.8.2)	Mandatory	This association links SCSIProtocolController to the scoping system.
CIM_SystemDevice (System to SCSIArbitraryLogicalUnit) (4.8.3)	Conditional	Conditional requirement: Elements that are mandatory if SCSIArbitraryLogicalUnit is instantiated. This association links SCSIArbitraryLogicalUnit to the scoping system.
CIM_SCSCIArbitraryLogicalUnit (4.8.4)	Optional	A SCSI Logical Unit that exists only for management of the array.
CIM_SCSCIProtocolController (4.8.5)	Mandatory	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Mandatory	Addition of a new array instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Mandatory	Deletion of an array instance

4.8.1 CIM_ComputerSystem

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 3 describes class CIM_ComputerSystem.

Table 3: SMI Referenced Properties/Methods for CIM_ComputerSystem

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
Name		Mandatory	Unique identifier for the array. Eg IP address
ElementName		Mandatory	User friendly name
OtherIdentifyingInfo	C	Mandatory	
IdentifyingDescriptions	C	Mandatory	
OperationalStatus		Mandatory	Overall status of the array

Table 3: SMI Referenced Properties/Methods for CIM_ComputerSystem

Properties	Flags	Requirement	Description & Notes
NameFormat		Mandatory	Format for Name property.
Dedicated		Mandatory	Indicates that this computer system is dedicated to operation as a storage array
PrimaryOwnerContact	M	Optional	Contact a details for owner
PrimaryOwnerName	M	Optional	Owner of the array

4.8.2 CIM_SystemDevice (System to SCSIProtocolController)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 4 describes class CIM_SystemDevice (System to SCSIProtocolController).

Table 4: SMI Referenced Properties/Methods for CIM_SystemDevice (System to SCSIProtocolController)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

4.8.3 CIM_SystemDevice (System to SCSIArbitraryLogicalUnit)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: SCSIArbitraryLogicalUnit

Table 5 describes class CIM_SystemDevice (System to SCSIArbitraryLogicalUnit).

Table 5: SMI Referenced Properties/Methods for CIM_SystemDevice (System to SCSIArbitraryLogicalUnit)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	

Table 5: SMI Referenced Properties/Methods for CIM_SystemDevice (System to SCSIArbitrary-LogicalUnit)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	

4.8.4 CIM_SCSIArbitraryLogicalUnit

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 6 describes class CIM_SCSIArbitraryLogicalUnit.

Table 6: SMI Referenced Properties/Methods for CIM_SCSIArbitraryLogicalUnit

Properties	Flags	Requirement	Description & Notes
SystemCreationClass Name		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Mandatory	User-friendly name
Name		Mandatory	
OperationalStatus		Mandatory	

4.8.5 CIM_SCSIProtocolController

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 7 describes class CIM_SCSIProtocolController.

Table 7: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Properties	Flags	Requirement	Description & Notes
SystemCreationClass Name		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	

STABLE

STABLE**Clause 5: Block Services Package****5.1 Description****5.1.1 General**

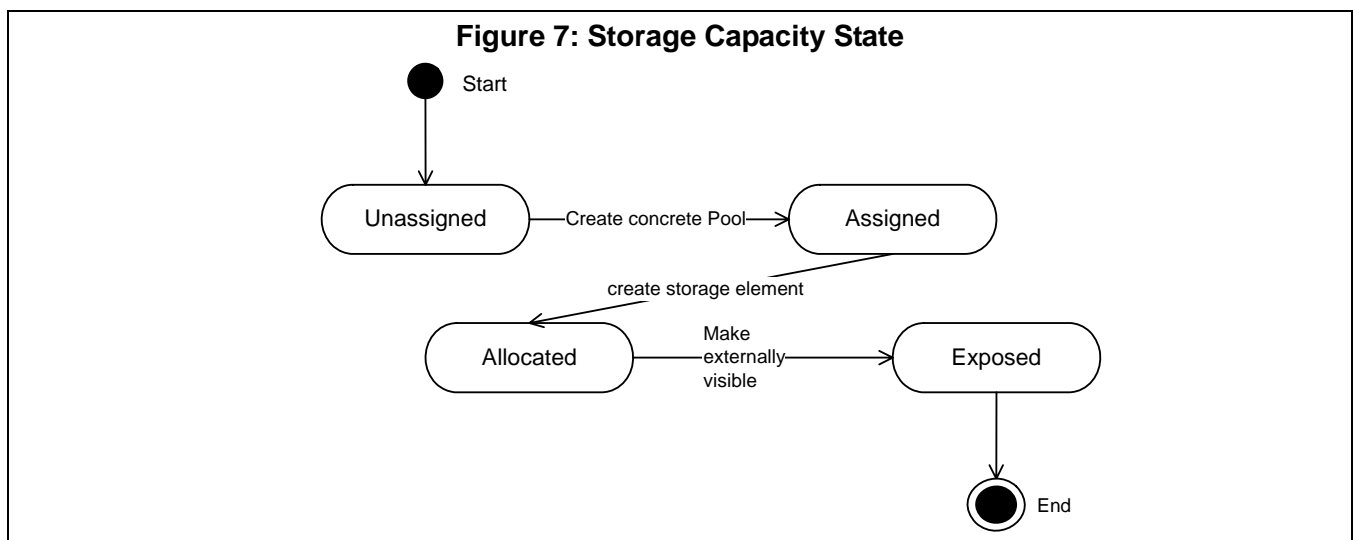
Many devices and applications provide their storage capacity to external devices and applications (block consumers) through block-based I/O. This subprofile defines a standard expression of existing storage capacity, the assignment of capacity to StoragePools, and allocation of capacity to be used by external devices or applications.

A block is:

- The unit in which data is stored and retrieved on disk and tape devices.
- A unit of application data from a single information category that is transferred within a single sequence.

5.1.2 Storage Capacity States

Figure 7 illustrates the state of a block of storage.



Each block of capacity within a storage device or application has a state. StorageVolumes and LogicalDisks, the storage elements described in this section, are distinct groupings of blocks. An unconfigured storage device or application may not have its capacity organized into concrete StoragePools. All blocks within that unconfigured device or application start in an unassigned state. Once a block is a member of a concrete StoragePool, storage capacity can be assigned. Once a block is a member of a storage element, like a StorageVolume or LogicalDisk, the storage capacity has been allocated for use by a block consumer. Once a block is visible to one or more block consumers, that capacity is exposed.

5.1.3 StoragePools**5.1.3.1 General**

A StoragePool is a storage element; its storage capacity has a given set of capabilities. Those 'StorageCapabilities' indicate the 'Quality of Service' requirements that can be applied to objects created from the StoragePool.

A StoragePool is a mandatory part of modeling disk storage systems that support the Block Services package. However, user manipulation of StoragePools is optional and may not be supported by all disk storage systems. This profile defines the support required to expose functions for creating and modifying StoragePools.

StoragePools are scoped relative to the ComputerSystem (indicated by the HostedStoragePool association). Objects created from a StoragePool have the same Computer System scope.

Child objects (e.g., StorageVolumes, LogicalDisks, or StoragePools) created from a StoragePool are linked back to the parent StoragePool using an AllocatedFromStoragePool association.

There are two properties of StoragePools that describe the size of the 'underlying' storage:

- TotalManagedStorage describes the total storage in the StoragePool.
- RemainingManagedStorage describes the storage currently remaining in the StoragePool.

The Usage property indicates if a storage pool is reserved for use by the array itself; or if the storage pool is reserved for certain operations such as "Reserved for Local Replication Services".

5.1.3.2 Primordial StoragePool

A primordial StoragePool is a type of StoragePool that contains unformatted, unprepared, or unassigned capacity. Storage capacity is drawn from the primordial StoragePool to create concrete StoragePools. A primordial StoragePool aggregates storage capacity not assigned to a concrete StoragePool. StorageVolumes and LogicalDisks are allocated from concrete StoragePools.

At least one primordial StoragePool shall always exist on the block storage system to represent the unallocated storage on the storage device. The sum of TotalManagedStorage attributes for all primordial StoragePools shall be equal to the total size of the storage of the storage system. The primordial property shall be true for primordial StoragePools.

A primordial StoragePool can be used to determine the amount of capacity on the block storage system that is not assigned to a concrete StoragePool.

5.1.3.3 Concrete StoragePool

A concrete StoragePool is a type of StoragePool. This concrete StoragePool is the only type of StoragePool created or modified by behaviors described in this package. A concrete StoragePool subdivides the storage capacity available in a block server to enable creation or modification of StorageVolumes and LogicalDisks. Concrete StoragePools can be used to assign capacity based on such factors as QoS, cost per megabyte, or ownership of storage. A concrete StoragePool may aggregate the capacity of one or many RAID groups or RAID ranks. A RAID group or rank may be created when the StorageVolume or LogicalDisk is created.

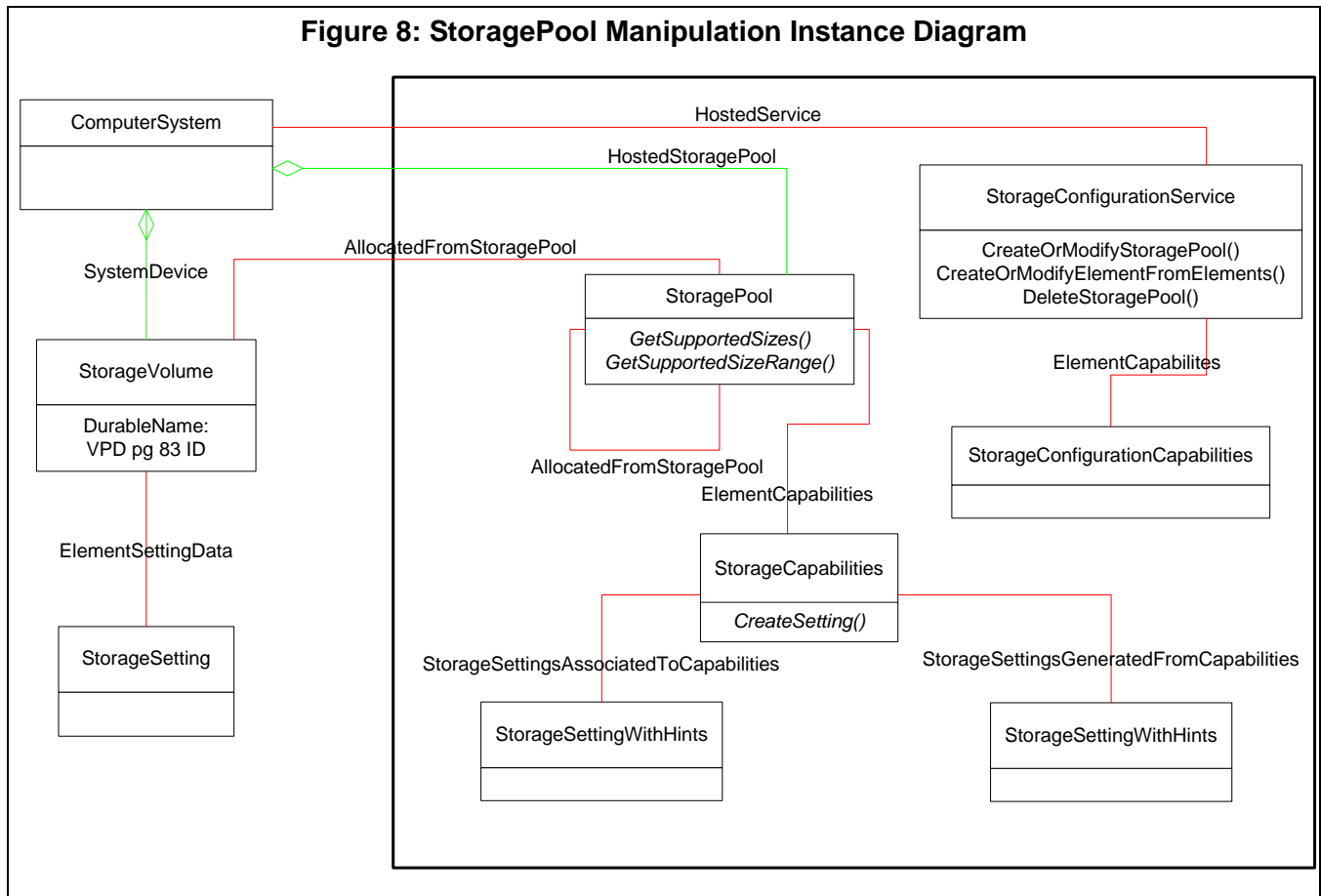
5.1.4 Blocks, Metadata, and Capacity Reported

This subprofile uses the term *metadata* to signify the capacity drawn for the creation of stripes, data copies, and similar items. The capacity removed for such constructs when creating storage elements, like StoragePools, StorageVolumes, and LogicalDisks, is reported in the difference between the capacity of the parent StoragePool and the capacity of the child storage element allocated from that parent. The TotalManagedSpace property represents the capacity that may be used to create or expand child storage elements. The RemainingManagedSpace property represents capacity left to create a new storage element or expand an existing storage element. One may use this profile to calculate capacity used for metadata.

There is likely to be a difference between a) the capacity calculated by adding up the capacity of all the disks, as reported by the manufacturers, or by adding up the LUNs consumed by a block server, as reported by the block server that exposes them, and b) the capacity that can be used to create other storage organizations or constructs from this capacity, like StoragePools, StorageVolumes, and LogicalDisks. This difference in capacity can be used for disk formatting, for example. The difference in the capacity of the primordial StoragePool and the capacity used to produce the primordial StoragePool is not reported through this subprofile.

5.1.5 StoragePool Management Instance Diagram

Figure 8 shows an instance diagram for StoragePool manipulation.



5.1.6 StoragePool, StorageVolume and LogicalDisk Manipulation

5.1.6.1 General

StorageVolumes are allocations of storage capacity that shall be exposed from a system through an external interface. In the CIM class hierarchy, they are a subclass of a StorageExtent. In SCSI terms, they are logical units.

LogicalDisks are the manifestations of the consumption of storage capacity on a general purpose computer, i.e., a host, as revealed by the operating system or a StorageVolume manager. In the CIM class hierarchy, they are also a subclass of a StorageExtent. LogicalDisks are a mandatory part of modeling host-based StorageVolume managers.

StorageVolumes and LogicalDisks are consumable storage capacity. These storage elements are the only StorageExtents available to consumers of the block service and a block device.

However, creation or modification of StorageVolumes or LogicalDisks from StoragePools is optional and may not be supported by a given disk storage system. This subprofile defines the support mandatory if the storage system exposes functions for creating StorageVolumes from StoragePools.

EXPERIMENTAL

The Usage property indicates if a volume or a logical disk is reserved for a special purpose. For example, a volume may be reserved for use by the array itself ("Reserved by the ComputerSystem"), or a volume may have been "set aside" for use by the Migration Services, in which case the usage property of the volume is set to "Reserved by Migration Services".

EXPERIMENTAL

5.1.6.2 StoragePool Manipulation Methods

The StorageConfigurationService, in conjunction with the capacity grouping concept of a StoragePool, allows SMI-S clients to configure StoragePools within block storage systems without specific knowledge about the block storage system configuration. The service has the following StoragePool manipulation methods:

- **CreateOrModifyStoragePool:** Create a StoragePool with a set of capabilities defined by the input StorageSetting, with possible sources being other StoragePool(s) or StorageExtents. Or modify a StoragePool to increase or decrease its capacity.
- **DeleteStoragePool:** Delete a StoragePool and return the freed-up storage to the underlying entities.

5.1.6.3 Storage Element Manipulation Methods

The StorageConfigurationService allows SMI-S clients to configure block storage systems with StorageVolumes (ex. LUNs) without specific knowledge about the storage system capacity. The service has the following methods for storage element manipulation:

- **CreateOrModifyElementFromStoragePool:** Create StorageVolume or LogicalDisk, possibly with a specific StorageSetting, from a source StoragePool. Also modify a StorageVolume or LogicalDisk to increase or decrease its capacity.
- **CreateOrModifyElementFromElements:** Create a StorageVolume or LogicalDisk using ComponentExtents of a parent and source StoragePool. Also alter the set of member StorageExtents of a StorageVolume or LogicalDisk or change the consumption of an existing set of member StorageExtents.
- **ReturnToStoragePool:** Return an element previously created with CreateOrModifyElementFromStoragePool to the originating StoragePool.

EXPERIMENTAL

- To locate Pools, Volumes, or Logical Disks based on their current usage, use the method `StorageConfigurationService.GetElementsBasedOnUsage`.

EXPERIMENTAL

5.1.6.4 Storage Capability Methods

The StorageCapabilities instances provide the ability to create and modify settings for use in StorageVolume creation using the following methods (part of the StorageCapabilities class):

- **CreateSetting:** Creates a setting consistent with the StorageCapabilities, may be modified before use in creating a StoragePool, StorageVolume, or LogicalDisk.
- **GetSupportedStripeLengths** and **GetSupportedStripeLengthRange:** Returns the possible stripe lengths for that capability

- **GetSupportedStripeDepths** and **GetSupportedStripeDepthRange**: Returns the possible stripe depths for that capability
- **GetSupportedParityLayouts**: Returns the possible parity layouts, rotated or non-rotated, for that capability.

See 5.5.3 for details on the associations from Setting to Capabilities.

5.1.6.5 Storage Element Size Retrieval

The **StoragePool** instances provide the ability to retrieve the possible sizes for the **StorageVolume** or **LogicalDisk** creation or modification given a **StorageSetting** as a goal:

- **GetSupportedSizes**: Returns a list of discrete sizes, given a goal. Also can return the discontinuous capacity in the **StoragePool** not yet assigned to a concrete **StoragePool** or allocated to a storage element.
- **GetSupportedSizeRange**: Returns the range of possible sizes, given a goal.
- **GetAvailableExtents**: Returns an array of **StorageExtent** references that matches a given goal and are components of the **StoragePool** and are not already members of an existing consumable storage element, child **StoragePool**, **StorageVolume**, or **LogicalDisk**.

5.1.7 Declaring Storage Configuration Options

The **StorageConfigurationCapabilities** class associated to the **StorageConfigurationService** (SCS) defines what behavior is supported by the implementation.

Table 8 defines how the **SupportedSynchronousActions** and **SupportedAsynchronousActions** array values map to methods in the **StorageConfigurationService** class. The presence of an 'Action' from Table 8 in the **SupportedSynchronousActions** array indicates that the associated 'SCS Method' does not produce a Job by side-effect. Likewise, the presence of an 'Action' from Table 8 in the **SupportAsynchronousActions** array indicates that the associated 'SCS Method' does produce a Job by side-effect and a client may use the Job to monitor the progress of the work being done. An 'Action' may be present in both arrays; if so, the implementation may or may not produce a Job by side effect.

Table 8: Mapping: Supported Actions to Methods

Action	SCS Method
StoragePool Creation, StoragePool Modification	CreateOrModifyStoragePool
StoragePool Deletion	DeleteStoragePool
Storage Element Creation, Storage Element Modification	CreateOrModifyElementFromStoragePool,
Storage Element Return	ReturnToStoragePool
Storage Element from Element Creation, Storage Element From Element Modification	CreateOrModifyElementFromElements
Usage Modification	RequestUsageChange

The **SupportedStorageElementTypes** array declares what type of storage element may be created or modified by this implementation. Because **StoragePools** are a mandatory part of this implementation, support of the **StoragePool** methods implies support of creation or modification of storage elements of type **StoragePool**.

EXPERIMENTAL

The arrays `SupportedStorageElementUsage` and `SupportedStoragePoolUsage` express what usage values apply to the storage elements types. That is, the storage element shall have one of the stated usages.

The arrays `ClientSettableElementUsage` and `ClientSettablePoolUsage` express what usage values may be manipulated by SMI-S Clients. That is, only storage elements of the given type may have their usage change changed.

EXPERIMENTAL

The `SupportedStoragePoolFeatures` array declares what `StoragePool` behavior is supported, as shown in Table 9.

Table 9: SupportedStoragePoolFeatures Array - Behavior

Supported StoragePool Behavior	Explanation
2 "InExtents"	A <code>StoragePool</code> may be created from <code>StorageExtents</code> .
3 "Single InPools", 4 "Multiple InPools"	A <code>StoragePool</code> may be the source of capacity for <code>StoragePool</code> creation or modification, i.e., concrete <code>StoragePools</code> may be created from other <code>StoragePools</code> .
5 "StoragePool QoS Change"	A new setting may be used to modify the quality of service of a <code>StoragePool</code> .
6 "StoragePool Capacity Expansion"	A <code>StoragePool</code> may be expanded
7 "StoragePool Capacity Reduction"	A <code>StoragePool</code> may be shrunk. This operation may be destructive

The `SupportedStorageElementFeatures` array declares which special features the configuration methods support, shown in Table .

Table 10: SupportedStoragePoolFeatures Array - Special Features

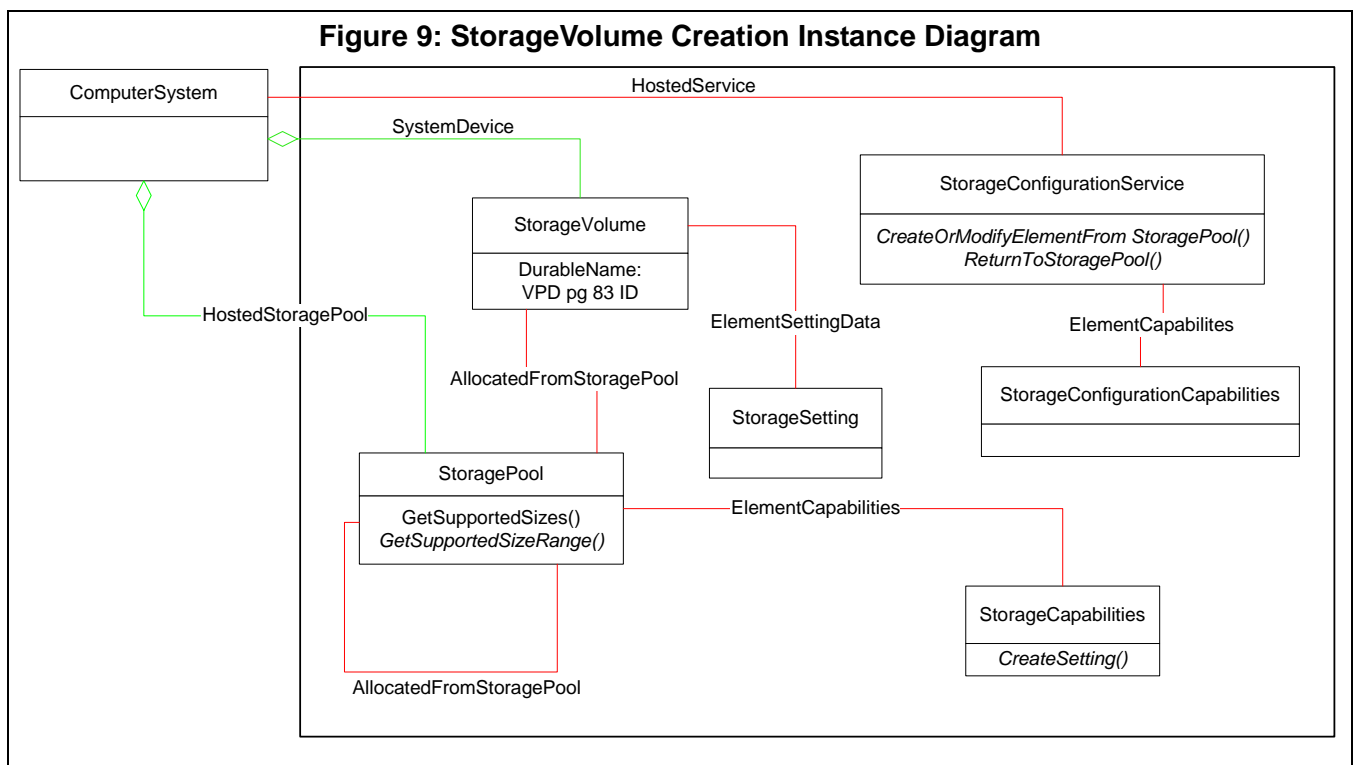
Supported Special Features	Explanation
3 "StorageVolume Creation", 5 "StorageVolume Modification"	The SMI-S implementation can create or modify <code>StorageVolumes</code> respectively.
8 "LogicalDisk Creation", 9 "LogicalDisk Modification"	The SMI-S implementation can create or modify <code>LogicalDisks</code> respectively.
6 "Single InPool", 7 "Multiple InPools"	If a SMI-S implementation supports the creation or modification of storage elements, then the implementation shall support this creation or modification of concrete <code>StoragePools</code> from either a single <code>StoragePool</code> only or from multiple input <code>StoragePools</code> .

Supported Special Features	Explanation
11 "Storage Element QoS Change", 12 "Storage Element Capacity Expansion", 13 "Storage Element Capacity Reduction"	The SMI-S implementation can change the quality of service, grow the capacity of a StorageVolume or LogicalDisk, and shrink the capacity of a StorageVolume or LogicalDisk respectively.

The SupportedStoragePoolFeatures array indicates which storage elements may be manipulated by SMI-S Clients and thereby which elements can be modified in the ways expressed by these features.

5.1.8 StorageVolume Creation Instance Diagram

Figure 9 shows an instance diagram from StorageVolume creation.



5.1.9 Backward Compatibility

This package is designed to be backward compatible with the "Pool Manipulation Capabilities, and Settings" Subprofile and the "LUN Creation" Subprofile from SMI-S 1.0.x. These subprofiles are deprecated. The Block Services package subsumes all the functionality from these subprofiles. However, to maintain backward compatibility, implementations of this package produce RegisteredProfile instances for these deprecated subprofiles as supporting SMI-S 1.0.3 with one exception. If the BlockServices implementation produces LogicalDisks and not StorageVolumes, then advertising support for these deprecated subprofiles is discouraged. If the implementation supports SLP and the deprecated subprofile RegisteredProfile instances are produced, then these deprecated subprofiles shall be advertised via SLP. See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 42:, "Server Profile".

EXPERIMENTAL

Since the Usage property on StoragePool, StorageVolume, or LogicalDisk did not exist in SMI-S 1.1 and prior versions of SMI-S, the Usage property may be Null. A client may try to utilize a storage element that is reserved for a restricted usage. In this case, the operation may fail because the supplied volume can not be used for this purpose or as a target for the operation.

EXPERIMENTAL

5.1.10 Capacity Management

Capacity characteristics of storage systems vary greatly in cost and performance. Storage capacity may need to be partitioned. StoragePools provide a means to aggregate this storage according to characteristics determined by the storage administrator or by the factory when the storage system is assembled.

A StoragePool is an aggregation of storage suitable for configuration and allocation or “provisioning”. A StoragePool may be preformatted into a form (such as a RAID group) that makes StorageVolume creation easier.

StoragePools can be drawn from a StoragePool; the result is indicated with the AllocatedFromStoragePool association).

A StoragePool has a set of capabilities held in the StorageCapabilities class. These capabilities reflect the configuration parameters that are possible for elements created from this StoragePool. The StorageCapabilities define, in terms common across all storage system implementation, which characteristics an administrator can expect from the storage capacity. These capabilities are expressed in ranges. The storage implementation can delineate the capabilities and define the ranges of these capabilities, as appropriate. Some implementations may require several narrowly defined capabilities, while others may be more flexible.

The capabilities expressed by the storage system can change over time. The number of primordial StoragePools can also change over time.

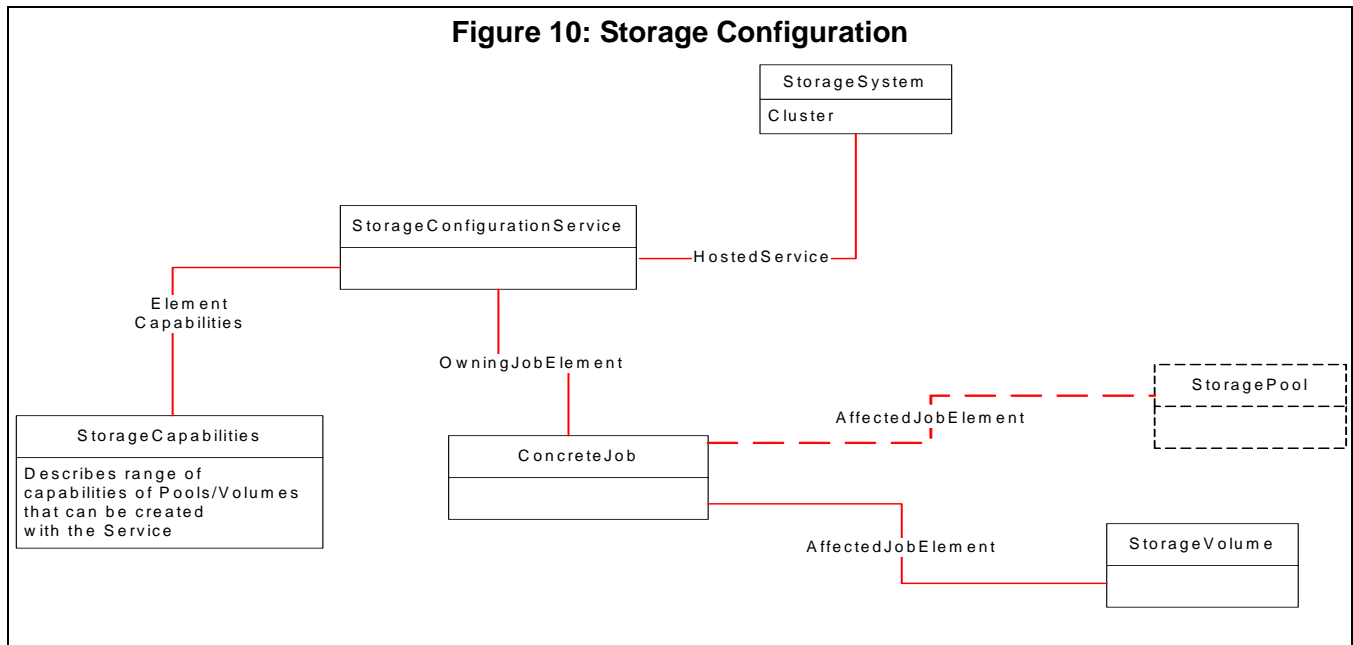
These storage capabilities are given the scope of the storage system when they are associated to the StorageConfiguratonService or the scope of a single StoragePool when associated to same. The capabilities expressed at the service scope are equal to the union of all primordial StoragePool capabilities. The capabilities can also be given the scope of a concrete StoragePool.

The storage administrator has the choice of any capability expressed by the storage system. The administrator should use this opportunity to partition the capacity. Once storage elements are drawn from the StoragePool, the administrator can be assured that the elements produced will have the capabilities previous defined.

The model allows for automation of the allocation process. An automaton can use the capabilities properties to search across subsystems for storage providing desired capabilities and then create StoragePools and/or storage elements as necessary. Inventories may be made of the capacity by capabilities.

The model also provides a means by which some common characteristics of all available storage systems can be inventoried and managed. Note that the storage system will differ in other significant ways, and these

characteristics can also be the basis for capacity pooling decisions. A sample configuration is illustrated in Figure 10.



See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30: Job Control Subprofile for details on the usage of the `StorageConfigurationJob`, `AssociatedStorageConfigurationJob`, and `OwningJobElement` associations.

The definition of storage capabilities intentionally avoids vendor specific details of `StorageVolume` configuration such as RAID types. Although RAID types imply performance and availability levels, these levels cannot be easily compared between vendor implementation—particularly in comparisons with reliability of non-RAID storage (i.e., certain virtualization appliances). There are capabilities of reliability and availability other than data redundancy. The `StorageSetting` class is provided by clients to describe the desired configuration of the allocated storage. In general, the types of parameters exposed and controlled via the `StorageCapabilities/StorageSetting` classes are:

- **NSPOF (No Single Point of Failure).** Indicates whether the `StoragePool` can support storage configured with No Single Points of Failure within the storage system. This parameter does not include the path from the system to the host.
- **Data Redundancy.** Describes the number of complete copies of data maintained. Examples include RAID 5 where one copy is maintained and mirroring where two or more copies are maintained.
- **Package Redundancy.** Describes how many physical components (packages), such as disk drives, can fail without data loss (including a spare, but not more than a single global spare). Examples include RAID5 with a Package Redundancy of 1, RAID6 with 2, RAID 6 with 2 global (to the system) spares would be 3.
- **ExtentStripeLength.** Describes the number of underlying `StorageExtents` across which data is striped in the common striping-based storage organizations. Also the number of 'members' or 'columns'. For non-striped organizations (e.g., mirror or JBOD), the `ExtentStripeLength` shall be 1.
- **UserDataStripeDepth.** Describes the number of bytes forming a stripe in common striping-based storage organizations. The stripe is defined as the size of the portion of a stripe that lies on one `StorageExtent`. `ExtentStripeLength` times `UserDataStripeDepth` yields the size of one stripe of user data.
- **ParityLayout.** Specifies whether a parity-based storage organization is using rotated or non-rotated parity.

Package Redundancy and Data Redundancy values associated to RAID levels are indicated in Table 11.

5.1.11 Mapping of RAID levels to Data Redundancy and Package Redundancy

Table 11 reflects available definitions of RAID levels.

Table 11: RAID Mapping

RAID Level	Package Redundancy	Data Redundancy	StorageExtent Stripe Length	User Data Stripe Depth	Parity Layout
JBOD	0	1	1	NULL	NULL
0 (Striping)	0	1	2 to N ¹	Vendor Dependent	NULL
1	1	2 - N	1	NULL	NULL
10	1	2 - N	2 to N	Vendor Dependent	NULL
0+1	1	2 - N	2 to N	Vendor Dependent	NULL
3 or 4	1	1	3 to N	Vendor Dependent	1
4DP	2	1	4 to N	Vendor Dependent	1
5 (3/5) ²	1	1	3 to N	Vendor Dependent	2
6, 5DP ³	2	1	3 to N	Vendor Dependent	2
15	2	2 - N	3 to N	Vendor Dependent	2
50	1	1	3 to N	Vendor Dependent	2
51	2	2 - N	3 to N	Vendor Dependent	2

1. The character 'N' represents the variable for the total number of StorageExtents.

2. '3/5' indicate RAID 5 implementations that are sometimes called RAID 5.

3. 'DP' is double parity.

It is the nature of RAID technology that even though RAID levels are the same, the storage service provided could differ, depending on the storage device implementations. Expressing the storage service level provided in end-user terms relieves the SMI-S Client and end-user from having to know what RAID Levels mean for a particular implementation. Instead, RAID level defines storage provided in storage-level terms. If a single storage device implements RAID levels that have the same package redundancy and data redundancy, the implementor should have the SMI-S Client differentiate via StorageSettingsWithHints. Additionally, the SMI-S Provider author can predefine StorageCapabilities that match best practice RAID Levels, including differentiation with StorageSettingWithHints when the StorageVolume or LogicalDisk exists. In this case, the ElementName property is

used to correlate between the capability and device documentation. Alternatively, the capability may be expressed in broader ranges for more flexible storage systems.

StorageSetting instances whose "ChangeableType" property is "0", "Fixed - Not Changeable", (identifying the StorageSettings which represent certain non-changeable sets of preset storage property data, describing "fixed", or pre-defined Settings, corresponding to preset RAID levels), the Element name should contain a string value from a comprehensive list of well-known RAID configuration names. The ElementName string value should be the name of the RAID level, from this list, which most closely describes the storage characteristics of the StorageSetting in question. This list of RAID level strings includes, but is not limited to: "JBOD", "RAID0", "RAID1", "RAID0+1", "RAID01E", "RAID10", "RAID3", "RAID4", "RAID4DP", "RAID5", "RAID3/5", "RAID5DP", "RAID6", "RAID15", "RAID50", "RAID51". In addition, the "Description" property of the pre-defined StorageSettings should (optionally) contain similar RAID level information in a more free-form text format, including vendor-specific and/or value-added annotations, for example: "RAID 3, with spares", or "RAID 5, 7D + 1P".

5.1.12 Storage Setting Associations to Storage Capabilities

A Storage Setting instance can be associated to its parent StorageCapabilities instance through either the StorageSettingsAssociatedToCapabilities or StorageSettingsGeneratedFromCapabilities association instances. The nature of the associated setting is different depending on the association instance used.

A Storage Setting associated via a StorageSettingsAssociatedToCapabilities instance shall not be modifiable by the client (ChangeableType = 0 "Fixed - Not Changeable"). These types of settings are used to define the possible configurations of StoragePools, StorageVolumes or LogicalDisks where the number of possibilities are small because the capabilities of the device itself are likewise limited. When an instance of a Capabilities class is created as a side effect of creating a concrete StoragePool, this type of Storage Setting may also be created or an existing Storage Setting associated to this new Capabilities instance as well. A client can use the StorageSettingsAssociatedToCapabilities association to find the default goal for the Capabilities instance, using the DefaultSetting property. There shall be one default per combination of a StoragePool instance, associated StorageCapabilities instances, and associated StorageSetting instances.

A Storage Setting associated via a StorageSettingsGeneratedFromCapabilities instance may be modified by a client (ChangeableType = 1 "Changeable - Transient" or Changeable = 2 "Changeable - Persistent"). When a Setting is created from a Capabilities instance, it is transient (e.g., ChangeableType = 1), i.e., the Setting instance may not remain for long. This Setting may be removed from the CIMOM after reboots or after a set period of time. The client should create and use the Setting as soon as possible. Alternatively, some implementations will allow the client to request that the Setting be retained. This request is made by changing the ChangeableSettingType to 3 "Changeable - Persistent". SMI-S does not define normative behavior for the changing of the ChangeableType property.

EXPERIMENTAL

5.1.13 The Usage Property

The intended usage of storage elements and storage pools is specified in the Usage property of these components. For the most part, the usage of these components is 2 "Unrestricted". However, a system manager and/or a client may decide that certain storage elements are to be set aside for a specific application. For example, a number of volumes are created for the sole purpose of being used for Migration Services. In this case, the volumes are created using a storage setting with the StorageElementInitialUsage of "Reserved by Migration Services". Alternatively, a client may request an "Unrestricted" volume to be converted to "Reserved by Migration Services" by invoking the method StorageConfigurationService.RequestUsageChange. The Provider shall honor the request if the client has access to the storage element and the requested change is valid. The property ClientSettableUsage indicates what usage values are valid for a given component.

The companion property OtherUsageDescription may be used to indicate a component's usage that is not covered by the usage value map. The Usage property value in this case shall be set to 1 "Other".

The Usage and OtherUsageDescription properties are maintained by the Provider. Restricted values may already exist for static elements that pre-exist when the Provider is discovered.

The Usage and OtherUsageDescription property values may change as a side effect of other method calls, e.g. a StorageVolume that may have been a replica target candidate at one time, may no longer be a replica target candidate once it is active as a replica target.

Storage elements that support the Usage property will also have a property called ClientSettableUsage. This property indicates which usage values may be manipulated by a client using the method StorageConfigurationService.RequestUsageChange.

Using the method StorageConfigurationService.GetElementsBasedOnUsage, clients are able to retrieve storage elements and storage pools based on their current usage values. For example, a client can retrieve all the volumes that are candidate to be used as a Local Replica Target. Using the same method StorageConfigurationService.GetElementsBasedOnUsage with the criteria parameter set to 2 "Available Only", clients are able to retrieve the available (i.e. not in use) storage elements and storage pools based on their current usage value.

Some methods change the usage of a storage element. For example, a client supplies a volume to be used as a target in the call to the method CreateReplica.

The following table describes some of the representative values of the Usage property (storage element refers to a StorageVolume, LogicalDisk, or StoragePool):

Table 12: Meaning of Usage values

Usage Value	Description
Reserved by the ComputerSystem	The storage element is used by the array itself for firm-ware, storage processor software, etc.
Reserved for Local Replication Services	The storage element is designated for activities related to the CopyServices. For example, SNAP cache.
Local Replica Target	The storage element is suitable to be used as replica target.
Element Component	The StorageVolume or LogicalDisk is now acting as a StorageExtent. In this case, the storage element no longer appears in the list of these element types. Use the method GetElementsBasedOnUsage to locate such storage elements.

EXPERIMENTAL

5.1.14 Read-Only Model Requirements

This package defines classes and behavior to express the assignment and allocation of storage capacity and the mechanism for configuring the storage capacity. The expression of the assignment and allocation of storage capacity through the StoragePool, StorageVolume, StorageExtent, LogicalDisk and related associations is mandatory. An implementation should offer the configuration of one or more classes of storage elements. The expression of the support for the configuration of storage is through the support of the StorageConfigurationService. If an instance of this class is not provided, then a client can assume that no configuration operations are supported. A implementation shall not provide an instance of the StorageConfigurationService if none of the extrinsic methods of the service are supported.

If the implementation is only supporting read-only information about the capacity assignment and allocation but does not offer modification of the capacity configuration, then that implementation is said to be a *read-only* implementation. In such a model, only classes listed in Table 13 are required. Classes not explicitly listed are not required for *read-only* implementations.

Table 13: Classes Required In *Read-Only* Implementation

Required Classes	Reason for Requirement
StoragePool, StorageVolume and/or LogicalDisk, HostedStoragePool and AllocatedFromStoragePool	Reporting of unassigned, assigned, and allocated capacity
StorageCapabilities and ElementCapabilities	Reporting of block server capabilities
StorageSetting and ElementSettingData used is associated to StorageVolume and LogicalDisk	Reporting of the capabilities of existing StorageVolumes and LogicalDisks

5.1.15 StorageExtent Conservation

5.1.15.1 General

StorageExtent Conservation is the construct where the remaining capacity after the partial use of a StorageExtent is itself represented as a StorageExtent, based on the antecedent StorageExtent. Note that the StorageExtent class itself does not report the amount of capacity that is used by another StorageExtent that draws capacity from it. In order to calculate the remaining space from a StorageExtent model without StorageExtent Conservation, the client would have to calculate the existence of remaining capacity through finding unused ranges of blocks as expressed by the StorageExtent's BasedOn associations.

This notion allows a client to use those remaining StorageExtents to determine the physical components like disk drives and network ports that are associated to this remaining space in order to pick the StorageExtent best suiting its needs for, for example, storage network redundancy or performance history.

5.1.15.2 Requirements for the General use of StorageExtents

The general use of StorageExtents, which is optional for this subprofile, is subject to the following requirements:

- Allocating capacity from a StoragePool shall not reduce the total size of the StoragePool.
- A given StorageExtent instance shall not be a component of more than one StoragePool. However, an given block may be accounted for in the range of blocks represented by more than one StorageExtent instance. In other words, a given block may be associated to more than one StoragePool.
- The use of all or some of the capacity of an StorageExtent directly, by passing the reference to the StorageExtent in a method call, or indirectly, by passing the size of the desired storage element, shall result in the creation of new StorageExtents that are components of the new StorageVolume or LogicalDisk.
- Any remaining capacity from the StorageExtent shall be represented by a new ComponentExtent of the source StoragePool that is based on the partitioned StorageExtent. This StorageExtent is called a *remaining StorageExtent*.
 - 1) If the Size requested is smaller than the total consumable size of the StorageExtents or StoragePools, then these resources are partially used. In this case, the model shall reflect what capacity was used and what capacity remains of the StorageExtents or StoragePools passed as arguments to CreateOrModifyStoragePool and CreateOrModifyElementFromElements methods.

- 2) Once the capacity represented by a remaining StorageExtent is used to assign or allocate capacity, the remaining StorageExtent either shrinks in size or is removed from the model. A remaining StorageExtent shall not be molded to have other StorageExtents based on it.
- A StorageExtent that was split or partially used may be made whole by the deletion of the storage element whose creation or modification gave rise to the partial use of the StorageExtent in the first place.

Figure 11, Figure 12, and Figure 13 illustrate the use of StorageExtents to represent the partitioning of a StorageExtent's capacity. An implementation of this subprofile may also implement Clause 15: Extent Composition Subprofile. Extent Conservation requires the instantiation of additional ComponentExtents that represent remaining space. These ComponentExtents are in addition to those modeled by the Extent Composition Subprofile. Available StorageExtents, including remaining space StorageExtents, which meet specific goal requirements, are found using the GetAvailableExtents method of the StoragePool.

The modeling of remaining StorageExtents is not within the scope of the Extent Composition Subprofile. However, the recipes written for Clause 15: Extent Composition Subprofile will tolerate these additional extents. The modeling of free/unused extents is defined only in the 5.1.15 StorageExtent Conservation.

Support of the GetAvailableExtents and CreateOrModifyElementFromElements methods are not required by the Block Services package nor Clause 15: Extent Composition Subprofile. An implementation may support the representation of StorageVolume or LogicalDisk structure through Clause 15: Extent Composition Subprofile but not support these methods.

If an implementation supports the GetAvailableExtents and CreateOrModifyElementFromElements methods and the Block Services Package, then it shall also implement Clause 15: Extent Composition Subprofile. See 5.5.3. Additionally, the implementation shall implement either both methods (if it implements one of the methods) or neither method.

The most virtualized Storage Extents are those that have no dependent storage extents that are either StorageVolumes or LogicalDisks. There are three associations that may represent the most virtualized storage components of a StoragePool:

- ConcreteComponent
- AssociatedComponentExtent
- AssociatedRemainingExtent.

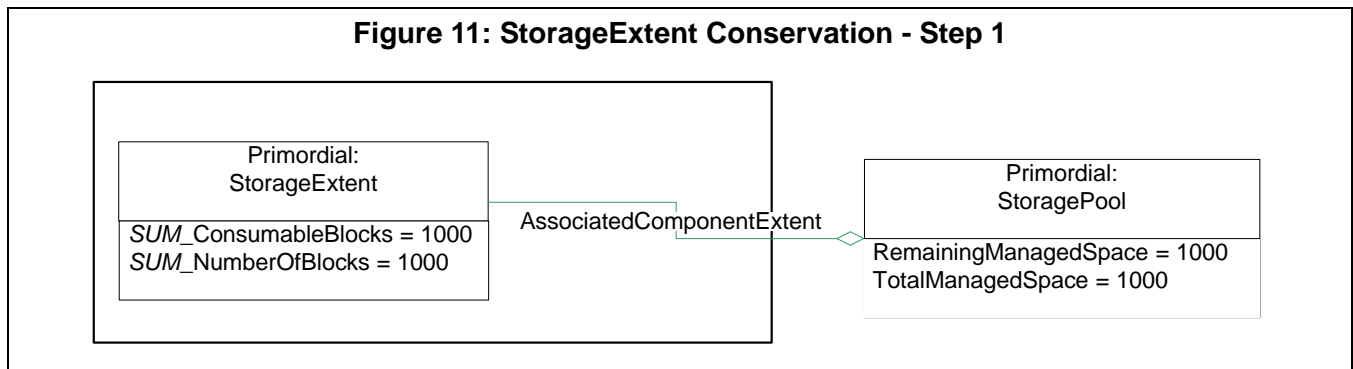
If there are StorageExtents associated to a StoragePool via ConcreteComponent, these StorageExtents shall also be associated to the same StoragePool via AssociatedComponentExtent or AssociatedRemainingExtent. The set of instances associated to this StoragePool via ConcreteComponent shall equal the union of the sets of StorageExtents associated to the same StoragePool via AssociatedComponentExtent and AssociatedRemainingExtent. The subset of AssociatedRemainingExtent StorageExtents represents remaining capacity, as defined in preceding paragraphs. These StorageExtents are remaining StorageExtents. The subset of AssociatedComponentExtent StorageExtents represents capacity that has not yet been allocated, is allocated in part, or is allocated in its entirety.

5.1.15.3 The Three Steps of StorageExtent Conservation

Figure 11, Figure 12, and Figure 13 show how StorageExtents are partitioned to represent the partial usage of the capacity in the construction of a concrete StoragePool and a concrete StorageVolume. For the purposes of illustration all the numbers in the figures are expressed in blocks even though some of the class properties are in blocks and others are in bytes. The solid line box around the elements in the diagram groups those classes that are defined in Clause 15: Extent Composition Subprofile.

The initial state in Figure 11 starts with a primordial StoragePool that is realized by a primordial StorageExtent. This StorageExtent is part of the initial capacity of the device or added to the device in a process defined outside of this subprofile. The process of assigning capacity to a StoragePool and allocating capacity to a StorageVolume or LogicalDisk is defined inside of this subprofile. To simplify the diagram, the StoragePool has only one

ComponentExtent box that represents many StorageExtents. The “SUM_” prefix indicates that the size of the StorageExtents are a summation. Both the StoragePool and StorageExtent start with 1000 blocks of storage capacity.



A concrete StoragePool is drawn from the primordial StoragePool in step 2, shown in Figure 12. Figure 12 groups the instances modeled using Clause 15: Extent Composition Subprofile with a dark box. The concrete StoragePool takes only half the capacity of the parent StoragePool. In this particular example, the metadata required by the implementation is written to the storage after this step. Another StorageExtent is created to represent the remaining capacity of the primordial StoragePool that was not used in the creation of the concrete StoragePool. ConsumableBlocks remain constant after the creation of the StorageExtent as a representation of the space actually available for use is other storage elements that are based on the StorageExtent. The remaining space StorageExtent can be used for the creation of other StorageVolumes or Logical Devices. If GetAvailableExtents were called on the primordial StoragePool at this point, a reference to the remaining StorageExtent shall be returned. A reference to the original primordial StorageExtent shall not be returned because the StorageExtent is entirely allocated.

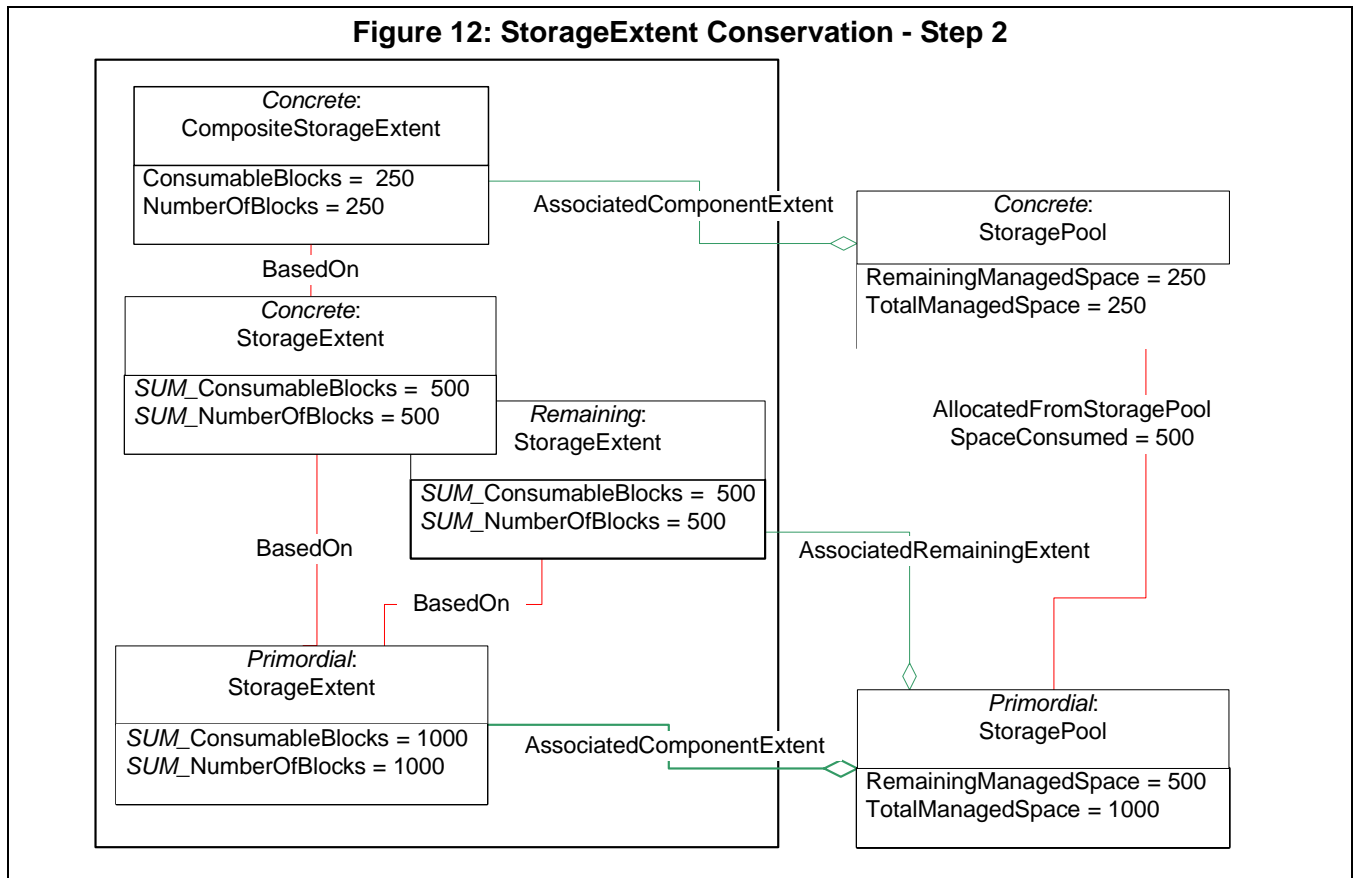
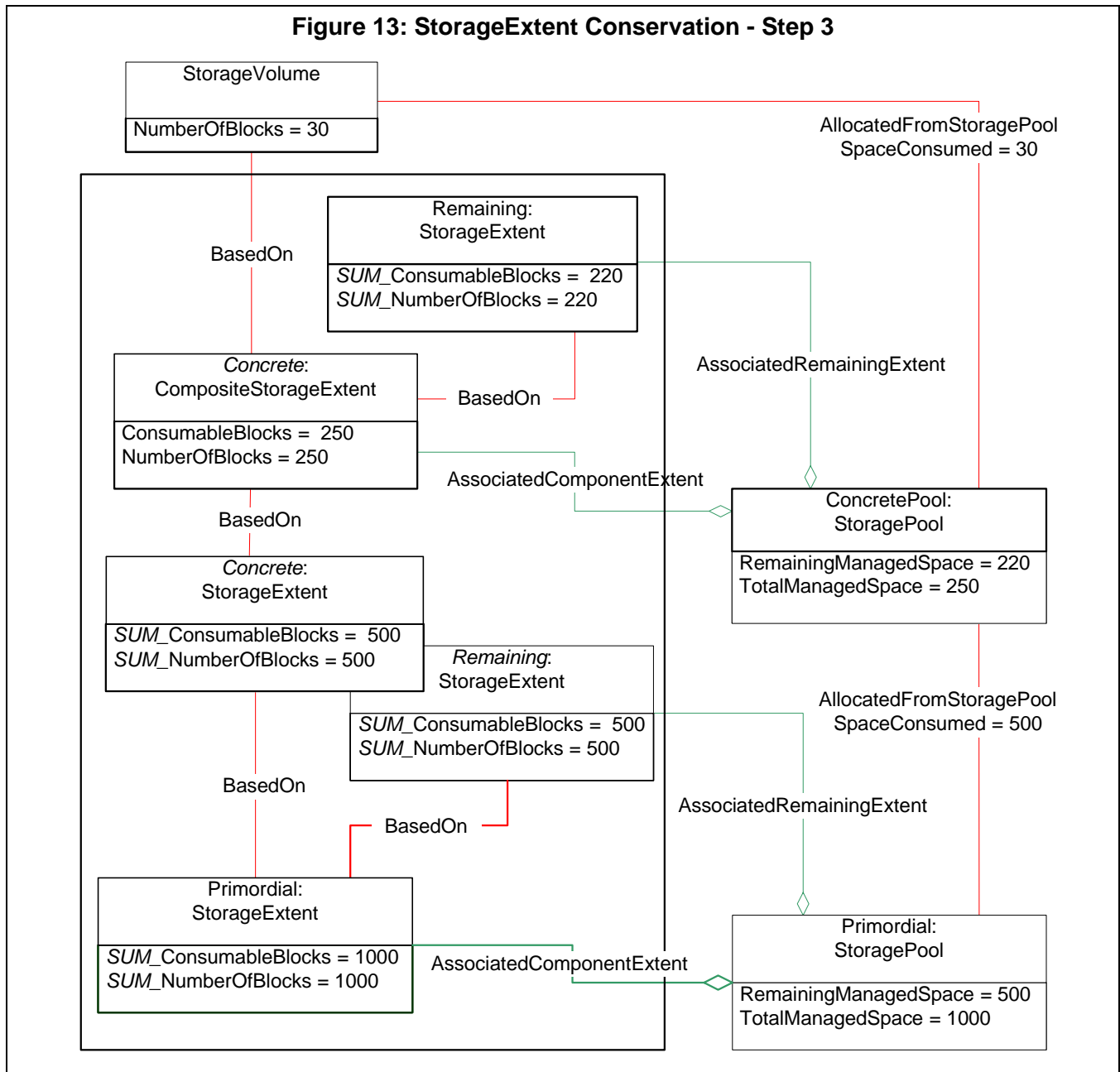
Figure 12: StorageExtent Conservation - Step 2

Figure 13 shows a StorageVolume creation. Figure 13 groups the instances modeled using Clause 15: Extent Composition Subprofile with a dark box. This particular implementation draws storage capacity for metadata (for its own house-keeping) during the creation of the StorageVolume. Not shown is the case where the metadata is drawn from capacity during the creation of the concrete StoragePool. A RAID 1 stripe is written over three StorageExtents. These StorageExtents are likely to be disk drives. Again, a remaining StorageExtent is created to represent the capacity of the parent concrete StoragePool that is not used in the creation of the StorageVolume. A call to the concrete StoragePool's GetAvailableExtents method yields a reference to the remaining StorageExtent.

Figure 13: StorageExtent Conservation - Step 3

In all cases, the TotalManagedSpace and RemainingSpace attributes reflect the total capacity and the capacity that can be drawn from a StoragePool, respectively. In this figure, the metadata is drawn from the capacity in the creation of the storage element.

- The capacity drawn by the metadata from the parent StoragePool is reflected by the sum of associated AllocatedFromStoragePool.SpaceConsumed minus the StoragePool.TotalManagedSpace of the child StoragePool.
- The capacity drawn by the metadata from each StorageVolume or LogicalDisk is reflected by SpaceConsumed minus NumberOfBlocks times BlockSize.

5.1.16 Formulas For Calculating Capacity

These formulas define calculations that shall be valid in a conformant implementation:

- RemainingManagedSpace plus AllocatedFromStoragePool.SpaceConsumed from all of the StorageVolumes, LogicalDisks, and StoragePools allocated from the StoragePool shall equal TotalManagedSpace.
- A parent StoragePool's TotalManagedSpace equals RemainingManagedSpace plus the sum of all related AllocatedFromStoragePool SpaceConsumed.
- If Clause 15: Extent Composition Subprofile is implemented:
 - 1) The StoragePool's TotalManagedSpaceshall be equal to the sum of all the AssociatedComponentExtent StorageExtent's BlockSize times ConsumableBlocks.
 - 2) Using the BasedOn association from the StoragePool's component StorageExtents (found using ConcreteComponent or AssociatedComponentExtent, or AssociatedRemainingExtent), the sum of the Dependent StorageExtent's NumberOfBlocks shall be equal to the ConsumableBlocks of the Antecedent StorageExtent.
 - 3) The StoragePool's RemainingManagedSpace shall be equal to the sum of BlockSize times Consumable-Blocks for the union of the following sets of StorageExtents:
 - a) The set of StorageExtents associated to the StoragePool via AssociatedComponentExtent where each StorageExtent does not participate in an Antecedent relationship via one or more BasedOn associated with either a StorageVolume or a LogicalDisk.
 - b) The set of StorageExtents associated to the StoragePool via AssociatedRemainingExtent.

5.1.17 Storage Element Manipulation

The StorageConfigurationService class contains methods to allow creation, modification and deletion of StorageVolumes or LogicalDisks. The capabilities of a StorageConfigurationService or StoragePool to provide storage are indicated using the StorageCapabilities class. This class allows the Service or StoragePool to advertise its capabilities (using implementation independent attributes) and a client to set the attributes it desires.

The concept of "hints" is included. Hints allow a client to provide general requirements to the system as to how it expects to use the storage. Hints allow a client to provide extra information to "tune" a StorageVolume or LogicalDisk. If a client chooses to supply these hints when creating a StorageVolume or LogicalDisk, the StorageSystem can either use the hints to determine a matching configuration or ignore them.

When creating a StorageVolume or LogicalDisk, a reference to an instance of StorageSetting is passed as a parameter to the StorageConfigurationService.CreateOrModifyElementFromStoragePool or CreateOrModifyElementFromElements methods. This reference provides a goal for that element.

The current 'service level' being achieved is reported via the StorageVolume or LogicalDisk class itself. For example, data redundancy reported in the Setting associated to the storage element may be different from the data redundancy reported in the storage element itself. This difference indicates that a copy of the data is no longer available.

StorageVolumes or LogicalDisks are created from StoragePools via a StorageConfigurationService's CreateOrModifyElementFromStoragePool() method. A StorageVolume creation operation takes time, and a Client needs to be aware that the operation is not complete until the StorageVoume.OperationalStatus is OK. A Client may also monitor the progress of the operation using the ConcreteJob class and its properties.

The name of a StorageVolume, LogicalDisk, or StoragePool can be changed. The existence of the EnabledLogicalElementCapabilities instance associated to the storage element indicates that the storage element can be named. If ElementNameEditSupported is set to TRUE, then the ElementName of the associated storage

element name can be modified. The `ElementNameMask` property provides the regular expression that indicates the name limits; see Table 16, “CIM Elements for Block Services” for details for this property.

This model does not help in communicating whether or not the element name can be provided in the creation or the modification of the storage element through these `StorageConfigurationService` methods (if there are no existing storage elements of a given type):

- `CreateOrModifyStoragePool()`
- `CreateOrModifyElementFromStoragePool()`
- `CreateOrModifyElementFromElements()`

First, there shall be a single `EnabledLogicalElementCapabilities` for each storage element type.

Note that the `ElementType` parameter of these methods requests the element to be created or modified. There shall be a single mask for each storage element type. Each of these instances shall be associated to the `StorageConfigurationService` via the `ElementCapabilities` association. Each of these `EnabledLogicalElementCapabilities` instances may also be used to express the capabilities of storage elements. The `ElementNames` of these `EnabledLogicalElementCapabilities` instances that define the possibility of naming `StoragePools`, `StorageVolumes`, and `LogicalDisks` type shall be of the values of "StoragePool Enabled Capabilities", "StorageVolume Enabled Capabilities", and "LogicalDisk Enabled Capabilities" respectively. If the implementation supports the creation or modification of a given element type and the modification of the name of the storage element, then it shall produce the aforementioned `EnabledLogicalElementCapabilities` instances.

If a storage element's name is modifiable through one of the aforementioned `StorageConfigurationService` methods, it shall also be modifiable through instance modification. However, a storage element's name may be modifiable through instance modification, but may not be modifiable through these service methods.

EXPERIMENTAL

By default, storage elements are created with the 2 "Unrestricted" value for their `Usage` property. To specify a different value for the `Usage` property, set the appropriate `StorageExtentInitialUsage` or `StoragePoolInitialUsage` of the applicable `StorageSetting` before creating the storage element. Subsequently, the `Usage` property can be modified by calling the `StorageConfigurationService.RequestUsageChange` method.

EXPERIMENTAL

5.2 Health and Fault Management Considerations

The extrinsic methods should produce `Errors` (instances of `CIM_Error`) instead of some of the failure return codes. CIM errors include parameter errors, hardware errors, and time-out errors. See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 28: Health Package for details.

EXPERIMENTAL

The standard messages specific to this profile are listed in Table 14. See *Storage Management Technical Specification, Part 1 Common Architecture* Clause 9: Standard Messages for a list and description all standard messages.

Table 14: Standard Messages for Block Services Package

Message ID	Message Name
MP17	Invalid property combination during instance creation or modification
DRM19	Stolen capacity
DRM20	Invalid extent passed
DRM21	Invalid deletion attempted

EXPERIMENTAL

5.3 Cascading Considerations

Not defined in this standard.

5.4 Supported Profile, Subprofiles and Packages

Table 15: Supported Profiles for Block Services

Registered Profile Names	Mandatory	Version
Job Control	No	1.2.0
Extent Composition	No	1.2.0

5.5 Methods of this Profile

5.5.1 Extrinsic Methods on StorageCapabilities

5.5.1.1 CreateSetting

CreateSetting is a method in StorageCapabilities and is invoked in the context of a specific StorageCapabilities instance.

```
uint32 CreateSetting(
    [In] uint16 SettingType,
    [Out] CIM_StorageSetting REF NewSetting)
```

This method on the `StorageCapabilities` class is used to create a `StorageSetting` using the `StorageCapabilities` as a template. The purpose of this method is to create a `StorageSetting` that is associated directly with the `StorageCapabilities` on which this method is invoked and has properties set in line with those `StorageCapabilities`. The contract defined by the `StorageCapabilities` shall constrain the `StorageSetting` used as the Goal.

The `StorageCapabilities` associated with the `StoragePool` define what types of storage can be allocated. The client shall determine what subset of the parent `StoragePool` capabilities to use, albeit a primordial `StoragePool` or a concrete `StoragePool`. The `StorageSetting` provided to the `StoragePool` creation method defines what measure of capabilities are desired for the following storage allocation. First, the client retrieves a `StorageSetting` or creates and optionally modifies an existing `StorageSetting`. If no satisfactory `StorageSetting` exists, then the client uses this method to create a `StorageSetting`.

The client has the option to have a `StorageSetting` generated with the default capabilities from the `StorageCapabilities`. If a '2' ("Default") is passed for the Setting Type parameter, the Max, Goal, and Min setting attributes are set to the default values of the parent `StorageCapabilities`. Otherwise, with using '3' ("Goal"), the new `StorageSetting` attributes are set to the related attributes of the parent `StorageCapabilities`, e.g., Min to Min and Max to Max. If the `StorageSetting` requested already exists, associated to the `StorageCapabilities`, then the method returns this existing `StorageSetting`. This type of `StorageSetting`, newly created or already existing, is associated to the `StorageCapabilities` via the `GeneratedStorageSetting` association.

Only a `StorageSetting` created in this manner may be modified or deleted by the client. The client uses the `NewSetting` parameter to set the new `StorageSetting` to the values desired (using `ModifyInstance` or `SetProperties` intrinsic methods).

The implementation shall not generate a Setting whose values fall outside of the range of the parent Capabilities.

The `StorageSetting` cannot be used to create storage that is more capable than the parent `StorageCapabilities`. The `ModifyInstance` and `SetProperties` CIM Operations shall fail when the Setting has a Max value greater (or a Min value less) than the parent `StorageCapabilities`.

If the storage device supports hints, then the new `StorageSetting` contains the default hint values for the parent `StorageCapabilities`. The client can use these values as a starting point for hint modification (using intrinsic methods).

`StorageSetting` instances associated with `StorageVolume` or `LogicalDisk` shall not be modified or deleted directly.

Once this type of `StorageSetting` is used as the Goal for the creation or modification of a `StoragePool`, the Goal setting properties are copied into a new `StorageCapabilities` instance. The new `StorageCapabilities` instance is associated to the newly created or modified `StoragePool`. If the `StoragePool` was modified, then the previous `StorageCapabilities` shall be removed. The new `StorageCapabilities` instance, associated with the new `StoragePool`, should describe the parameters used in its creation or modification.

Once this type of `StorageSetting` is used as the Goal for the creation or modification of a `StorageVolume` or `LogicalDisk`, the Goal `StorageSetting` shall be duplicated, with the exception of the instance keys. The duplicate Setting is associated to the newly created or modified `StoragePool`, `StorageVolume`, or `LogicalDisk`. The generated Setting may be removed thereafter. The new `StorageSetting` instance, associated with the new storage element, should describe the parameters used in its creation or modification.

The following set of methods (5.5.1.2, 5.5.1.3, and 5.5.1.4) can be implemented to allow a client to be more specific about the configuration of the stripe length, stripe depth, and parity in a Setting. Thereby the client can get specific RAID levels or quality of service characteristics.

The stripe length, stripe depth, and party extrinsic methods may be supported. These methods may be supported in the content of one capabilities and not in another within the same implementation. Sometimes the block striping is done as part of the creation of the concrete `StoragePool`, and sometimes the block striping is done as part of the creation of a `StorageVolume` or `LogicalDisk`. There may be implementations that allow striping to be done in both steps.

A client may use `StorageSettingHints` to imply desired striping (or other) characteristics are desired. The striping and parity methods and properties may be used in combination with hints. The hints express a ranking of preference. While the striping and parity methods and properties are much more explicit. When the hints and the stripe and parity Settings properties are used in combination, the striping and parity properties of the Setting are also considered hints, and the implementation may still create or modify the `StoragePool` or storage element using its best effort.

This specification does not define how the ranking of hints relates to the exact nature of the `StoragePool` or storage element created or the nature of their modification.

5.5.1.2 Getting Stripe Length

```
uint32 GetSupportedStripeLengths(
    [Out] uint16 StripeLengths[])
```

This method is used to report discrete `ExtentStripeLengths` for `StorageVolume`, `LogicalDisk`, or `StoragePool` creation. Some systems may support only discrete stripe lengths.

```
uint32 GetSupportedStripeLengthRange(
    [Out] uint16 MinimumStripeLength,
    [Out] uint16 MaximumStripeLength,
    [Out] uint32 StripeLengthDivisor)
```

This method is used to report a range of possible `ExtentStripeLengths` for `StorageVolume`, `LogicalDisk`, or `StoragePool` creation. Some systems may support only a range of sizes. This method reports the continuum of discrete sizes between the minimum and maximum as defined by intervals of the divisor (e.g., if given a min of 10 and a max of 50, the discrete values would be 20, 30, 40, and 50).

Either method may be supported. Return codes are:

- 0, "Method completed OK", means success.
- 1, "Method not supported",
- 2, "Choices not available for this Capability." Although the method may be supported by Capabilities in this implementation, it is not supported for this Capability. Usually, this return code indicates that the stripe length has already been set in the parent `StoragePool` and may not be changed.
- 3, "Use [GetSupportedStripeLengths|GetSupportStripeLengthRange] instead". This return code tells the client that this stripe method is not supported, but the other stripe method is supported.

5.5.1.3 Getting Stripe Depth

```
uint32 GetSupportedStripeDepths(
    [Out] uint64 StripeDepths)
```

This method is used to report discrete `UserDataStripeDepths` for `StorageVolume`, `LogicalDisk`, and `StoragePool` creation. Some systems may support only discrete depth byte sizes.

```
uint32 GetSupportStripeDepthRange(
    [Out] uint64 MinimumStripeDepth,
    [Out] uint64 MaximumStripeDepth,
    [Out] uint64 StripeDepthDivisor)
```

This method is used to report a range of possible `UserDataStripeDepths` for `StorageVolume`, `LogicalDisk`, or `StoragePool` creation. Some systems may support only a range of sizes. This method reports the continuum of discrete sizes between the minimum and maximum as defined by intervals of the divisor (e.g., if given a min of 10 and a max of 50, the discrete values would be 20, 30, 40, and 50).

Either method may be supported. Return codes are:

- 0, "Method completed OK", means success.

- 1, "Method not supported"
- 2, "Choices not available for this Capability". Although the method may be supported by Capabilities in this implementation, it is not supported for this Capability. Usually, this return code indicates that the stripe depth has already been set in the parent StoragePool and may not be changed.
- 3, "Use [GetSupportedStripeDepths | GetSupportStripeDepthRange] instead". This return code tells the client that this stripe method is not supported, but the other stripe method is supported.

5.5.1.4 Getting Parity

```
uint32 GetSupportedParityLayouts(
    [Out] ParityLayout[])
```

This method is used to return the type of parity, non-rotated or rotated, that the capability supports.

Return codes:

- 0, "Method completed OK" means success.
- 1, "Method not supported"
- 2. "Choice not available for this Capability." Although the method may be supported by Capabilities in this implementation, it is not supported for this Capability. Usually, this return code indicates that the parity has already been set in the parent StoragePool and may not be changed.

5.5.2 Intrinsic Methods on StorageSetting

The following Intrinsic write methods are supported on StorageSetting:

- DeleteInstance
- ModifyInstance

5.5.3 Extrinsic Methods on StorageConfiguration

5.5.3.1 The RAID characteristics of the new or modified StoragePool

This design supports the implementation choice of the application of RAID striping during either the creation or modification of a StoragePool, StorageVolume, or LogicalDisk. Generally, without the implementation of Clause 15: Extent Composition Subprofile, a client cannot determine the storage elements that are used to represent the RAID striping without at least one StorageVolume or LogicalDisk. Even if the subprofile is supported, the client can make this determination only after each of the supported element types are created.

Once each of the storage element types are created, the client can use the StorageExtents on which the storage element is based to determine the RAID striping type applied. The RAID group is represented by a CompositeStorageExtent instance.

If the ExtentStripeLength property is not supported by an implementation, this design does not provide for interoperable behavior in the creation or modification of StoragePools, StorageVolumes, or LogicalDisks to provide reference to member StorageExtents.

5.5.3.2 Element Naming

Several of the following methods allow a client to 1) specify a name for the storage element that is being created or 2) change the name of a storage element being modified.

If the implementation supports the naming of storage elements, then the ElementName property reports the name assigned to the storage element. If the implementation creates a name even when the client does not specify one, then this element contains that system defined name. If the implementation does not create a name for the storage element when the client does not specify a name, then this property should be null. If the implementation does not

support the naming of elements and the client provides a value in the `ElementName` parameter of one of the following methods that specify an `ElementName` parameter, then the implementation shall reject the method call.

EXPERIMENTAL

The possible `ExtentStripeLengths`, `ExtentStripeDepths`, and `ParityLayouts` for a given `StorageCapabilities` may be fetched using these methods in that class:

- `GetSupportedStripeLengths()`
- `GetSupportedStripeLengthRange()`
- `GetSupportedParityLayouts()`
- `GetSupportedStripeDepths()`
- `GetSupportedStripeDepthRange()` methods

These methods are useful when the `ExtentStripeLength`, `ExtentStripeDepth`, and `ParityLayout` values in the given instance of `StorageCapabilities` are expressed in a range, where the minimum and the maximum are not equal.

EXPERIMENTAL

5.5.3.3 CreateOrModifyStoragePool

```
uint32 CreateOrModifyStoragePool(
    [In] string ElementName
    [Out] CIM_ConcreteJob ref Job,
    [In] CIM_StorageSetting ref Goal,
    [In,out] UInt64 Size,
    [In] string InPools[ ],
    [In] string InExtents[ ],
    [Out] CIM_StoragePool ref Pool);
```

This method is used to create a `StoragePool` from either a source `StoragePool` or a list of `StorageExtents`. Any required associations (such as `HostedStoragePool`) are created in addition to the instance of `StoragePool`. The parameters are as follows:

- **Job:** If a Job was created as a side-effect of the execution of the method, then a reference to that Job is returned through this parameter.
- **Goal:** This is the Service Level that the `StoragePool` is expected to provide. This may be a null value in which case a default setting is used.
- **Size:** As an input this is the desired size of the `StoragePool`. If it is not possible to create a `StoragePool` of the desired size, a return code of "Size not supported" is returned with size set to the nearest supported size.
- **InPools[]:** This is an array of strings containing Object references (see 4.11.5 of *DMTF DSP0200 CIM Operations over HTTP* for format) to source `StoragePools`.
- **InExtents[]:** This is an array of strings containing Object references (see 4.11.5 of *DMTF DSP00200 CIM Operations over HTTP* for format) to source `StorageExtents`. An array of source `StoragePools` or an array of source `StorageExtents` or both can be defined. See 5.1.15.
- **TheElement:** If the method completes without creating a Job, then the `TheElement` is the storage element that is created. Otherwise, `TheElement` may or may not be Null. When the `TheElement` is NULL, then the storage element created can be determined by using the Job model.

5.5.3.4 The CreateOrModifyStoragePool method and the primordial StoragePool

A client may pass a reference to a primordial StoragePool in order to be explicit in indicating from which primordial StoragePool a concrete StoragePool needs to be created. If no StoragePool references are passed in the creation of a StorageVolume or LogicalDisk, the implementation shall determine the parent StoragePool based on the Goal and the Size.

A client may also pass a reference to a primordial StoragePool to express from what reserve to draw capacity if the capacity needed is greater than the total capacity represented by the input StoragePools and StorageExtents. Any capacity request, using the Size parameter, not satisfied by the referenced StoragePools and StorageExtents is drawn from the primordial StoragePool referenced. If no primordial StoragePool reference is passed and the capacity requested is greater than the referenced StoragePools and StorageExtents, then the method shall fail with the “Size not supported” return code. The use of a primordial StoragePool reference in this manner is not recommended, but the behavior is retained to maintain backward compatibility. The client should align the size requested to what can be satisfied by the concrete StoragePools and StorageExtents referenced.

A client should pass only concrete StoragePools when creating a StoragePool from several StoragePools.

5.5.3.5 DeleteStoragePool

```
uint32 DeleteStoragePool(
    [Out] CIM_ConcreteJob ref Job,
    [in] CIM_StoragePool ref Pool);
```

This method allows a client to delete a previously created StoragePool. All associations to the deleted StoragePool are also removed as part of the action. In addition, the RemainingManagedStorage of the associated parent primordial StoragePool will change accordingly.

Note: This method will be denied (“Failed”) if there are any AllocatedFromStoragePool associations where the deleted StoragePool is the Dependent.

5.5.3.6 CreateOrModifyElementFromStoragePool

```
uint32 CreateOrModifyElementFromStoragePool (
    [In,
    string ElementName
    Values {“StorageVolume”, “StorageExtent”,
    “LogicalDisk”},
    ValueMap{“2”, “3”, “4”}]
    Uint16 ElementType;
    [Out] CIM_ConcreteJob ref Job,
    [In] CIM_StorageSetting ref Goal,
    [In, Out] Uint64 Size,
    [In] CIM_StoragePool ref InPool,
    [In, Out] CIM_LogicalElement ref TheElement );
```

This method allows an element of a type specified by the enumeration ElementType to be created from the input StoragePool. The parameters are:

- **ElementType:** This enumeration specifies what type of object to create.
- **Job:** If a Job was created as a side-effect of the execution of the method, then a reference to that Job is returned through this parameter. See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30: Job Control Subprofile.
- **Goal:** This is the Service Level that the element is expected to provide. The Setting shall be a subset of the Capabilities available from the parent StoragePool. Goal may be a null value, in which case the default Setting for the StoragePool is used.
- **Size:** As an input, this is the desired size of the element. If it is not possible to create a StorageVolume of the desired size, a return code of “Size not supported” is returned with size set to the nearest supported size.

- InPool: This shall contain the reference to the source StoragePool.
- TheElement:
 - As Input: If the TheElement parameter is not null, then this method shall attempt to modify the reference element. Otherwise, this method shall attempt to create a new element.
 - As Output: If the method completes without creating a Job, then the TheElement is the storage element that is created. Otherwise, TheElement may be NULL. When the TheElement is NULL, the storage element that is created can be determined by using the Job model.

5.5.3.7 CreateOrModifyElementFromElements

```
uint32 CreateOrModifyElementFromElements(
    [In,
      Values {"Storage Volume", "Storage Pool",
             "Logical Disk"},
      ValueMap{"2","4", "5"}]
    uint16 ElementType,
    [In, Out] CIM_ConcreteJob REF Job,
    [In] CIM_ManagedElement REF Goal,
    [In, Out] uint64 Size,
    [In] CIM_StorageExtent REF InElements[],
    [In, Out] CIM_LogicalElement REF TheElement);
```

The parameters are:

- ElementType: This enumeration specifies the type of object to create.
- Job: If a Job was created as a side-effect of the execution of the method, then a reference to that Job is returned through this parameter. See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30: Job Control Subprofile
- Goal: This is the Service Level that the element is expected to provide. The Setting shall be a subset of the Capabilities available from the parent StoragePool. Goal may be a null value, in which case the default Setting for the StoragePool is used.
- Size: As an input, this is the desired size of the element. If it is not possible to create a StorageVolume of the desired size, a return code of "Size not supported" is returned with size set to the nearest supported size.
- InElements: References to the StorageExtents to be used for the storage element creation or modification. The referenced StorageExtents shall be ComponentExtents of a single StoragePool, a parent of new or existing storage element. The parent StoragePool shall be a direct parent or an indirect parent, a grandparent, of the storage element. The InElements parameter of the CreateOrModifyElementFromElements() parameter is used to provide new StorageExtents to be used for this storage element. Therefore, the use of the parameter in the reduction of capacity for TheElement is invalid.
- TheElement:
 - As Input: If the TheElement parameter is not null, then this method shall attempt to modify the reference element. Otherwise, this method shall attempt to create a new element.
 - As Output: If the method completes without creating a Job, then the TheElement is the storage element that is created. Otherwise, TheElement may be NULL. When the TheElement is NULL, the storage element created can be determined by using the Job model.

5.5.3.8 ReturnToStoragePool

```
uint32 ReturnToStoragePool (
    [Out] CIM_ConcreteJob ref Job,
    [In] CIM_LogicalElement ref Element);
```

This method allows a client to delete a previously created element such as a StorageVolume.

EXPERIMENTAL

5.5.3.9 RequestUsageChange

```
uint32 RequestUsageChange (
    [In,
        ValueMap { "2", "3" },
        Values { "Set", "Modify \"Other\" description only"
    }]
    uint16 Operation,
    [In] uint16 UsageValue,
    [In] string OtherUsageDescription,
    [Out] CIM_ConcreteJob ref Job,
    [In] CIM_LogicalElement ref TheElement);
```

The parameters are:

- Operation: This specification defines the usage of the 2 “Set” value for the parameters, which means to set the Usage to one of the possible usage values. This parameter is required.
- UsageValue: The usage value possible for the type of storage element, whose reference is passed to this method. This parameter is required.
- OtherUsageDescription: Not defined this specification. This parameter is not required.
- Job: If a Job was created as a side-effect of the execution of the method, then a reference to that Job is returned through this parameter. See *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30: Job Control Subprofile
- TheElement: This requirement parameter contains a reference to the storage element whose usage is to be changed.

If the storage element can not be changed to the requested usage because it is invalid to do so, then the implementation shall return an invalid parameter error.

EXPERIMENTAL

5.5.3.10 Return Values

Each method has this set of defined return codes:

```
ValueMap { "0", "1", "2", "3", "4", "5", "6", "..", "4096", "4097" },
Values { "Job completed with no error", "Not Supported", "Unknown",
    "Timeout", "Failed", "Invalid Parameter", "In Use", "DMTF Reserved",
    "Method parameters checked - job started", "Size not supported" }
```

Only the following return codes shall be supported:

- 0 - “Job completed with no error”
The method has completed immediately with no errors (and with no asynchronous execution required).
- 1 - “Not Supported”
This method is not supported at this time.
- 3 - “Timeout” or 4 - “Failed”
The provider has problems accessing the hardware (or other implementation-specific reasons)⁴. The provider should return a standard message communicating the nature of the value rather than returning this code.

- 5 - “Invalid Parameter”
One or more of the parameters are invalid (invalid object paths, for instance). The provider should return a standard message, communicating which parameters are invalid and why, rather than returning this code.
- 4096 - “Method parameters checked - job started”
The method parameters have been checked, and the method is being executed asynchronously.
- 4097 - “Size not supported”
For a Create/Modify method, the requested size is not supported. The Size parameter and the size of the storage element is set to the nearest supported and larger size.). Only the methods that create or modify storage elements, other than their usage, shall return this code.

A vendor shall not extend the Value map to express vendor specific error situations not catered for by the standard messages.

EXPERIMENTAL

5.5.3.11 GetElementsBasedOnUsage

```
uint GetElementsBasedOnUsage(
    [In,
        ValueMap { "2", "3", "4", "5" }
        Values { "StorageVolume", "StorageExtent",
            "StoragePool", "Logical Disk", } ]
    uint16 ElementType,
    [In] uint16 Usage,
    [In,
        ValueMap { "2", "3", "4" },
        Values { All, "Available Only", "In Use Only" } ]
    uint16 Criterion,
    [In] CIM_StoragePool ref ThePool,
    [Out] CIM_ManagedSystemElement ref TheElements[ ] );
```

All input parameters are required. The parameters are:

- ElementType: This enumeration specifies the type of object to create.
- UsageValue: The usage value possible for the type of storage element as indicated by the ElementType parameter.
- Criterion: Specifies whether to retrieve all elements - 2 “All”, available elements only - 3 “Available Only”, or the elements that are in use - 4 “In Use Only”.
- ThePool: Limits the search for the elements that satisfy the criteria in this StoragePool only. If null, all appropriate storage pools shall be included in the search.
- TheElements: Contains the array of references found to the storage element instances retrieved.

This method returns the following statuses:

- 0 - “Completed with No Error”:
The method has completed immediately with no errors
- 1 - “Not Supported”
This method is not supported at this time.
- 3 - “Timeout” or 4 - “Failed”
The provider has problems accessing the hardware (or other implementation-specific reasons)’. The provider should return a standard message communicating the nature of the value rather than returning this code.

- 5 - “Invalid Parameter”
One or more of the parameters are invalid (invalid object paths, for instance). The provider should return a standard message, communicating which parameters are invalid and why, rather than returning this code.

EXPERIMENTAL

5.5.4 Extrinsic Methods on StoragePool

5.5.4.1 General

The Extrinsic methods on StoragePool return sizes in units of bytes. These methods, each described in this section, are:

- GetSupportedSizes
- GetSupportedSizeRange
- GetAvailableExtents

5.5.4.2 GetSupportedSizes

```
uint32 GetSupportedSizes(
    [In] uint16 ElementType,
    [In] CIM_StorageSetting ref Goal,
    [Out] uint64 Sizes[ ]);
```

The parameters are:

- ElementType: This enumeration specifies what type of object to create.
- Goal: The Service Level the element is expected to provide. The setting shall be a subset of the Capabilities available from the parent StoragePool. Goal may be a null value, in which case the default Setting for the StoragePool is used.
- Sizes: An array containing all the possible sizes of an element in a creation or modification operation. If a possible value is repeated in the array, then that value shall be repeated. The sum of the sizes is the total remaining space for that goal.

This method is used to determine the possible sizes of child elements, e.g., StoragePool, StorageVolume or LogicalDisk, that can be created or modified using capacity from the StoragePool. The method is used for storage systems where discrete sizes are possible. This method is useful if the possible sizes do not differ from a fixed amount. One of the reported sizes can be used directly along with the Goal in the creation of a StoragePool, StorageVolume, or LogicalDisk. The method reports the continuum of discrete sizes between the minimum and maximum as defined by intervals of the divisor (e.g., if given a min of 10 and a max of 50, the discrete values would be 20, 30, 40, and 50).

5.5.4.2.1 GetSupportedSizeRange

```
uint32 GetSupportedSizeRange(
    [In] uint16 ElementType,
    [In] CIM_StorageSetting ref Goal,
    [Out] uint64 MinimumVolumeSize,
    [Out] uint64 MaximumVolumeSize,
    [Out] uint64 VolumeSizeDivisor);
```

- ElementType: This enumeration specifies what type of object to create.
- Goal: The service level the element is expected to provide. The Setting shall be a subset of the Capabilities available from the parent StoragePool. Goal may be a null value, in which case the default Setting for the StoragePool is used.

- **MinimumVolumeSize:** The minimum size an element can take on either as a creation or modification operation.
- **MaximumVolumeSize:** The maximum size an element can take on either as a creation or modification operation
- **VolumeSizeDivisor:** The value used to determine sizes between **MinimumVolumeSize** and **MaximumVolumeSize**.

This method is used to determine the possible sizes of child element, e.g., **StoragePool**, **LogicalDisk**, and **StorageVolume**, that can be created or modified using capacity drawn from the **StoragePool**. The out parameters tell the minimum element size, maximum element size, and possible sizes in that range. This method is useful when the number of possible sizes is so voluminous that reporting each discrete size would be impractical.

Both or either method may be supported by a storage subsystem, either as a decision made at implementation time or as a variable that depends on the state of the **StoragePool**. For example, when a **StoragePool** is first created allowing for possible sizes to be in 1024-byte blocks, the **GetSupportedSizeRange** method should be used to report possible sizes. This example **StoragePool** does not relocate blocks to avoid fragmentation of the capacity. As **StorageVolumes** or **LogicalDisks** are drawn from and returned to the **StoragePool**, the capacity becomes fragmented. In this case, the **GetSupportedSizes** method should be used to report the non-continuous regions of capacity that may be used for element creation. There are storage systems that can allocate the **StorageVolume** or **LogicalDisk** only in whole disks that need not be of uniform size; such storage systems support only the **GetSupportedSizes** method.

Both methods may be supported at the same time and may report different values when discontinuous and contiguous capacity is present in the **StoragePool**. In this case, the **GetSupportSizes** method is used to report the fragments of available capacity. The remaining contiguous capacity is reported as the largest element size possible. The **GetSupportSizeRange** is used to report element sizes that may be drawn from the contiguous capacity.

If there is no notion of continuity as being a stable state of the system, e.g., capacity is continuously and automatically being defragmented, the **GetSupportSizeRange** method should be used.

5.5.4.2.2 Return Values

Each method has this set of return codes:

```
ValueMap {"0", "1", "2"},
Values {"Method completed OK", "Method not supported", "Use <the other method
name> instead" }
```

If the above methods do not complete successfully, then either the methods are not supported or the other method should be used. The **GetSupportSizes** method can notify the SMI-S client that it should use the **GetSupportSizeRanges** instead; the **GetSupportedSizeRange** method can notify the SMI-S client that it should use the **GetSupportedSizes** method instead.

5.5.4.2.3 GetAvailableExtents

```
uint32 GetAvailableExtents(
    [In] CIM_StorageSetting REF Goal,
    [Out] CIM_StorageExtent REF AvailableExtents[ ] );
```

This method is used to retrieve the available **StorageExtents**—**ComponentExtents** of the **StoragePool**—that do not form the basis for **StorageVolumes** and **LogicalDisks** allocated from the **StoragePool**. If a **NULL** is passed for a **Goal**, then all the available **ComponentExtents** of the **StoragePool** are returned.

The **StorageExtent** references returned from this method refer to a subset of the **StorageExtents** associated to the **StoragePool** via **ConcreteComponent**, **AssociatedComponentExtent**, and **AssociatedRemainingExtent**. The **StorageExtents** referenced by the output of this method may not equal the set of **Component StorageExtents** because of any of the following reasons:

- The excluded **StorageExtents** may not be used with the **Goal**.

- The excluded StorageExtents may not be used for vendor-specific reasons.
- The excluded StorageExtents may not be used because of a usage restriction.

This method is designed as a companion for the CreateOrModifyElementFromElements method. A client may fetch the StoragePool's available ComponentExtents and attempt to call CreateOrModifyElementFromElement, or the client may use this method and have the agent provide the available StorageExtents. However, note it is possible that even though a StorageExtent may appear to be available from the implementation's model, the implementation may not allow the StorageExtent to be used for vendor specific reasons.

5.5.4.3 Return Values

Each method has this set of defined return codes:

```
ValueMap { "0", "1", "2", "3", "4", "5"},
Values { "Job completed with no error", "Not Supported", "Unknown",
        "Timeout", "Failed", "Invalid Parameter" }
```

- 0 - "Job completed with no error"
The method completes immediately with no errors (and with no asynchronous execution required)
- 1 - "Not Supported"
The implementation does not support the method.
- 5 - "Invalid Parameter"
One of the method parameters is incorrect (for instance invalid object paths).
- 3 - "Timeout" or 4 - "Failed"
The provider had problems accessing the hardware, or there were implementation-specific problems.

5.5.4.3.1 Storage Element Modification

Concrete StoragePools may be expanded, shrunk, or have their quality of service (QoS) changed (the Goal parameter) by a client.

This package does not define how primordial StoragePools are modified (if they can be modified) within a particular implementation.

The current capacity of a StoragePool is the value of the TotalManagedSpace property.

StorageVolumes and LogicalDisks may be expanded, shrunk, or have their quality of service (QoS) changed (the Goal parameter) by a client.

The current capacity of the StorageVolume, LogicalDisk, or StorageExtent is the ConsumableBlocks times the BlockSize.

Storage elements are StoragePools, StorageVolumes, and LogicalDisks.

Return values are:

- 5 "StoragePool QoS Change," 6 "StoragePool Capacity Expansion," 7 "StoragePool Capacity Reduction"
Within SupportedStoragePoolFeatures array within the StorageConfigurationCapabilities instance, indicates the types of StoragePool modification allowed.
- 11 "Storage Element QoS Change, 12 "Storage Element Capacity Expansion", and 13 "Storage Element Capacity Reduction"
Within the SupportedStorageElementFeatures array within the StorageConfigurationCapabilities instance, indicates the types of StorageVolume and LogicalDisk modifications allowed.

An implementation may support one or more of these options. If the implementation supports capacity expansion or capacity reduction options and the QoS change option, then it shall support the capacity change and the QoS change simultaneously in the modification of a given storage element.

A client can determine the resultant usable capacity to which a storage element may be changed by using the `GetSupportedSizes()` and `GetSupportedSizeRange()` methods on the parent `StoragePool`. These methods provide the possible storage capacity for new storage elements and for the modification of existing storage elements given a QoS goal. To obtain a size to use for storage element modification, the client simply select a size returned from the `GetSupportedSizes()` method or a size within the range returned from `GetSupportedSizeRange()` method.

Generally, the attempted `StoragePool` modification shall be characterized as a storage capacity expansion if the new capacity (the `Size` parameter) is greater than the current value of the `TotalManagedSpace` property of the `StoragePool` to be modified. Likewise, the attempted `StoragePool` modification shall be characterized as a storage capacity reduction if the desired new capacity (the `Size` parameter) is less than the current value of the `TotalManagedSpace` property of the `StoragePool` to be modified.

Generally, the attempted `StorageVolume` or `LogicalDisk` modification shall be characterized as a storage capacity expansion if the new capacity (the `Size` parameter) is greater than its current capacity. Likewise, the attempted `StorageVolume` or `LogicalDisk` modification shall be characterized as a storage capacity reduction if the desired new capacity (the `Size` parameter) is less than its current capacity.

A storage element may also be modified by providing the references to component `StorageExtents`. The list candidate component `StorageExtents` shall be provided through the execution of the `GetAvailableExtents()` method on the parent `StoragePool`. For example, the SMI-S Client determines which `StorageExtents` to use from the returned list based on their performance characteristics or their relationship to network ports or primordial storage.

A `StoragePool`'s capacity may be expandable by providing the references to existing component `StorageExtents` of the `StoragePool` and additional references to component `StorageExtents`. A `StoragePool`'s capacity may be reducible by providing references to some, but not all, of the current component `StorageExtents` of the `StoragePool`. If the summary of the capacity of the referenced input `StorageExtents` is greater than the `TotalManagedSpace` of the `StoragePool`, then this action shall be characterized as a capacity expansion. If this summary is less than the `TotalManagedSpace` of the `StoragePool`, then this action shall be characterized as capacity reduction.

A `StorageVolume`'s or `LogicalDisk`'s capacity may be expandable by providing references to additional component `StorageExtents` of the parent `StoragePool`. The capacity of a `StorageVolume` or `LogicalDisk` shall not be reducible by providing references to `StorageExtents`.

The capacity of storage elements that have only one member `StorageExtent` can only be reduced by passing a reference to the existing member and specifying a capacity, using the `Size` parameter, that is smaller than the current size of the storage element.

The specified `Size` parameter (in bytes), along with the specification of member `StorageExtents`, indicates how much of the provided `StorageExtents` is to be used for the storage element. The specified size represents the desired consumable capacity of the storage element. The capacity of the `StorageExtent` may be equal to either the capacity drawn in its creation from a parent `StorageExtent` or `StoragePool` or to the capacity that may be drawn from it in the creation of a dependent storage element. No direct comparison may be made by the client between the desired capacity and the capacity of the `StorageExtents`.

If the capacity desired is equal to the capacity of the storage element and the QoS is not altered, then the implementation shall return no error and start no job.

If the capacity requested is larger than is consumable given a QoS (new or existing) from the referenced `StorageExtents` or `StoragePools`, then that capacity shall be drawn from the parent primordial `StoragePool`. The effect of passing a capacity less than the current capacity of the storage element shall be to make available or free the capacity in the member `StorageExtents` to the difference between the current capacity of the storage element and the new capacity of the storage element. The amount of capacity freed depends on the virtualization (e.g.,

RAID method) employed in the previous configuration of the storage element. An invalid parameter error shall be produced if the capacity in bytes passed is less than the current capacity but greater than the capacity realizable from the StorageExtents referenced given a QoS. The size of a StorageExtent is the NumberOfBlocks times the BlockSize. The capacity of the StorageExtents references can be calculated; it is the sum of the sizes of all StorageExtents.

The number of StorageExtents desired, including existing and additional StorageExtents, for a StorageElement minus the PackageRedundancy shall be equal to the ExtentStripeLength times the DataRedundancy specified in the existing QoS goal. Clause 15: Extent Composition Subprofile defines how to determine the number of primordial StorageExtents used.

The quality of service (QoS) of a storage element may be modified. Generally, a QoS change indicates a reorganization of computing resources to meet the new requirements—either additional or fewer computing resources are used.

If the QoS is being modified, then clients may not be able to determine if desired size of the storage element constitutes an expansion or reduction, as specified previously. Such a modification shall be non-destructive to the data stored.

The QoS of a StoragePool shall not be changeable if that StoragePool has children storage elements. However, the package redundancy of parental StoragePools may be changed by changing the number of spare StorageExtents. See Clause 12: Disk Sparing Subprofile.

In the totality of this design, a SMI-S Client may change one of the following:

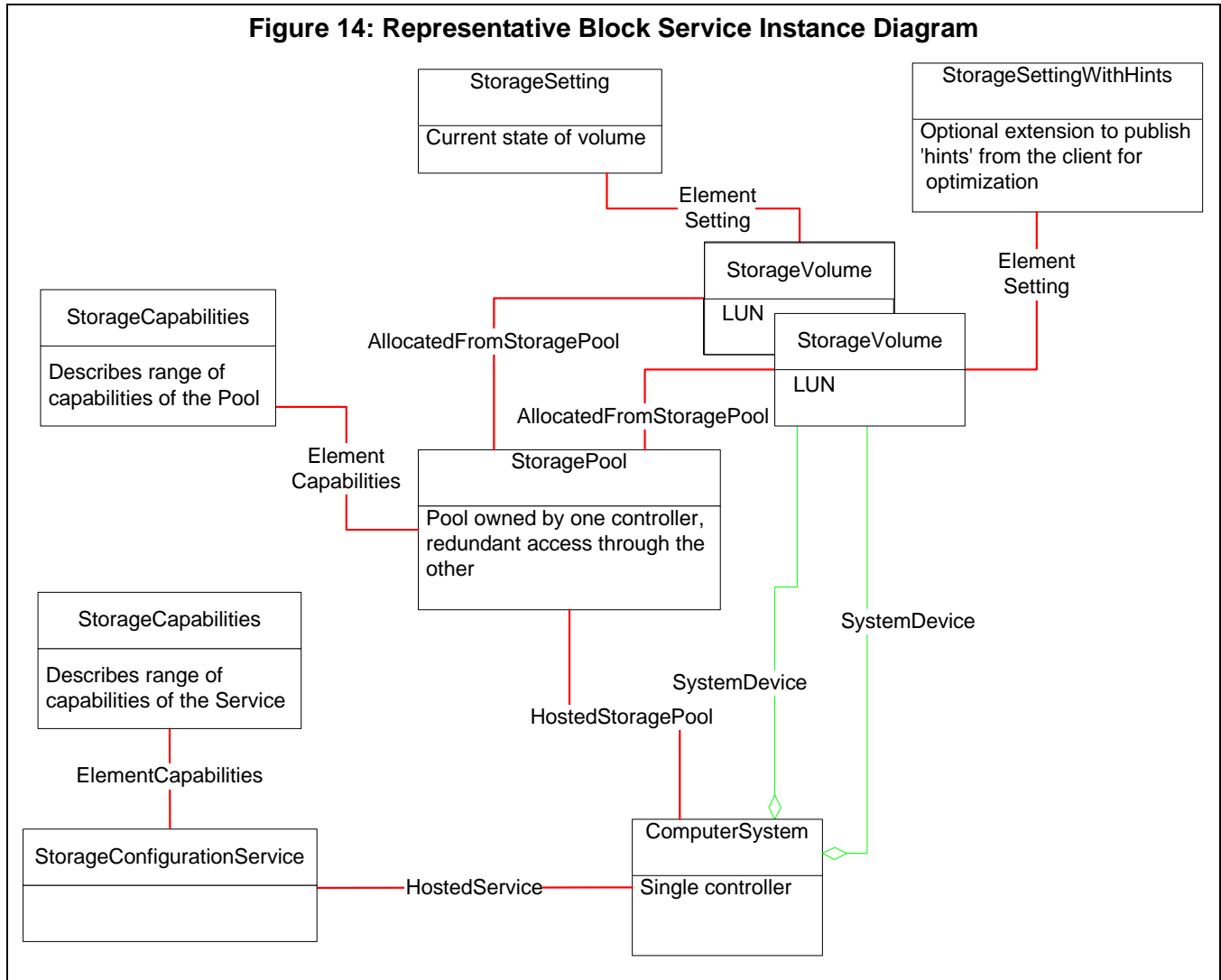
- The QoS,
- The Size (capacity)
- The Size and the member StorageExtents
- Only the member StorageExtents.

A SMI-S Client may not change the QoS and the member StorageExtents. There is no mechanism for a SMI-S Client to determine the quorum of StorageExtents for a given QoS if ExtentStripeLength is not provided.

5.6 Client Considerations and Recipes

5.6.1 Representative Instance Diagram

Figure 14 shows the classes and associations needed to model a single StoragePool with two StorageVolumes.



5.6.2 Goals and Settings

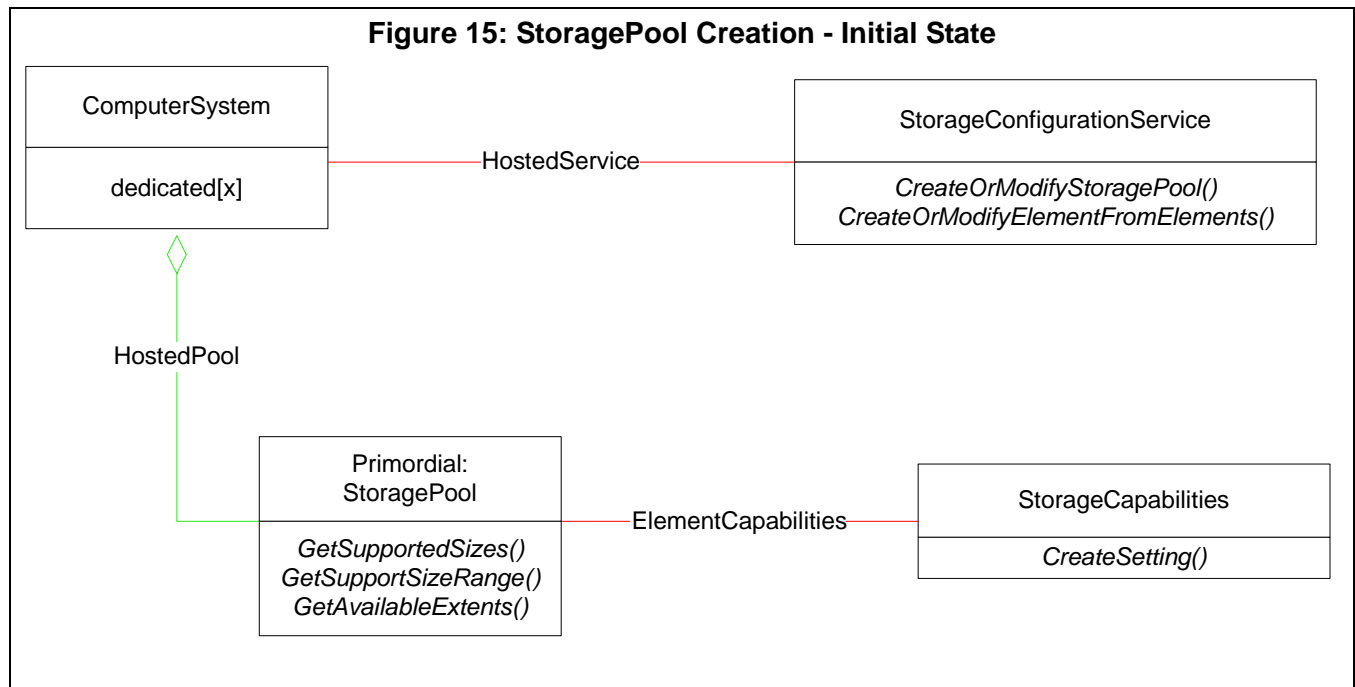
A implementation may persist the properties of the Setting as they were when the Setting was used to perform a configuration operation. However, the implementation may also construct the Setting given the current quality of service provided. An implementation of this package should retain the properties of the Setting as they were when the Setting was used as a Goal. For example, a client requests a package redundancy 2, the implementation is restarted and therefore cannot retrieve; the implementation sets this value to the current value of 1. Unless the client maintained the state of Setting as well, it will not be able to detect the difference between the initial Setting state and the current state for package redundancy, in the StorageVolume or LogicalDisk, for example.

If a client specifies a goal asking for no single point of failure, the implementation shall return an error if the system is not capable of supporting that goal. However, if a client specifies that single points of failure are allowed, the implementation may return storage that has potential single points of failure or it may return storage that has no single points of failure. In other words, the system may return a storage that is more capable than what the client has asked for.

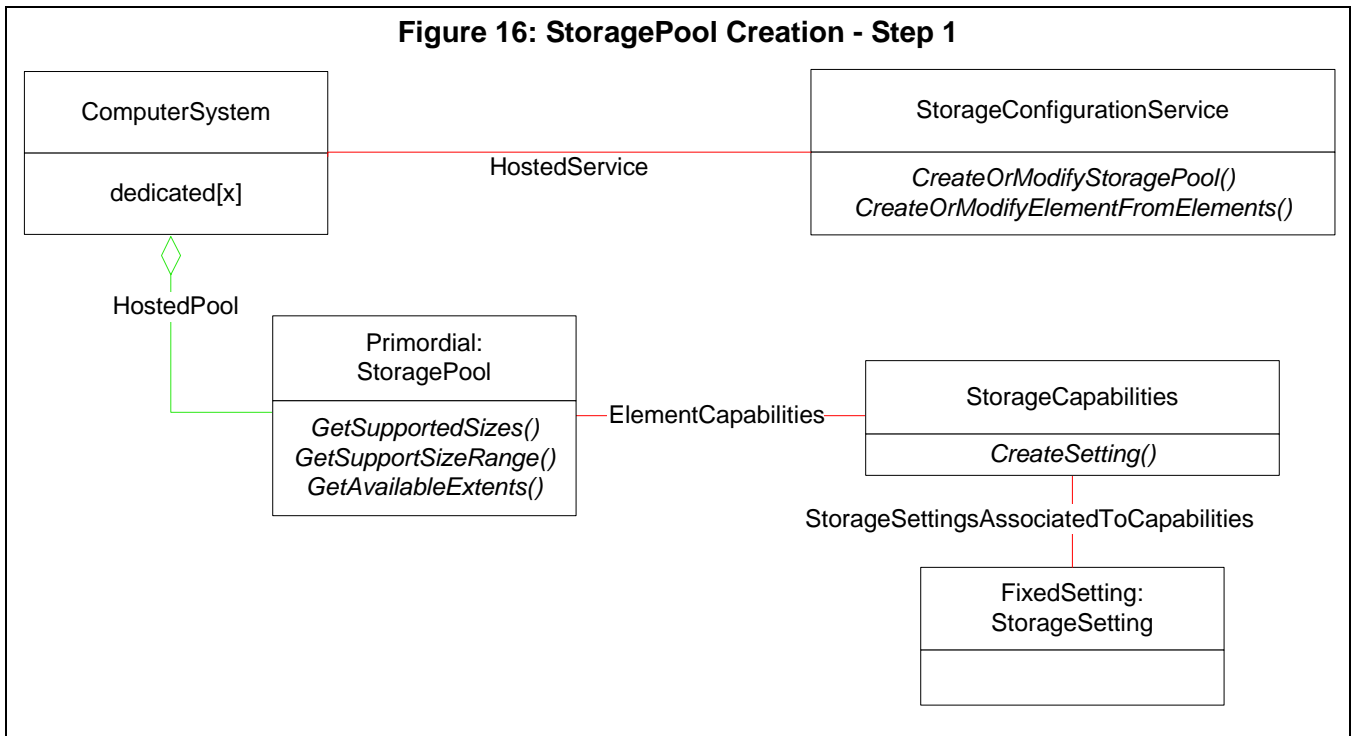
A client may request more data redundancy and package redundancy than what is required for the particular RAID level. An implementation may provide more of these redundancies than is required for its RAID levels. If allowed, the client request of additional data redundancy indicates that additional copies of the data are requested. If allowed, the client request of additional package redundancy results in additional drives, for example, being assigned to this storage element. The redundant package may be overassigned (e.g., assigned as extra packages for more than one storage element), or it may be dedicated. See Clause 12: Disk Sparing Subprofile for details on modeling the sparing functionality itself. In other words, these Goal properties can be used to assign additional copies of the data and redundancy at creation or modification time of a StoragePool, StorageVolume, or LogicalDisk.

5.6.3 Representative StoragePool Creation Example

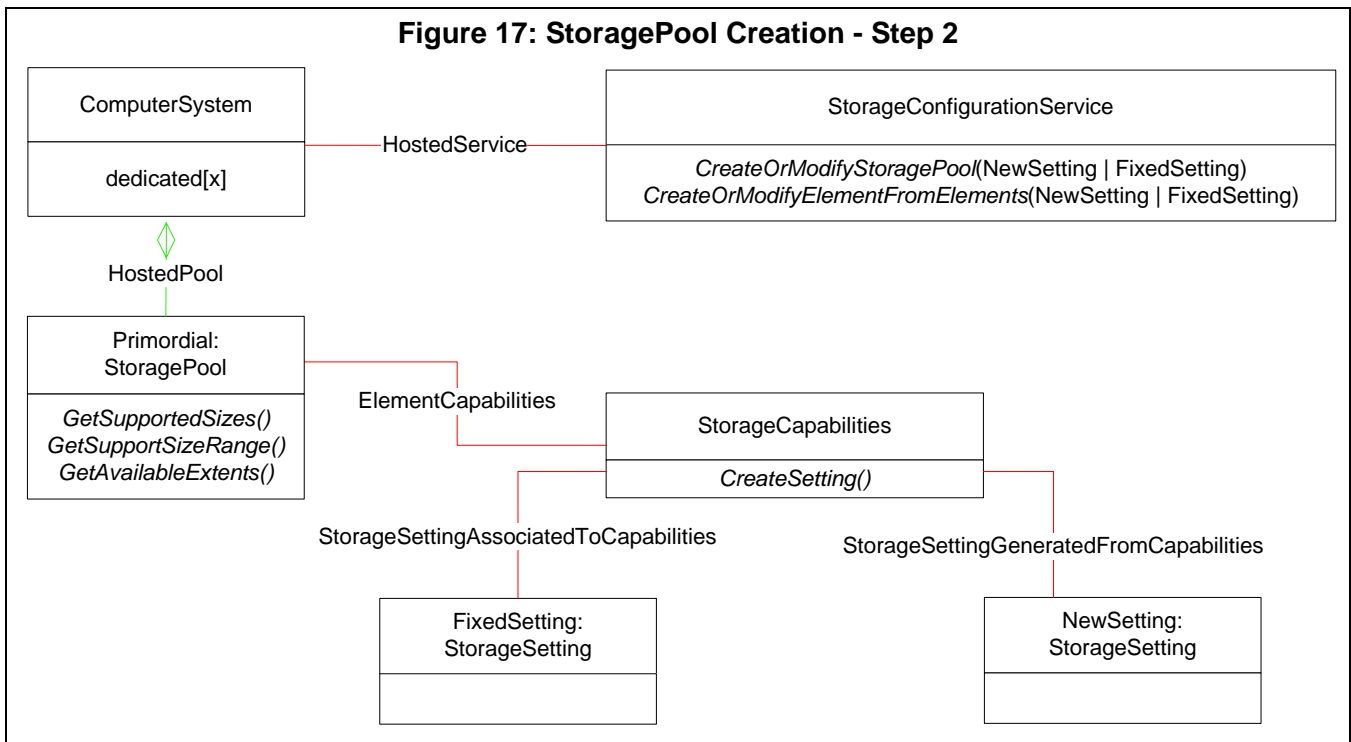
Figure 15 shows the initial state of the block storage system, a single primordial StoragePool that advertises its capabilities. The `GetSupportedSizes()` and `GetSupportedSizeRange()` methods determine what sizes of StoragePools can be created from the primordial StoragePool, given a goal StorageSetting. Alternatively, if the StoragePool is to be created from StorageExtents, `GetAvailableExtents()` obtains a list of available ComponentExtents of the StoragePool that also match the Goal.



Next, (Figure 16) the client uses the `CreateSetting` method on the `StorageCapabilities` instance to create an instance of a `StorageSetting`. This `Setting` object can be altered as desired. If the block storage system supports `StorageSettingWithHints`, an instance of this subclass is created rather than the `StorageSetting` superclass. Alternatively, the client can use one of the predefined `StorageSetting` instances. Pre-existing `Settings` can be located by using the `StorageSettingsAssociatedToCapabilities` association for factory or pre-defined settings or by using the `StorageSettingsGeneratedFromCapabilities` class, where the `StorageSetting.ChangeableType` = "2" ("Changeable - Persistent"); these `Settings` have been generated but were modified to persist.

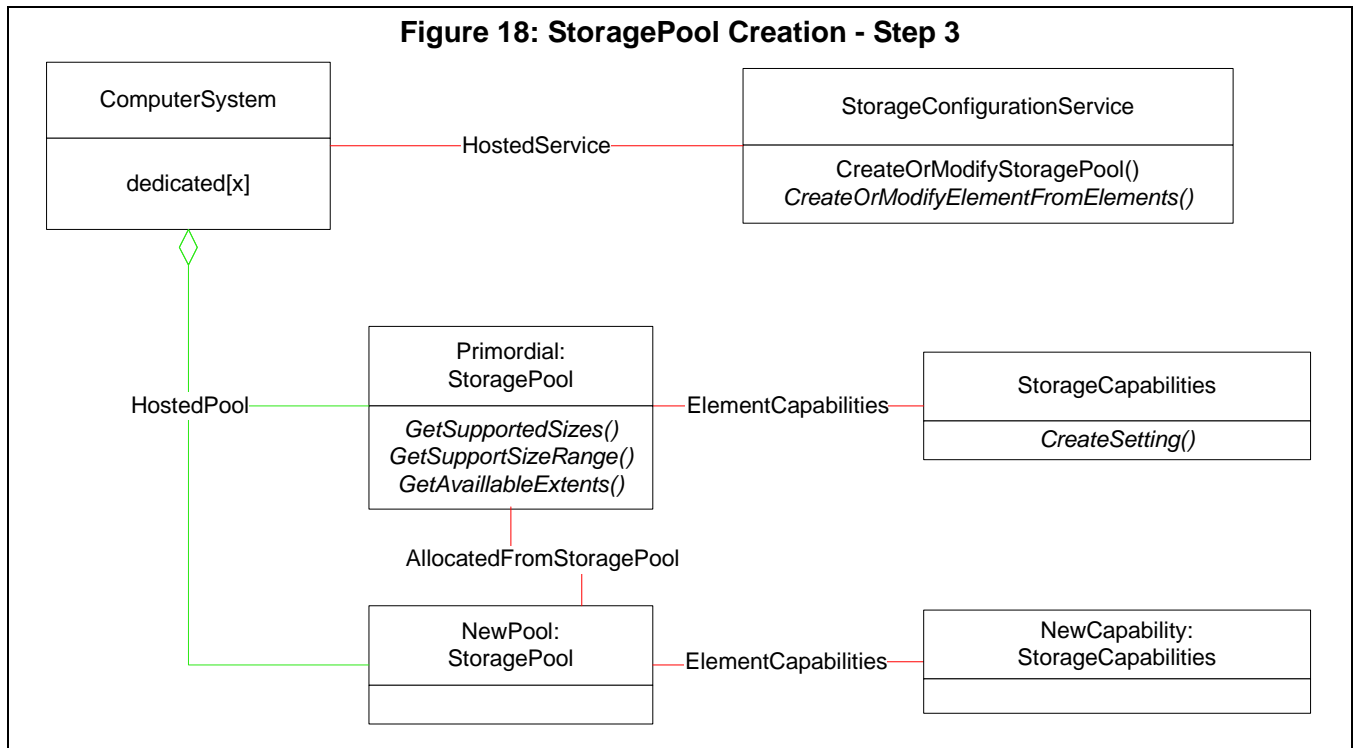
Figure 16: StoragePool Creation - Step 1

Once this generated Setting has been altered as required or, alternatively, a pre-defined Setting used, the Goal Setting is passed as an argument to the `CreateOrModifyStoragePool` method in the `StorageConfigurationService`. (Shown in Figure 17).

Figure 17: StoragePool Creation - Step 2

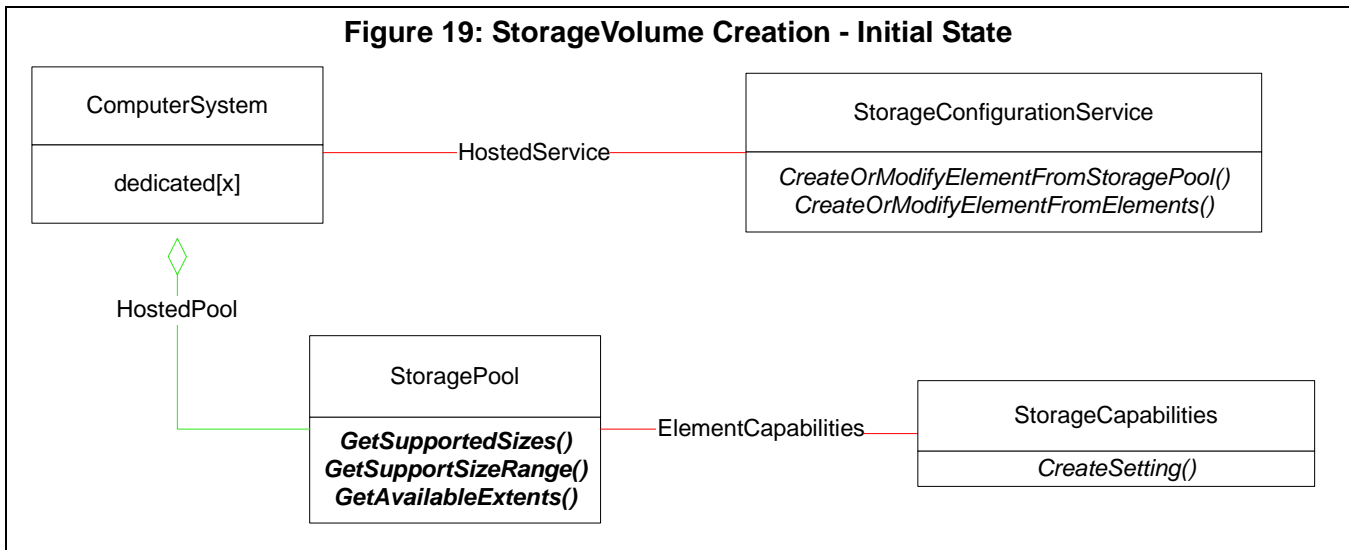
Alternatively, the client can create the StoragePool by passing the Goal, the desired ComponentExtents, and a "Pool" ElementType to CreateOrModifyElementFromElement. If a Size is passed as well, the size shall be equal to or less than the consumable size (in blocks) of the desired ComponentExtents. The list of available StorageExtents is best retrieved using the GetAvailableExtents() method. If the Size is less than the desired StorageExtents by less than the smallest StorageExtent passed, then one of the StorageExtents is partitioned into used and free parts. See 5.1.15.

The StoragePool is then created, as shown in Figure 18. If the generated Setting was used as the Goal, then this temporary StorageSetting is replaced with an equivalent object linked to the new StoragePool with ElementCapabilities. .

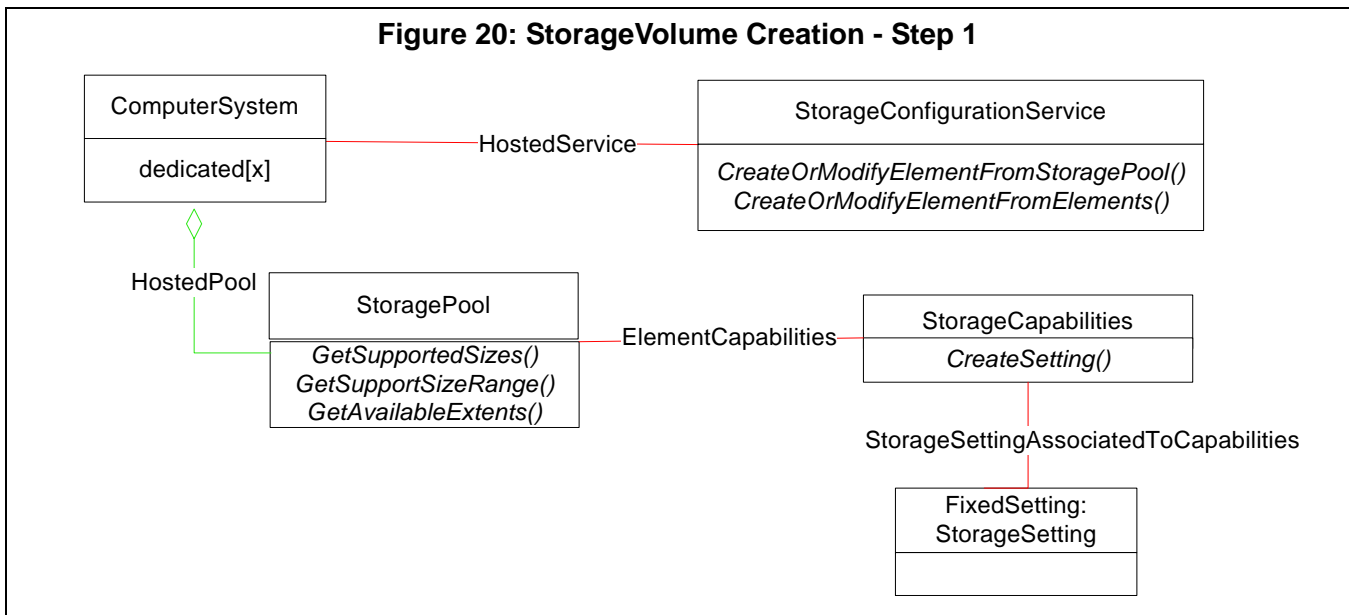


5.6.4 Representative example of StorageVolume or LogicalDisk Creation

Similarly to StoragePools, a client chooses a suitable source StoragePool by referencing the StorageCapabilities objects and using the GetSupportedSizes() and GetSupportSizeRange() methods, given a goal Setting. Alternatively, a client can retrieve the available ComponentExtents of the StoragePool, given a goal StorageSetting, with the GetAvailableExtents() methods. The client may create a StorageVolume or LogicalDisk by specifying a size, source StorageExtents, or a combination, as shown in Figure 19.

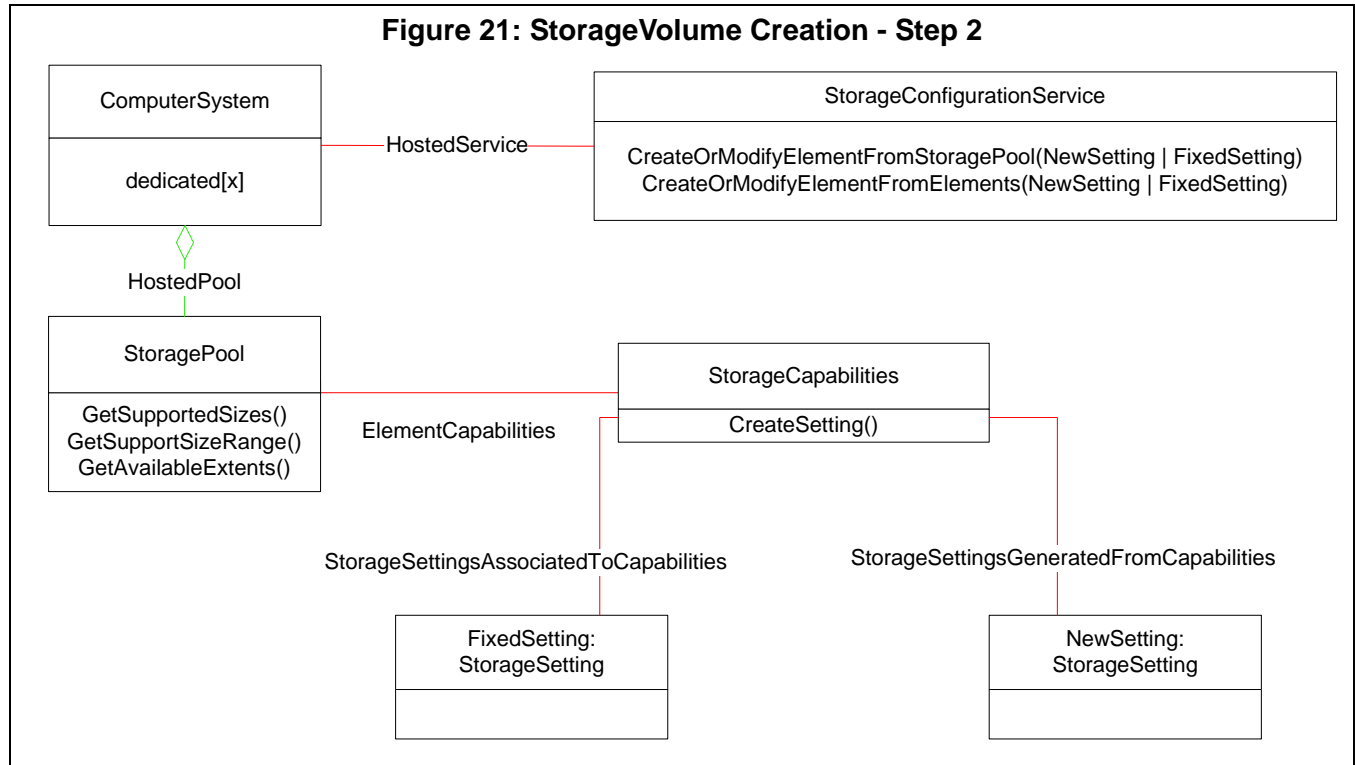
Figure 19: StorageVolume Creation - Initial State

Once a suitable StoragePool is found, a StorageSetting instance can be created using the CreateSetting method on the StorageCapabilities object. See Figure 19. If a suitable StorageSetting already exists, it can be used instead. Pre-existing Settings can be located by using the StorageSettingsAssociatedToCapabilities association, for factory or pre-defined settings, or by using the StorageSettingsGeneratedFromCapabilities where the StorageSetting.ChageableType = "2" ("Changeable - Persistent"); these Settings have been generated but were modified to persist, as illustrated in Figure 20. Another Setting already associated to a storage element can be used as a goal, but it shall not be modifiable.

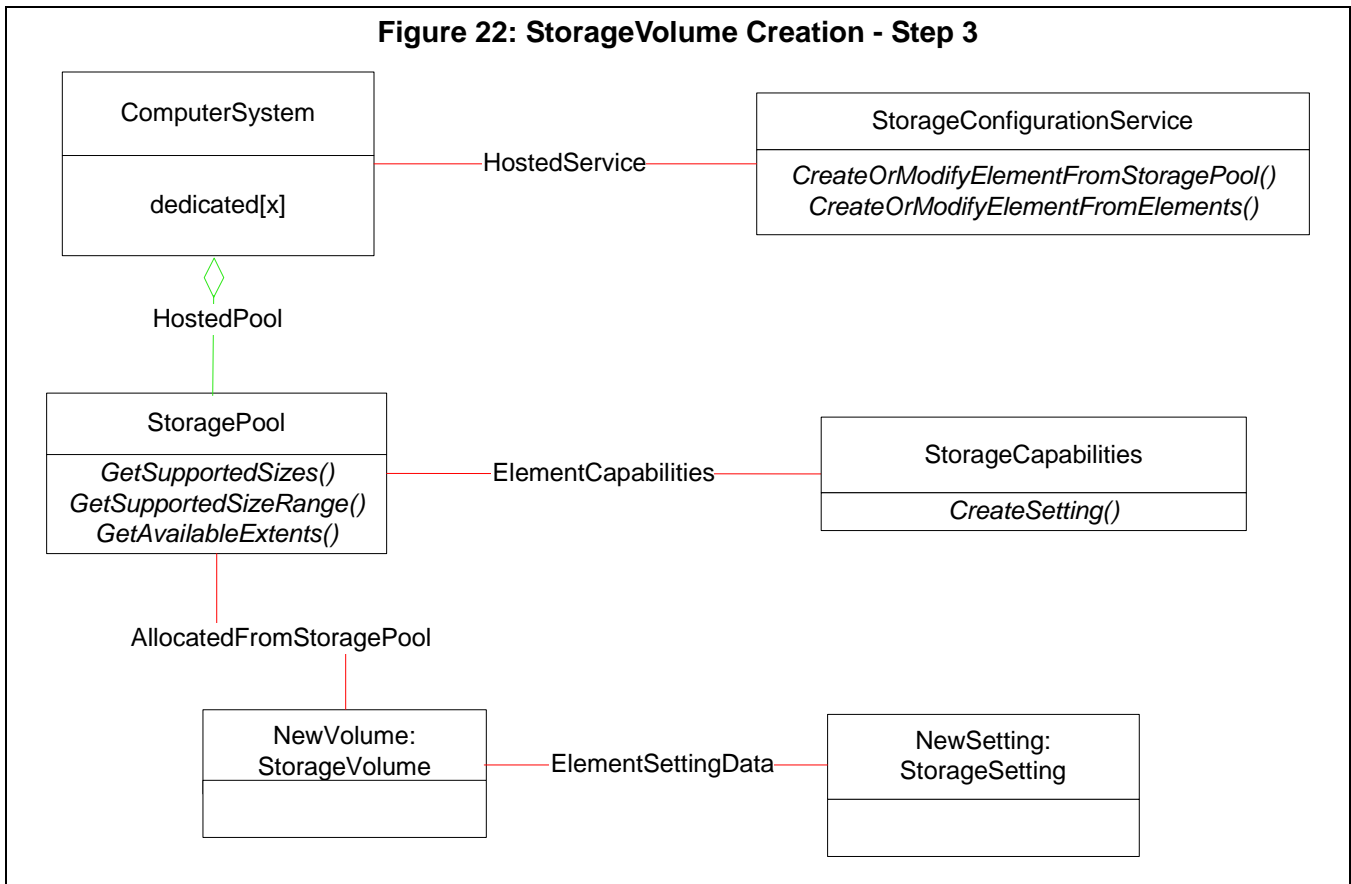
Figure 20: StorageVolume Creation - Step 1

If a new Setting is created, it is linked back to the originating StorageCapabilities object until it is used as an argument in a StorageConfiguration method. See Figure 21. Alternatively, the client can create the StorageVolume or LogicalDisk, for example, by passing the Goal, the desired ComponentExtents, and a ElementType to CreateOrModifyElementFromElement. If a Size is passed as well, the size shall be equal to or less than the consumable size (in blocks) of the desired ComponentExtents. The list of available StorageExtents is best retrieved using the GetAvailableExtents() method. If the Size is less than the desired StorageExtents by a size less

than smallest StorageExtent passed, then one of the StorageExtents is partitioned into used and free parts. See 5.1.15.



Once the StorageVolume has been created, the new or existing Setting is associated to the new storage element using the ElementSettingData association. The new Setting and the Goal setting may not be the very same instance. The client cannot assume that the instances are the *same* instance. See Figure 22.

Figure 22: StorageVolume Creation - Step 3

5.6.5 Summarize the StoragePools in a block storage system and verify the capacity reported

```

// DESCRIPTION
// This recipe retrieves and validates the total, remaining and consumed storage
// pool space on a block server.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the device, CIM_ComputerSystem, of interested has
//    previously been identified and defined in the $BlockServer-> variable.

// Step 1. Retrieve the storage pools on the device.
$Pools[] = Associators($BlockServer->,
    "CIM_HostedStoragePool",
    "CIM_StoragePool",
    "GroupComponent",
    "PartComponent",
    false,
    false,
    {"TotalManagedSpace", "RemainingManagedSpace"})

// Step 2. Summarize the space consumed and available in each storage pool.
for (#i in $Pools[]) {

```

```

#totalSpace = $Pools[#i].TotalManagedSpace
#remainingSpace = $Pools[#i].RemainingManagedSpace
$Pool-> = $Pools[#i].getObjectPath()

// Step 3. Retrieve the space consumed by each element allocated from the
// storage pool.
$Allocs[] = References($Pool->,
    "CIM_AllocatedFromStoragePool",
    "Antecedent",
    false,
    false,
    {"SpaceConsumed"})

#allocSpace = 0
for (#j in $Allocs[]) {
    #allocSpace = #allocSpace + $Allocs[#j].SpaceConsumed
}
if (#totalSpace != #allocSpace + #remainingSpace) {
    <ERROR! Device does not correctly represent capacity>
}
}

```

5.6.6 Create StoragePool and Storage Element on Block Server (e.g., Array or Volume Manager)

```

// DESCRIPTION
// The goal of this recipe is to create a storage element with the
// maximum capabilities of the block server.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.A reference to a CIM_ComputerSystem storage array is previously
//    defined in the $BlockServer-> variable
// 2.The settings for the new Storage Pool and Storage Volume or Logical Disk have
//    following size:
//    #RequestedSize      = 10 * 1024 * 1024 * 1024 // 10 GB
// 3.#StorageElementClass is set to the class name of the element being created
//    like CIM_StorageVolume or CIM_LogicalDisk.
// 4. #ElementType is set to the element to created
//    See CreateOrModifyElementFromStoragePool.ElementType

// Function GetMostCapable
// Get the capabilities that have the maximum DataRedundancy and
//    PackageRedundancy
// Input:
// An array of StorageCapabilities instances associated to the StoragePool.
sub REF GetMostCapable($CapabilitiesOffered[])
{

```

Block Services Package

```
<Sort the $CapabilitiesOffered[] so that the capability with the
greatest DataRedundantMax, PackageRedundancyMax, and
NoSinglePointOfFailure in the last element in the array.
NoSinglePointOfFailure == true is greater than
NoSinglePointOfFailure == false
>

return $CapabilitiesOffered[$CapabilitiesOffered.length-1]
}

// Function PoolSizeAvailable
// A return value of 0 means that no size is available
sub unit32 PoolSizeAvailable($PoolToDrawFrom->,
    $StorageSetting->, #RequestedSize, #RequestedElementType)

#ResultSize = 0
%InArguments["ElementType"] = #RequestedElementType
%InArguments["Goal"] = $StorageSetting->
#MethodReturn = InvokeMethod(
    $PoolToDrawFrom->,
    "GetSupportedSizes",
    %InArguments,
    %OutArguments)
if(#MethodReturn == 0)
{
    // this method is supported
    #SupportedSizes[] = %OutArguments["Sizes"]
    #i = 0
    #max = #SupportedSizes[].length
    while(#i < #max && #RequestedSize > #ResultSize)
    {
        #ResultSize = #SupportedSizes[#i++]
    }
    if(#RequestedSize > #ResultSize)
    {
        // we did not find a size
        #ResultSize = 0
    }
}
else if (#MethodReturn == 2)
{ // call GetSupportedSizeRange
    #MethodReturn =
    InvokeMethod(
        $PooltoDrawFrom->,
        "GetSupportedSizeRange",
        %InArguments,
        %OutArguments)
```

```

if(#MethodReturn != 1 && #MethodReturn != 2)
{
    // this method is supported
    #MaximumVolumeSize = %OutArguments["MaximumVolumeSize"]
    #MinimumVolumeSize = %OutArguments["MinimumVolumeSize"]
    #VolumeSizeDivisor = %OutArguments["VolumeSizeDivisor"]
    if(#RequestedSize >= #MinimumVolumeSize &&
        #RequestedSize <= #MaximumVolumeSize)
    {
        // Rounding up to next Size, which is dividable by Divisor
        #ResultSize = (#RequestedSize + (#VolumeSizeDivisor -
            (#RequestedSize MOD #VolumeSizeDivisor)))
    }
}
return #ResultSize
}

// MAIN
// Step 1. Get the configuration services and determine the service
// capabilities. Note that the device may not support storage
// configuration so it is possible that the service is not present and
// the desired management cannot be performed.
try {
    $Services->[] = AssociatorNames($BlockServer->,
        "CIM_HostedService",
        "CIM_StorageConfigurationService",
        null,
        null)

    // StorageConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($Services->[] == null) {
        <EXIT: Storage Configuration is not supported.>
    }
} catch (CIMException $Exception) {
    // StorageConfigurationService and/or HostedService may not be included in
    // the model implemented at all if Storage Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
        <EXIT: Storage Configuration is not supported.>
    }
}

// There should be only one storage configuration service
// Associated with the system
$StorageConfigurationService-> = $Services->[0]
$ServiceCapabilities[] = Associators(
    $StorageConfigurationService->,

```

```

    "CIM_ElementCapabilities",
    "CIM_StorageConfigurationCapabilities",
    null,
    null,
    false,
    false,
    null)

// There should be only one StorageConfigurationCapabilities instance
#SupportsPoolCreation = contains(
    2, // Storage Pool Creation
    $ServiceCapabilities[0].SupportedSynchronousActions[] ||
    contains(
        2, // Storage Pool Creation
        $ServiceCapabilities[0].SupportedAsynchronousActions[]))
#PoolCreationProducesJob = contains(
    2, // Storage Pool Creation
    $ServiceCapabilities[0].SupportedAsynchronousActions[])
#SupportsElementCreation1 = contains(
    5, // Storage Element Creation
    $ServiceCapabilities[0].SupportedSynchronousActions[])
#SupportsElementCreation2 = contains(
    3, // StorageElementCreation
    $ServiceCapabilities[0].SupportedStorageElementFeatures[])
#ElementCreationProducesJob = contains(
    5, // Storage Element Creation
    $ServiceCapabilities[0].SupportedAsynchronousActions[])
// If a storage element can not be created and that storage element is
// neither created synchronously or asynchronously, then fail the test
if (!$SupportedElementCreation2 &&
    !($SupportedElementCreation1 || #ElementCreationProducesJob))
{
    <EXIT: The StoragePool can be created, but the
        StorageElement creation is not supported.>
}

// Step 2. Enumerate over the CIM_HostedStoragePool associations to find
// all the StoragePools from which storage elements might be created.
$StoragePools[] = Associators(
    $BlockServer->,
    "CIM_HostedStoragePool",
    "CIM_StoragePool",
    null,
    null,
    false,
    false,
    {"InstanceID", "Primordial"})

```

```

// Step 3. For each StoragePool, follow the CIM_ElementCapabilities
//  asociation to the StorageCapabilities of that pool. Compare the
//  StorageCapabilities to the desired StorageSetting and find the
//  best match.
$PoolToDrawFrom-> = null
for #i in $StoragePools[]
{
    // If we can not create Storage Pool, then find a 'concrete'
    // Storage Pool from which to create a Storage Element
    #UsePrimordial = false
    if(#SupportsPoolCreation)
    {
        #UsePrimordial = true
        #RequestedElementType = 2 // StoragePool
    }
    else
    {
        #RequestedElementType = #ElementType
    }
    if ($StoragePools[#i].Primordial == #UsePrimordial)
    {
        $CapabilitiesOffered[] = Associators(
            $StoragePools[#i].getObjectPath(),
            "CIM_ElementCapabilities",
            "CIM_StorageCapabilities",
            null,
            null,
            false,
            false,
            null)
        $StorageCapabilitiesOffered = &GetMostCapable($CapabilitiesOffered[])
        $PoolToDrawFrom-> = $StoragePool[#i].getObjectPath()

// Step 4. Determine if the selected pool has enough space for
//  another pool.
//  If the block server supports hints, then the Storage Setting returned
//  will contain default hints

        // Create a setting
        %InArguments["SettingType"] = 3 // Goal
        #ReturnValue = InvokeMethod(
            $StorageCapabilitiesOffered.getObjectPath(),
            "CreateSetting",
            %InArguments,
            %OutArguments)
        if (#ReturnValue != 0 || null)

```

Block Services Package

```

{
    <ERROR! Unable to create storage setting >
}
$GeneratedStorageSetting-> = %OutArguments["NewSetting"]

// Determine the possible size, closest to the requested size
#PossibleSize = &PoolSizeAvailable(
    $PoolToDrawFrom->,
    $GeneratedStorageSetting->,
    #RequestedSize
    #RequestedElementType)
if(0 != #PossibleSize) // we found a size close to #RequestedSize
{
    }
    break;
}
else
{
    // Cause failure if there are no more candidate Pools
    $PoolToDrawFrom-> = NULL;
}
}
}
if ($PoolToDrawFrom-> == NULL)
{
    <ERROR! Unable to find a suitable pool from which to create the storage
        element >
}

// Step 5. Register for indications on configuration jobs
If(#PoolCreationProducesJob || #ElementCreateProducesJob)
{
    // '17' ("Completed") '2' ("OK")
    #Filter1 = <the filter from Job Control subprofile with the description
    "Modification of Operational Status for a Concrete Job to 'Complete' and 'OK'">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter1)

    // '17' ("Completed") '6' ("Error")
    #Filter2 = <the filter from Job Control subprofile with the description
    "Modification of Operational Status for a Concrete Job to 'Complete'
    and 'Error'">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter2)
}

// Step 6. Create the Storage Pool
if(#SupportsPoolCreation)
{

```


Block Services Package

```
%InArguments["ElementName"] = NULL// we do not care what
// the name is
%InArguments["Goal"] = $GeneratedStorageSetting->
%InArguments["Size"] = #PossibleSize
%InArguments["InExtents"] = null
%InArguments["Pool"] = null
%InArguments["InPools"] = $PoolToDrawFrom->
#ReturnValue = InvokeMethod(
    $StorageConfigurationService->,
    "CreateOrModifyStoragePool",
    %InArguments, %OutArguments)
if(#ReturnValue != 0 && #ReturnValue != 4096)
{
    // Storage Pool was not created
    <ERROR! Failed >
}
$PoolToDrawFrom-> = %OutArguments["Pool"]
$PoolCreationJob-> = %OutArguments["Job"]

if(#PoolCreationProducesJob && $PoolCreationJob-> != null)
{
    <Wait until the completion of the job
        using $PoolCreationJob-> as a filter>

    <Wait for indication from either filters defined in step 5
        If the indication states the Job is 'Complete' and 'Error'
        then exit with error
        ERROR! Job did not complete successfully
    >
}
$CapabilitiesOffered[] = Associators(
    $PoolToDrawFrom->,
    "CIM_ElementCapabilities",
    "CIM_StorageCapabilities",
    null,
    null,
    false,
    false,
    null)
$StorageCapabilitiesOffered = $CapabilitiesOffered[0]
}

// Step 7. Create Storage Element.
%InArguments["SettingType"] = 3 // "Goal"
#ReturnValue = InvokeMethod(
    $StorageCapabilitiesOffered.getObjectPath(),
    "CreateSetting",
    %InArguments,
```

Block Services Package

```

        %OutArguments)
if (#ReturnValue != 0)
{
    <ERROR! Unable to create storage setting >
}
$GeneratedStorageSetting-> = %OutArguments["NewSetting"]

%InArguments["ElementName"] = NULL
%InArguments["ElementType"] = #ElementType
%InArguments["Goal"] = $GeneratedStorageSetting->
%InArguments["Size"] = #PossibleSize
$InArguments["InPool"] = $PoolToDrawFrom->
%InArguments["TheElement"] = null
#ReturnValue = InvokeMethod(
    $StorageConfigurationService->,
    "CreateOrModifyElementFromStoragePool",
    %InArguments, %OutArguments)
if(#ReturnValue != 0 || #ReturnValue != 4096)
{
    // Method did not succeeded or succeeded but did not create a job
    <ERROR! Failed >
}
else if(#ReturnValue == 0 ||
    (#ReturnValue == 4096 && %OutArguments["TheElement"] != null))
{
    $CreatedElement-> = %OutArguments["TheElement"]
}
else // a Job was created and TheElement is null
{
    <Wait for indication from either filters defined in step 5
    If the indication states the Job is 'Complete' and 'Error'
    then exit with error
    ERROR! Job did not complete successfully
    >

    <Once the 'Job' has completed successfully, see step 5, then
    follow the AffectedJobElement association from the 'Job' to
    retrieve the storage element that was created.>
    $CreateElements[] = Associators(
        $Job->, // Object Name coersed from %OutArguments["Job"]
        "CIM_AffectedJobElement",
        #StorageElementClass,
        null,
        null,
        false,
        false,
        null)
    // Only one storage element will be created,

```

```

    $CreatedElement-> = $CreatedElement[0].getObjectPath()
}

```

5.6.7 Expand Storage Element on Block Server

```

// DESCRIPTION
// In this recipe, we attempt to expand a LUN on an array by 50%.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.A reference to the CIM_ComputerSystem that represents the array
//   $BlockServer->
// 2.A reference to the particular storage element we wish to expand.
//   $ElementToExpand->
// 3.It is assumed that to expand a storage element there needs to be
//   enough space available in the parent StoragePool to contain
//   another copy of the storage element whose size is equal to the
//   new size requested. This is especially the case if we were
//   modifying the settings as well as the size.
// 4.#ElementClassName is set to the class name of the storage element be
//   modified.
//   (e.g. CIM_StorageVolume or CIM_LogicalDisk)
// 5. #ElementType is set to the storage element to modified
//   See CreateOrModifyElementFromStoragePool.ElementType

// Step 1. Get the configuration services and determine the service
// capabilities
// Step 1. Get the configuration services and determine the service
// capabilities. Note that the device may not support storage
// configuration so it is possible that the service is not present and
// the desired management cannot be performed.
try {
    $Services->[] = AssociatorNames($BlockServer->,
        "CIM_HostedService",
        "CIM_StorageConfigurationService",
        null,
        null)
    // StorageConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($Services->[] == null) {
        <EXIT: Storage Configuration is not supported.>
    }
} catch (CIMException $Exception) {
    // StorageConfigurationService and/or HostedService may not be included in
    // the model implemented at all if Storage Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
        <EXIT: Storage Configuration is not supported.>
    }
}
}

```

Block Services Package

```
// There should be only one storage configuration service
// Associated with the system
$StorageConfigurationService-> = $Services->[0]
$ServiceCapabilities[] = Associators(
    $BlockServer->,
    "CIM_ElementCapabilities",
    "CIM_StorageConfigurationCapabilities",
    null,
    null,
    false,
    false,
    null)

// There should be only one StorageConfigurationCapabilities instance
#SupportsElementModification1 = contains(
    7, // Storage Element Modification
    $ServiceCapabilities[0].SupportedSynchronousActions[] ||
    contains(
        7, // Storage Element Modification
        $ServiceCapabilities[0].SupportedAsynchronousActions[])
#SupportsElementModification2 = contains(
    5, // Storage Element Modification
    $ServiceCapabilities[0].SupportedStorageElementFeatures[])
#ElementModificationProducesJob = contains(
    7, // Storage Element Modification
    $ServiceCapabilities[0].SupportedAsynchronousActions[])
if(!#SupportedElementModification1 || !#SupportedElementModification2)
{
    <EXIT: The ability to modify an existing Storage Element must be supported
    to continue.>
}

// Step 2. Read the current size of the Storage Element.
$StorageElement = GetInstance(
    $ElementToExpand->,
    false,
    false,
    false,
    {"BlockSize", "NumberOfBlocks"})
#PreviousSize = $StorageElement.BlockSize * $StorageElement.NumberOfBlocks

// Step 3. Follow the AllocatedFromStoragePool association from the
// storage element to find the pool from whence it came.
$Pools->[] = AssociatorNames(
    $ElementToExpand->,
    "CIM_AllocatedFromStoragePool",
```

Block Services Package

```
"CIM_StoragePool",
null,
null)

// A Storage Element has only one Pool parent
$ParentPool-> = $Pools->[0]

// Step 4. Determine whether the desired space for which to expand the
// storage element exists within the pool.
$StorageSetting->[] = AssociatorNames(
    $ElementToExpand->,
    "CIM_ElementSettingData",
    "CIM_StorageSetting",
    null,
    null)
$CurrentElementSetting-> = $StorageSetting->[0]
// Calculate the additional space needed
#SizeToExpand = 0.5 * #PreviousSize
// Calculate 150% of previous storage element size
#SizeToExpandTo = #PreviousSize + (0.5 * #PreviousSize)
#NewSizeAvailable =
    @<Create Storage Pool and Storage Element on Block Server>
        &PoolSizeAvailable(
            $ParentPool->,
            $CurrentElementSetting->,
            #SizeToExpand,
            #ElementType)
if (0 == #NewSizeAvailable)
{
    <ERROR! Unable to proceed because the requested size is unavailable >
}

// Step 5. Register for indications on configuration jobs
If(#ElementModificationProducesJob)
{
    // '17' ("Completed") '2' ("OK")
    #Filter1 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete'
and 'OK">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter1)

    // '17' ("Completed") '6' ("Error")
    #Filter2 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete'
and 'Error'">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter2)
```

```

}

// Step 6. Modify the Storage Element
// If there is a Job produced, wait for Job completion
%InArguments["ElementName"] = null// we do not care what the name is
%InArguments["ElementType"] = #ElementType
%InArguments["Goal"] = $CurrentElementSetting
%InArguments["Size"] = #SizeToExpandTo
%InArguments["InPool"] = $ParentPool->
%InArguments["TheElement"] = $ElementToExpand->
#ReturnValue = InvokeMethod(
    $StorageConfigurationService->
    "CreateOrModifyElementFromStoragePool"
    %InArguments
    %OutArgument
)
if(#ReturnValue != 0 && #ReturnValue != 4096)
{
    // Method succeeded or validated arguments and started a job
    <ERROR! Failed >
}
else if(#ReturnValue == 0)
{
    $CreatedElement-> = %OutArguments["TheElement"]
}
else // a Job was created and TheElement is null
{
    <Wait for indication from either filters defined in step 5
    If the indication states the Job is 'Complete' and 'Error'
    then exit with error
    ERROR! Job did not complete successfully
    >

    <Once the 'Job' has stopped, see step 4, then follow the
    AffectedJobElement association from the 'Job' to retrieve
    the storage element that was created.>
    $CreateElements[] = Associators(
        $Job->, // Object Name coerced from %OutArguments["Job"]
        "CIM_AffectedJobElement",
        #ElementClassName,
        null,
        null,
        false,
        false,
        null)

    // Only one Storage Element will be created,
    $CreatedElement-> = $CreateElements[0].getObjectPath()
}

```

```

// Step 7. Check the value of the "Size" out parameter. See if it is
// equal to size expected. If so, we got what we asked for and we're done.
#SizeExpandedTo = %OutArguments["Size"]
if (#SizeExpandedTo == #SizeToExpandTo)
{
    < indicate the storage element was successfully expanded >
}
else
{
    if (#SizeExpandedTo <= #PreviousSize)
    {
        < indicate the storage element was not expanded >
    }
    else
    {
        < indicate the storage element was only partially expanded to
            #SizeExpandedTo >
    }
}
}

```

5.6.8 Create Storage Element from Elements on Block Server

```

// DESCRIPTION
// The goal of this recipe is to create a storage element with the maximum
// capabilities of the block server. If supported, the pool creation specifies
// the disk(s) to use as input rather than the size.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.A reference to a CIM_ComputerSystem Host is previously
// defined in the $Host-> variable
// 2. The references for input disks that are to be used for creating the pool
// are in $DisksForPool->[] array. All these must be associated to the
// primordial pool with CIM_ConcreteComponent association.
// On being transferred to a Concrete pool they will be disassociated from
// the primordial pool.
// 3. The storage element will be created using available disks in the
// concrete returned by GetAvailableExtents.
// 4.The settings for the new Storage Pool and Logical Disk are defined in
// the following variables:
// #RequestedSize = 10 * 1024 * 1024 * 1024 // 10 GB
// 5.#StorageElementClass is set to the class name of the element being
// created like CIM_StorageVolume or CIM_LogicalDisk.
// 6. #ElementType is set to the element to created
// 2 - StorageVolume
// 4 - LogicalDisk
// See CreateOrModifyElementFromStoragePool.ElementType

```

Block Services Package

```
// MAIN
// Step 1. Get the configuration services and determine the service
// capabilities. Note that the device may not support storage
// configuration so it is possible that the service is not present and
// the desired management cannot be performed.
try {
    $Services->[] = AssociatorNames($Host->,
        "CIM_HostedService",
        "CIM_StorageConfigurationService",
        null,
        null)

    // StorageConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($Services->[] == null) {
        <EXIT: Storage Configuration is not supported.>
    }
} catch (CIMException $Exception) {
    // StorageConfigurationService and/or HostedService may not be included in
    // the model implemented at all if Storage Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
        <EXIT: Storage Configuration is not supported.>
    }
}

// There should be only one storage configuration service
// Associated with the system
$StorageConfigurationService-> = $Services->[0]
$ServiceCapabilities[] = Associators($StorageConfigurationService->,
    "CIM_ElementCapabilities",
    "CIM_StorageConfigurationCapabilities",
    null,
    null,
    false,
    false,
    null)

// There should be only one StorageConfigurationCapabilities instance
#SupportsPoolCreation = contains(2, // Storage Pool Creation
    $ServiceCapabilities[0].SupportedSynchronousActions[]
    || contains(2, // Storage Pool Creation
    $ServiceCapabilities[0].SupportedAsynchronousActions[]))
#PoolCreationProducesJob = contains(2, // Storage Pool Creation
    $ServiceCapabilities[0].SupportedAsynchronousActions[])
#SupportsElementCreation1 = contains(11, // Storage Element from Element Creation
    $ServiceCapabilities[0].SupportedSynchronousActions[])
#SupportsElementCreation2 = contains(3, // LogicalDiskCreation
    $ServiceCapabilities[0].SupportedStorageElementFeatures[])
```


Block Services Package

```
#ElementCreationProducesJob = contains(11, // Storage Element from Element
                                      Creation
                                      $ServiceCapabilities[0].SupportedAsynchronousActions[])
#SupportsInExtents = contains(2, // InExtents
                              $ServiceCapabilities[0].SupportedStoragePoolFeatures[])

// If StorageExtent creation is not supported, the set of specific disks from
// which to allocate the StoragePool is not supported by the device.
if (!#SupportsInExtents) {
    <EXIT: The StoragePool cannot be created from a specific set of disks.>
}

// If a storage element can not be created and that storage element is
// neither created synchronously or asynchronously, then fail the test
if (!#SupportedElementCreation2 &&
    !(#SupportedElementCreation1 || #ElementCreationProducesJob)) {
    <EXIT: The StoragePool can be created, but the
        storage element from element creation is not supported.>
}

// Step 2. Enumerate over the CIM_HostedStoragePool associations to find
// all the StoragePools from which storage elements might be created.
$StoragePools[] = Associators($Host->,
    "CIM_HostedStoragePool",
    "CIM_StoragePool",
    null,
    null,
    false,
    false,
    {"InstanceID", "Primordial"})

// Step 3. For each StoragePool, follow the CIM_ElementCapabilities
// association to the StorageCapabilities of that pool. Compare the
// StorageCapabilities to the desired StorageSetting and find the
// best match.
$PoolToDrawFrom-> = null
for (#i in $StoragePools[]) {
    // If we can not create Storage Pool, then find a 'concrete'
    // Storage Pool from which to create a Storage Element
    #UsePrimordial = false
    if (#SupportsPoolCreation) {
        #UsePrimordial = true
        #RequestedElementType = 2 // StoragePool
    } else {
        #RequestedElementType = #ElementType
    }
    if ($StoragePools[#i].Primordial == #UsePrimordial) {
        $CapabilitiesOffered[] = Associators(
```

Block Services Package

```
$StoragePools[#i].getObjectPath(),
"CIM_ElementCapabilities",
"CIM_StorageCapabilities",
null,
null,
false,
false,
null)
$StorageCapabilitiesOffered = &GetMostCapable($CapabilitiesOffered[])
$PoolToDrawFrom-> = $StoragePool[#i].getObjectPath()

// Step 4. Determine if the selected pool has enough space for
// another pool. If the block server supports hints, then
// the StorageSetting returned will contain default hints
// Create a setting
%InArguments["SettingType"] = 3 // Goal
#ReturnValue = InvokeMethod(
    $StorageCapabilitiesOffered.getObjectPath(),
    "CreateSetting",
    %InArguments,
    %OutArguments)
if (#ReturnValue != 0 || null) {
    <ERROR! Unable to create storage setting >
}
$GeneratedStorageSetting-> = %OutArguments["NewSetting"]

// Determine the possible size, closest to the requested size
#PossibleSize = &PoolSizeAvailable(
    $PoolToDrawFrom->,
    $GeneratedStorageSetting->,
    #RequestedSize,
    #RequestedElementType)
if (0 != #PossibleSize) {
    // Located a size close to #RequestedSize
    break;
} else {
    // Cause failure if there are no more candidate Pools
    $PoolToDrawFrom-> = NULL;
}
}
}
if ($PoolToDrawFrom-> == NULL) {
    <ERROR! Unable to find a suitable pool from which to create the storage
        element>
}

// Step 5. Register for indications on configuration jobs
if (#PoolCreationProducesJob || #ElementCreateProducesJob) {
```

```

// '17' ("Completed") '2' ("OK")
#Filter1 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete'
and 'OK">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter1)

// '17' ("Completed") '6' ("Error")
#Filter2 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete'
and 'Error'">
    @{Determine if Indications already exist or have to be
        created}&createIndication(#Filter2)
}

// Step 6. Create the Storage Pool
if (#SupportsPoolCreation) {
    %InArguments["ElementName"] = NULL// leave up to the device
    %InArguments["Goal"] = $GeneratedStorageSetting->
    %InArguments["Size"] = null
    %InArguments["InExtents"] = $DisksForPool->[]
    %InArguments["Pool"] = null
    $InPools->[0] = $PoolToDrawFrom->
    %InArguments["InPools"] = $InPools->[]
    #ReturnValue = InvokeMethod($StorageConfigurationService->,
        "CreateOrModifyStoragePool",
        %InArguments, %OutArguments)
    if (#ReturnValue != 0 && #ReturnValue != 4096) {
        // Storage Pool was not created
        <ERROR! Failed>
    }
    $PoolToDrawFrom-> = %OutArguments["Pool"]
    $PoolCreationJob-> = %OutArguments["Job"]

    if (#PoolCreationProducesJob && $PoolCreationJob-> != null) {
        <Wait until the completion of the job
            using $PoolCreationJob-> as a filter>

        <Wait for indication from either filters defined in step 5
            If the indication states the Job is 'Complete' and 'Error'
            then exit with error
            ERROR! Job did not complete successfully>
    }
    $CapabilitiesOffered[] = Associators($PoolToDrawFrom->,
        "CIM_ElementCapabilities",
        "CIM_StorageCapabilities",
        null,
        null,

```

Block Services Package

```
        false,
        false,
        null)
    $StorageCapabilitiesOffered = $CapabilitiesOffered[0]
}

// Step 7. Call GetAvailableExtents to find available extents for creating
// the storage element.
%InArguments["Goal"] = $GeneratedStorageSetting->
#ReturnValue = InvokeMethod($PoolToDrawFrom->,
    "GetAvailableExtents",
    %InArguments, %OutArguments)
if (#ReturnValue != 1) {
    // Not supported
    <EXIT! Method not supported, can not finish this recipe>
} else if (#ReturnValue != 0) {
    // Method did not succeeded or succeeded but did not create a job
    <ERROR! Failed>
}
$DisksForElement->[] = %OutArguments["AvailableExtents"]

// Step 8. Create Storage Element
%InArguments["SettingType"] = 3 // "Goal"
InvokeMethod($StorageCapabilitiesOffered.getObjectPath(),
    "CreateSetting",
    %InArguments,
    %OutArguments)
if (#ReturnValue != 0) {
    <ERROR! Unable to create storage setting >
}
$GeneratedStorageSetting-> = %OutArguments["NewSetting"]

%InArguments["ElementName"] = NULL
%InArguments["ElementType"] = #ElementType
%InArguments["Goal"] = $GeneratedStorageSetting->
%InArguments["Size"] = #PossibleSize
$InPools->[0] = $PoolToDrawFrom->
%InArguments["InPool"] = $InPools->
%InArguments["InElements"] = $DisksForElement->[]
%InArguments["TheElement"] = null // Create new element
#ReturnValue = InvokeMethod($StorageConfigurationService->,
    "CreateOrModifyElementFromElements",
    %InArguments, %OutArguments)
if (#ReturnValue != 0 && #ReturnValue != 4096) {
    // Method did not succeeded or succeeded but did not create a job
    <ERROR! Failed>
} else if (#ReturnValue == 0 ||
```

```

        (#ReturnValue == 4096 && %OutArguments["TheElement"] != null))) {
        $CreatedElement-> = %OutArguments["TheElement"]
    } else // a Job was created and TheElement is null {
        <Wait for indication from either filters defined in step 5
        If the indication states the Job is 'Complete' and 'Error'
        then exit with error
        ERROR! Job did not complete successfully>

        <Once the 'Job' has completed, see step 5, then follow the
        AffectedJobElement association from the 'Job' to retrieve
        the storage element that was created.>
        $CreateElements[] = Associators(
            $Job->, // Object Name coersed from %OutArguments["Job"]
            "CIM_AffectedJobElement",
            #StorageElementClass,
            null,
            null,
            false,
            false,
            null)
        // Only one LogicalDisk will be created,
        $CreatedElement-> = $CreateElements[0].getObjectPath()
    }
}

```

5.6.9 Optional RECIPE: Intentionally General a CIM Error

```

// DESCRIPTION
// Validate reporting an error/exception
// when InvokeMethod is called with an invalid parameter.
//
// This recipe intentionally supplies an invalid "ElementType".
//
// This recipe attempts to optionally utilize properties of CIM_Error
// if CIM_Error is implemented.

// 1. Insert an error
// 2. Catch the exception
// 3. Report the error

// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.A reference to a storage setting is previously defined
//     in the $StorageSetting-> variable.
// 2.A size that is possible for the creation of a storage element
//     is provided in the #PossibleSize,
// 3.A reference to Pool is previous defined in the $PoolToDrawFrom-> variable
// 4.A object paths for source input Pools is previous defined in the
//     $InPools variable
// 5. A reference to the StorageConfigurationService is already defined

```

Block Services Package

```
//      in the StorageConfiguratonServivce-> variable
//
%InArguments["ElementType"] = 1000 // Invalid ElementType
%InArguments["Goal"] = $StorageSetting->
%InArguments["Size"] = #PossibleSize
%InPools->[0] = $PoolToDrawFrom->
%InArguments["InPool"] = $InPools->
%InArguments["TheElement"] = null
try
{
    #ReturnValue = InvokeMethod(
        $StorageConfigurationService->,
        "CreateOrModifyElementFromStoragePool",
        %InArguments, %OutArguments)
}
catch (CIM Exception $Exception) {
    // For SMI-S 1.1, optionally allow for implementation of CIM_Error.
    if($Exception.MessageID <> null) { // CIM_Error is implemented
        // For example
        if($Exception.MessageArguments[2] ==
            "CreateOrModifyElementFromStoragePool") &&
            $Exception.MessageArguments[0] == "1" && // Second method parameter
            $Exception.MessageID = "MP5")
        {
            <EXIT: Success -- CIM_Error is constructed properly>
        }
        else {
            <ERROR! Improperly constructed CIM_Error>
        }
    }
    else {
        <display, optional CIM_Error is not implemented>
        if($Exception.CIMStatusCode != CIM_ERR_INVALID_PARAMETER) {
            <ERROR! Improper CIM status code returned>
        }
        else {
            <EXIT: Success -- correct CIM status code reported>
        }
    }
}

if (#ReturnValue != CIM_ERR_INVALID_PARAMETER) { // 5 = Invalid parameter
    <ERROR! Invalid return value >
}
```

5.7 Registered Name and Version

Block Services version 1.2.0

5.8 CIM Elements

Table 16: CIM Elements for Block Services

Element Name	Requirement	Description
CIM_StoragePool (Primordial) (5.8.1)	Mandatory	The primordial StoragePool. It is created by the provider and cannot be deleted or modified. It cannot be used to allocate any storage element other than concrete StoragePools.
CIM_StoragePool (Concrete) (5.8.2)	Mandatory	The concrete StoragePool. A concrete StoragePool shall be allocated from another StoragePool. It shall be used for allocating StorageVolumes and LogicalDisks as well as other concrete StoragePools.
CIM_HostedStoragePool (5.8.3)	Mandatory	
CIM_HostedService (5.8.4)	Optional	
CIM_StorageConfigurationCapabilities (5.8.5)	Optional	
CIM_StorageConfigurationService (5.8.6)	Optional	
CIM_OwningJobElement (5.8.7)	Conditional	Conditional requirement: Support for Job Control profile.
CIM_StorageCapabilities (5.8.8)	Mandatory	
CIM_StorageSetting (5.8.9)	Mandatory	
CIM_StorageSettingWithHints (5.8.10)	Optional	
CIM_ElementSettingData (5.8.11)	Mandatory	
CIM_ElementCapabilities (StorageCapabilities to StoragePool) (5.8.12)	Mandatory	Associates StorageCapabilities with StoragePool.
CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService) (5.8.13)	Mandatory	Associates StorageConfigurationCapabilities with StorageConfigurationService.
CIM_AllocatedFromStoragePool (Pool from Pool) (5.8.14)	Mandatory	AllocatedFromStoragePool
CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool) (5.8.15)	Conditional	AllocatedFromStoragePool
CIM_StorageVolume (5.8.16)	Conditional	Representation of a virtual disk (for SCSI, a logical unit). A StorageVolume is allocated from a concrete StoragePool. See the "Standard Formats for Logical Unit Names" section in the Storage Management Technical Specification, Part 1 Common Architecture for details on how to set Name, NameFormat, and NameNamespace properties.
CIM_SystemDevice (System to StorageVolume or LogicalDisk) (5.8.17)	Mandatory	Associates top level system from Array, Virtualizer, ... to StorageVolume or LogicalDisk

Table 16: CIM Elements for Block Services

Element Name	Requirement	Description
CIM_BasedOn (StorageVolume to extent from Extent Composition) (5.8.18)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_LogicalDisk (5.8.19)	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. A LogicalDisk is allocated from a concrete StoragePool.
CIM_BasedOn (LogicalDisk to extent from Extent Composition) (5.8.20)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_StorageSettingsAssociatedToCapabilities (5.8.21)	Optional	This class associates the StorageCapabilities with the preset setting. Any StorageSetting instance associated with this association shall work, unmodified, to create a storage element. The preset settings should not change overtime and represent possible settings for storage elements are set of design time rather than runtime. All StorageSetting instances linked with this association shall have a ChangeableType of "0" ("Fixed - Not Changeable").
CIM_StorageSettingsGeneratedFromCapabilities (5.8.22)	Mandatory	This class associates the StorageCapabilities with the StorageSetting generated from it via the CreateSetting method. StorageSettings instances generated in this manner, as identified with this association, may be removed from the model at any time by the implementation if the ChangeableType of the associated setting is set to "2" ("Changeable - Transient"). All StorageSettings associated with this class shall be changeable, ChangeableType is "2" or "3". Some implementations may permit the modification of the ChangeableType property itself on StorageSetting instances associated via this class. Provided this is allowed, an client may change the ChangeableType to "3" ("Changeable - Persistent") to have this setting retained either after generation of the instance or after its modification by the client. The DefaultSetting property of the StorageSetting instances linked with this association is meaningless.
CIM_ConcreteComponent (storage element to extent from Extent Composition) (5.8.23)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_AssociatedComponentExtent (StoragePool to extent from Composition) (5.8.24)	Conditional	Conditional requirement: Support for Extent Composition profile.

Table 16: CIM Elements for Block Services

Element Name	Requirement	Description
CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition) (5.8.25)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_EnabledLogicalElementCapabilities (5.8.26)	Optional	This class is used to express the naming and possible requested state change possibilities for storage elements.
CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool) (5.8.27)	Optional	Associates EnabledLogicalElementCapabilities with StorageConfigurationService.
CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk) (5.8.28)	Optional	Associates EnabledLogicalElementCapabilities with StorageConfigurationService.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool) (5.8.29)	Optional	Expressed the ability for the element to be named or have its state changed.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk) (5.8.30)	Optional	Expressed the ability for the element to be named or have its state changed.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Creation/Deletion of StoragePool
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Deletion of StoragePool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Creation of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Deletion of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Deprecated WQL - Change of status of a Storage Volume, if Storage Volume is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.CIM_StorageVolume::OperationalStatus <> PreviousInstance.CIM_StorageVolume::OperationalStatus	Optional	Experimental CQL - Change of status of a Storage Volume, if Storage Volume is implemented.

Table 16: CIM Elements for Block Services

Element Name	Requirement	Description
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Creation of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deletion of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deprecated WQL - Change of status of LogicalDisk, if LogicalDisk is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Experimental CQL - Change of status of LogicalDisk, if LogicalDisk is implemented.

5.8.1 CIM_StoragePool (Primordial)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 17 describes class CIM_StoragePool (Primordial).

Table 17: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	Shall be true.
InstanceID		Mandatory	
ElementName		Optional	
PoolID		Mandatory	A unique name in the context of this system that identifies this Pool.
TotalManagedSpace		Mandatory	

Table 17: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)

Properties	Flags	Requirement	Description & Notes
RemainingManagedSpace		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
GetSupportedSizes()		Mandatory	List the discrete storage element sizes that can be created or expanded from this Pool.
GetSupportedSizeRange()		Mandatory	List the size ranges for storage element that can be created or expanded from this Pool.
GetAvailableExtents()		Optional	List the StorageExtents from this Pool that may be used to create or expand a storage element. The StorageExtents may not already be in use as supporting capacity for existing storage element.

5.8.2 CIM_StoragePool (Concrete)

Created By: Extrinsic: StorageConfigurationService.CreateOrModifyStoragePool

Modified By: Extrinsic: StorageConfigurationService.CreateOrModifyStoragePool

Deleted By: Extrinsic: StorageConfigurationService.DeleteStoragePool

Class Mandatory: Mandatory

Table 18 describes class CIM_StoragePool (Concrete).

Table 18: SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	Shall be false.
InstanceID		Mandatory	
ElementName		Optional	
PoolID		Mandatory	A unique name in the context of this system that identifies this Pool.
TotalManagedSpace		Mandatory	
RemainingManagedSpace		Mandatory	
Usage		Optional	The specialized usage intended for this element.

Table 18: SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)

Properties	Flags	Requirement	Description & Notes
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
GetSupportedSizes()		Mandatory	List the discrete storage element sizes that can be created or expanded from this Pool.
GetSupportedSizeRange()		Mandatory	List the size ranges for storage element that can be created or expanded from this Pool.
GetAvailableExtents()		Optional	List the StorageExtents from this Pool that may be used to create or expand a storage element. The StorageExtents may not already be in use as supporting capacity for existing storage element.

5.8.3 CIM_HostedStoragePool

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 19 describes class CIM_HostedStoragePool.

Table 19: SMI Referenced Properties/Methods for CIM_HostedStoragePool

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

5.8.4 CIM_HostedService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 20 describes class CIM_HostedService.

Table 20: SMI Referenced Properties/Methods for CIM_HostedService

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	The Service hosted on the System.

5.8.5 CIM_StorageConfigurationCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 21 describes class CIM_StorageConfigurationCapabilities.

Table 21: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
SupportedStoragePoolFeatures		Mandatory	Lists what StorageConfigurationService functionalities are implemented. Matches 2 3 5 6 7 (InExtents or Single InPool or Storage Pool QoS Change or Storage Pool Capacity Expansion or Storage Pool Capacity Reduction).
SupportedSynchronousActions		Optional	Lists what actions, invoked through StorageConfigurationService methods, shall not produce Concrete jobs.
SupportedStorageElementTypes		Mandatory	Lists the type of storage elements that are supported by this implementation.
SupportedAsynchronousActions		Optional	Lists what actions, invoked through StorageConfigurationService methods, may produce Concrete jobs.
SupportedStorageElementFeatures		Mandatory	Lists actions supported through the invocation of StorageServiceService.CreateOrModifyElementFromStoragePool(). Matches 3 5 8 9 11 12 13 (StorageVolume Creation or StorageVolume Modification or LogicalDisk Creation or LogicalDisk Modification or Storage Element QoS Change or Storage Element Capacity Expansion or Storage Element Capacity Reduction).
SupportedStorageElementUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on supported storage elements.

Table 21: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
ClientSettableElementUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on the usage of client-settable elements.
SupportedStoragePoolUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on storage pools.
ClientSettablePoolUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on the usage of a client-settable storage pool.

5.8.6 CIM_StorageConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 22 describes class CIM_StorageConfigurationService.

Table 22: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
CreationClassName		Mandatory	
SystemName		Mandatory	
Name		Mandatory	
CreateOrModifyStoragePool()		Mandatory	Create (or modify) a StoragePool. A job may be created as well.
DeleteStoragePool()		Mandatory	Start a job to delete a StoragePool.
CreateOrModifyElementFromStoragePool()		Mandatory	Create or modify a storage element. A job may be created as well.
CreateOrModifyElementFromElements()		Optional	Create or modify a storage element using component StorageExtents of the Pool. A job may be created as well.
ReturnToStoragePool()		Mandatory	Release the capacity represented by this storage element back to the Pool.
RequestUsageChange()		Optional	Allows a client to change the Usage for the element.

Table 22: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Properties	Flags	Requirement	Description & Notes
GetElementsBasedOnUsage()		Optional	Allows a client to retrieve elements for a specialized Usage.

5.8.7 CIM_OwningJobElement

Conditional on support for Job Control profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: JobControl

Table 23 describes class CIM_OwningJobElement.

Table 23: SMI Referenced Properties/Methods for CIM_OwningJobElement

Properties	Flags	Requirement	Description & Notes
OwnedElement		Mandatory	
OwningElement		Mandatory	

5.8.8 CIM_StorageCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 24 describes class CIM_StorageCapabilities.

Table 24: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	The user-friendly name for this instance of Capabilities. In addition, the user friendly name can be used as a index property for a search or query. (Note: ElementName does not have to be unique within a namespace) If the capabilities are fixed, then this property should be used as a means for the client application to correlate between capabilities and device documentation.

Table 24: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
ElementType		Mandatory	Enumeration indicating the type of instance to which this StorageCapabilities applies. Shall be either 5 or 6 (StoragePool or StorageConfigurationService).
NoSinglePointOfFailure		Mandatory	Indicates whether or not the associated instance supports no single point of failure. Values are: FALSE = does not support no single point of failure, and TRUE = supports no single point of failure.
NoSinglePointOfFailureDefault		Mandatory	Indicates the default value for the NoSinglePointOfFailure property.
DataRedundancyMin		Mandatory	DataRedundancyMin describes the minimum number of complete copies of data that can be maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. Possible values are 1 to n.
DataRedundancyMax		Mandatory	DataRedundancyMax describes the maximum number of complete copies of data that can be maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. Possible values are 1 to n.
DataRedundancyDefault		Mandatory	DataRedundancyDefault describes the default number of complete copies of data that can be maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. Possible values are 1 to n.
PackageRedundancyMin		Mandatory	PackageRedundancyMin describes the minimum number of spindles or logical devices that can be used. Package redundancy describes how many disk spindles or logical devices can fail without data loss including, at most, one spare. Examples would be RAID5 with a Package Redundancy of 1, RAID6 with 2. Possible values are 0 to n.
PackageRedundancyMax		Mandatory	PackageRedundancyMax describes the maximum number of spindles or logical devices that can be used. Package redundancy describes how many disk spindles or logical devices can fail without data loss including, at most, one spare. Examples would be RAID5 with a Package Redundancy of 1, RAID6 with 2. Possible values are 0 to n.
PackageRedundancyDefault		Mandatory	PackageRedundancyDefault describes the default number of spindles or logical devices that can be used. Package redundancy describes how many disk spindles or logical devices can fail without data loss including, at most, one spare. Examples would be RAID5 with a Package Redundancy of 1, RAID6 with 2. Possible values are 0 to n.

Table 24: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
ExtentStripeLengthDefault		Optional	Describes what the default stripe length, the number of members or columns, a storage element will have when created or modified using this capabilities. A NULL means that the setting of stripe length is not supported at all or not supported at this level of storage element allocation or assignment.
ParityLayoutDefault		Optional	ParityLayoutDefault describes what the default parity a storage element will have when created or modified using this capabilities. A NULL means that the setting of the parity is not supported at all or is not supported at this level of storage element allocation or assignment
UserDataStripeDepthDefault		Optional	UserDataStripeDepthDefault describes what the number of bytes forming a stripe that a storage element will have when created or modified using this capabilities. A NULL means that the setting of stripe depth is not supported at all or not supported at this level of storage element allocation or assignment.
CreateSetting()		Mandatory	Generate a setting to use as a goal for creating or modifying storage elements.
GetSupportedStripeLengths()		Optional	List the possible discrete stripe lengths supported at this time of this method's execution.
GetSupportedStripeLengthRange()		Optional	List the possible stripe length ranges supported at the time of this method's execution
GetSupportedParityLayouts()		Optional	List the possible parity layouts supported at the time of this method's execution.
GetSupportedStripeDepths()		Optional	List the possible stripe depths supported at the time of this method's execution.
GetSupportedStripeDepthRange()		Optional	List the possible strip depth ranges supported at the time of this method's execution.

5.8.9 CIM_StorageSetting

Created By: Extrinsic: StorageCapabilities.CreateSetting

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 25 describes class CIM_StorageSetting.

Table 25: SMI Referenced Properties/Methods for CIM_StorageSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	The user-friendly name for this instance of SettingData. In addition, the user friendly name can be used as a index property for a search of query. (Note: Name does not have to be unique within a namespace.)
NoSinglePointOfFailure		Mandatory	Indicates the desired value for No Single Point of Failure. Possible values are false = single point of failure, and true = no single point of failure.
DataRedundancyMin		Mandatory	DataRedundancyMin describes the minimum number of complete copies of data to be maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. Possible values are 1 to n.
DataRedundancyMax		Mandatory	DataRedundancyMax describes the maximum number of complete copies of data to be maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. Possible values are 1 to n.
DataRedundancyGoal		Mandatory	
PackageRedundancyMin		Mandatory	PackageRedundancyMin describes the minimum number of spindles or logical devices to be used. Package redundancy describes how many disk spindles or logical devices can fail without data loss including, at most, one spare. Examples would be RAID5 with a Package Redundancy of 1, RAID6 with 2. Possible values are 0 to n.
PackageRedundancyMax		Mandatory	PackageRedundancyMax describes the maximum number of spindles or logical devices to be used. Package redundancy describes how many disk spindles or logical devices can fail without data loss including, at most, one spare. Examples would be RAID5 with a Package Redundancy of 1, RAID6 with 2. Possible values are 0 to n.
PackageRedundancyGoal		Mandatory	
ExtentStripeLength		Optional	ExtentStripeLength describes the desired stripe length goal.
ExtentStripeLengthMin		Optional	ExtentStripeLengthMin describes the minimum acceptable stripe length.
ExtentStripeLengthMax		Optional	ExtentStripeLengthMax describes the maximum acceptable stripe length.
ParityLayout		Optional	ParityLayout describes the desired parity layout. The value may be 1 or 2 (Non-rotated Parity or Rotated Parity).
UserDataStripeDepth		Optional	UserDataStripeDepth describes the desired stripe depth.

Table 25: SMI Referenced Properties/Methods for CIM_StorageSetting

Properties	Flags	Requirement	Description & Notes
UserDataStripeDepthMin		Optional	UserDataStripeDepthMin describes the minimum acceptable stripe depth.
UserDataStripeDepthMax		Optional	UserDataStripeDepthMax describes the maximum acceptable stripe depth.
ChangeableType		Mandatory	This property informs a client if the setting can be modified. It also tells the client how long this setting is expected to remain in the model. If the implementation allows it, the client can use the property to request that the setting's existence be not transient.
StorageExtentInitialUsage		Optional	The Usage value to be used when creating a new storage element.
StoragePoolInitialUsage		Optional	The Usage value to be used when creating a new storage pool.

5.8.10 CIM_StorageSettingWithHints

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 26 describes class CIM_StorageSettingWithHints.

Table 26: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	The user-friendly name for this instance of SettingData. In addition, the user friendly name can be used as a index property for a search of query. (Note: Name does not have to be unique within a namespace.)
NoSinglePointOfFailure		Mandatory	
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	

Table 26: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Properties	Flags	Requirement	Description & Notes
DataAvailabilityHint		Mandatory	This hint is an indication from a client of the importance placed on data availability. Values are 0=Don't Care to 10=Very Important.
AccessRandomnessHint		Mandatory	This hint is an indication from a client of the randomness of accesses. Values are 0=Entirely Sequential to 10=Entirely Random.
AccessDirectionHint		Mandatory	This hint is an indication from a client of the direction of accesses. Values are 0=Entirely Read to 10=Entirely Write
AccessSizeHint		Mandatory	This hint is an indication from a client of the optimal access sizes. Several sizes can be specified. Units("Megabytes")
AccessLatencyHint		Mandatory	This hint is an indication from a client how important access latency is. Values are 0=Don't Care to 10=Very Important.
AccessBandwidthWeight		Mandatory	This hint is an indication from a client of bandwidth prioritization. Values are 0=Don't Care to 10=Very Important.
StorageCostHint		Mandatory	This hint is an indication of the importance the client places on the cost of storage. Values are 0=Don't Care to 10=Very Important. A StorageVolume provider might choose to place data on low cost or high cost drives based on this parameter.
StorageEfficiencyHint		Mandatory	This hint is an indication of the importance placed on storage efficiency by the client. Values are 0=Don't Care to 10=Very Important. A StorageVolume provider might choose different RAID levels based on this hint.
ChangeableType		Mandatory	

5.8.11 CIM_ElementSettingData

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 27 describes class CIM_ElementSettingData.

Table 27: SMI Referenced Properties/Methods for CIM_ElementSettingData

Properties	Flags	Requirement	Description & Notes
IsDefault		Mandatory	An enumerated integer indicating that the referenced setting is a default setting for the element, or that this information is unknown. Value shall be 0,1 or 2 (Unknown or Is Default or Is Not Default).
IsCurrent		Mandatory	An enumerated integer indicating that the referenced setting is currently being used in the operation of the element, or that this information is unknown. Value shall be 0,1 or 2 (Unknown or Is Default or Is Not Default).
ManagedElement		Mandatory	StorageVolume or LogicalDisk
SettingData		Mandatory	The Setting Data object associated with the ManagedElement.

5.8.12 CIM_ElementCapabilities (StorageCapabilities to StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 28 describes class CIM_ElementCapabilities (StorageCapabilities to StoragePool).

Table 28: SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageCapabilities to StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element.

5.8.13 CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 29 describes class CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService).

Table 29: SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

5.8.14 CIM_AllocatedFromStoragePool (Pool from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 30 describes class CIM_AllocatedFromStoragePool (Pool from Pool).

Table 30: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	Antecedent references the parent pool from which the dependent pool is allocated.
Dependent		Mandatory	

5.8.15 CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Array|Virt|HHRC

Table 31 describes class CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool).

Table 31: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

5.8.16 CIM_StorageVolume

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: Array|Virt|HHRC

Table 32 describes class CIM_StorageVolume.

Table 32: SMI Referenced Properties/Methods for CIM_StorageVolume

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Optional	User-friendly name
Name	CD	Mandatory	Identifier for this volume; based of datapath standards such as SCSI or ATAPI.
OtherIdentifyingInfo	CD	Optional	Additional correlatable names
IdentifyingDescriptions		Optional	
NameFormat		Mandatory	The type of identifier in the Name property.
NameNamespace		Mandatory	The namespace that defines uniqueness for the NameFormat.
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	Value shall be 2 3 6 8 15 (OK or Degraded or Error or Starting or Dormant).

Table 32: SMI Referenced Properties/Methods for CIM_StorageVolume

Properties	Flags	Requirement	Description & Notes
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks of capacity consumed from the parent StoragePool.
ConsumableBlocks		Mandatory	The number of blocks usable by consumers.
IsBasedOnUnderlyingRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.

5.8.17 CIM_SystemDevice (System to StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: Mandatory

Table 33 describes class CIM_SystemDevice (System to StorageVolume or LogicalDisk).

Table 33: SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

5.8.18 CIM_BasedOn (StorageVolume to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 34 describes class CIM_BasedOn (StorageVolume to extent from Extent Composition).

Table 34: SMI Referenced Properties/Methods for CIM_BasedOn (StorageVolume to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	StorageExtent instance from Extent Composition
Dependent		Mandatory	

5.8.19 CIM_LogicalDisk

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: LVM

Table 35 describes class CIM_LogicalDisk.

Table 35: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Optional	User-friendly name
Name		Mandatory	OS Device Name
NameFormat		Mandatory	Format for name
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	Value shall be 2 3 6 8 15 (OK or Degraded or Error or Starting or Dormant).
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks of capacity consumed from the parent StoragePool.

Table 35: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
ConsumableBlocks		Mandatory	The number of blocks usable by consumers.
IsBasedOnUnderlyingRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.

5.8.20 CIM_BasedOn (LogicalDisk to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 36 describes class CIM_BasedOn (LogicalDisk to extent from Extent Composition).

Table 36: SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	StorageExtent instance from Extent Composition
Dependent		Mandatory	

5.8.21 CIM_StorageSettingsAssociatedToCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 37 describes class CIM_StorageSettingsAssociatedToCapabilities.

Table 37: SMI Referenced Properties/Methods for CIM_StorageSettingsAssociatedToCapabilities

Properties	Flags	Requirement	Description & Notes
DefaultSetting		Mandatory	This boolean designates the setting that will be used if the CreateSetting() method is called with providing the NewSetting parameter. However, some implementations may require that the NewSetting parameter be non null. There may be only one default setting per the combination of StorageCapabilities and associated StoragePool as associated through ElementCapabilities.
Dependent		Mandatory	The StorageSetting reference.
Antecedent		Mandatory	The StorageCapabilities reference.

5.8.22 CIM_StorageSettingsGeneratedFromCapabilities

Created By: Extrinsic: CreateSetting

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 38 describes class CIM_StorageSettingsGeneratedFromCapabilities.

Table 38: SMI Referenced Properties/Methods for CIM_StorageSettingsGeneratedFromCapabilities

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

5.8.23 CIM_ConcreteComponent (storage element to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 39 describes class CIM_ConcreteComponent (storage element to extent from Extent Composition).

Table 39: SMI Referenced Properties/Methods for CIM_ConcreteComponent (storage element to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	StoragePool instance from Extent Composition

5.8.24 CIM_AssociatedComponentExtent (StoragePool to extent from Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 40 describes class CIM_AssociatedComponentExtent (StoragePool to extent from Composition).

Table 40: SMI Referenced Properties/Methods for CIM_AssociatedComponentExtent (Storage-Pool to extent from Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	Parent StoragePool
PartComponent		Mandatory	The referenced StorageExtent represents capacity has not been allocated, is allocated in part, or is allocated in its entirety.

5.8.25 CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 41 describes class CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition).

Table 41: SMI Referenced Properties/Methods for CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	Parent StoragePool

Table 41: SMI Referenced Properties/Methods for CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	The referenced StorageExtent represents the capacity of the StorageExtent on which it is based that was not used in resource allocation.

5.8.26 CIM_EnabledLogicalElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 42 describes class CIM_EnabledLogicalElementCapabilities.

Table 42: SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	The moniker for the instance.
ElementNameEditSupported		Mandatory	Denotes whether a storage element can be named
MaxElementNameLength		Mandatory	Specifies the maximum length in glyphs (letters) for the name. See MOF for details.
ElementNameMask		Mandatory	The regular expression that specifies the possible content and format for the element name. See MOF for details
RequestedStatesSupported		Optional	Expresses the states to which this element may be changed using the RequestStateChange method. If this property, it may be assumed that the state may not be changed.

5.8.27 CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 43 describes class CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool).

Table 43: SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

5.8.28 CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 44 describes class CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk).

Table 44: SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

5.8.29 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 45 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool).

Table 45: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.

Table 45: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	

5.8.30 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 46 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk).

Table 46: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	

STABLE

EXPERIMENTAL

Clause 6: Block Storage Views Profile

6.1 Description

6.1.1 Synopsis

Profile Name: Block Storage Views

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13 (SNIA_ classes are based on the 2.13 CIM Schema)

Central Class: SNIA_ViewCapabilities

Scoping Class: CIM_ComputerSystem

6.1.2 Overview

This Profile specifies SNIA_ View Classes for the Array, Storage Virtualizer and Volume Management Profiles.

In this release of SMI-S, SNIA_ view classes provide an optimization of retrieval of information provided by multiple (associated) instances in a Profile. There is no support for update of SNIA_ view classes instances. Update of a SNIA_ view class instance can only be accomplished by updating the base class instances from which the view is derived.

6.1.2.1 Goals of SNIA_ View Classes

6.1.2.1.1 Goals that SNIA_ View Classes are intended to address are

- Get more data in one call to CIM Server.

The CIM model for arrays and Storage Virtualizers involve a lot of classes and associations. The objective is to allow discovery of the array model using SNIA_ View Classes with a reduction in the number of association traversals required.

- Allow providers to optimize the Request.

In many cases, the data represented by a View Class is actually kept (and returned) by a device as one entity. When the "normalized" CIM model is traversed many calls are made to retrieve that one entity. The provider takes the data from the one entity and carves it up for each CIM request. In many cases this involves retrieving the same entity multiple times. The objective is to allow a Provider to return the single entity in one SMI-S request (for data that is typically kept together by the device).

6.1.2.1.2 Additional Goals

- Do more things in one call to CIM Server.

An example would be retrieval or discovery of model information with fewer calls. However, this goal also extends to updating the CIM model (e.g., configuration actions). The SNIA_ View Classes are NOT intended to help in the latter case. However, SNIA_ View Classes should facilitate access to underlying classes in support of configuration operations.

It is important to note that the SNIA_ View Classes proposal was based directly on experiences relating to the scalability and performance of SMI-S real-world implementations. The focus is on improving performance in large configurations (e.g. thousands of volumes and thousands of disk drives).

6.1.2.2 Specific Requirements and Objectives of View Classes

6.1.2.2.1 Pre-defined View Classes

In order to gain the desired performance advantage, it is felt that view classes would have to be pre-defined (in SMI-S) to allow provider optimization of the requested information.

- Enable Associator Calls to View Class instances.

It should be possible to retrieve a View Class by an associators call to the class.

However, it is desired that the association should be clearly distinguished from existing associations on the base classes.

- Enable Associator Calls from View Class instances.

It should be possible to get related classes (e.g., base classes) from the View Class by using associator calls.

Again, the associations used should be clearly distinguished from existing associations on the base classes.

6.1.2.2.2 Specific Views requested

- Getting asset information
- Mix of StorageVolume with LUN Mapping & Masking
- Getting port information (with endpoints) or ports & volumes
- Hardware ID & StorageVolumes
- Disk drive view
- Volumes & Settings
- Extent Composition
- Privilege Hierarchy
- Hardware ID <-> StorageVolume

Most of these requests are addressed by this Profile.

- Allow View Classes to be used where real classes would

This certainly includes "read" intrinsics and as parameters of Extrinsics

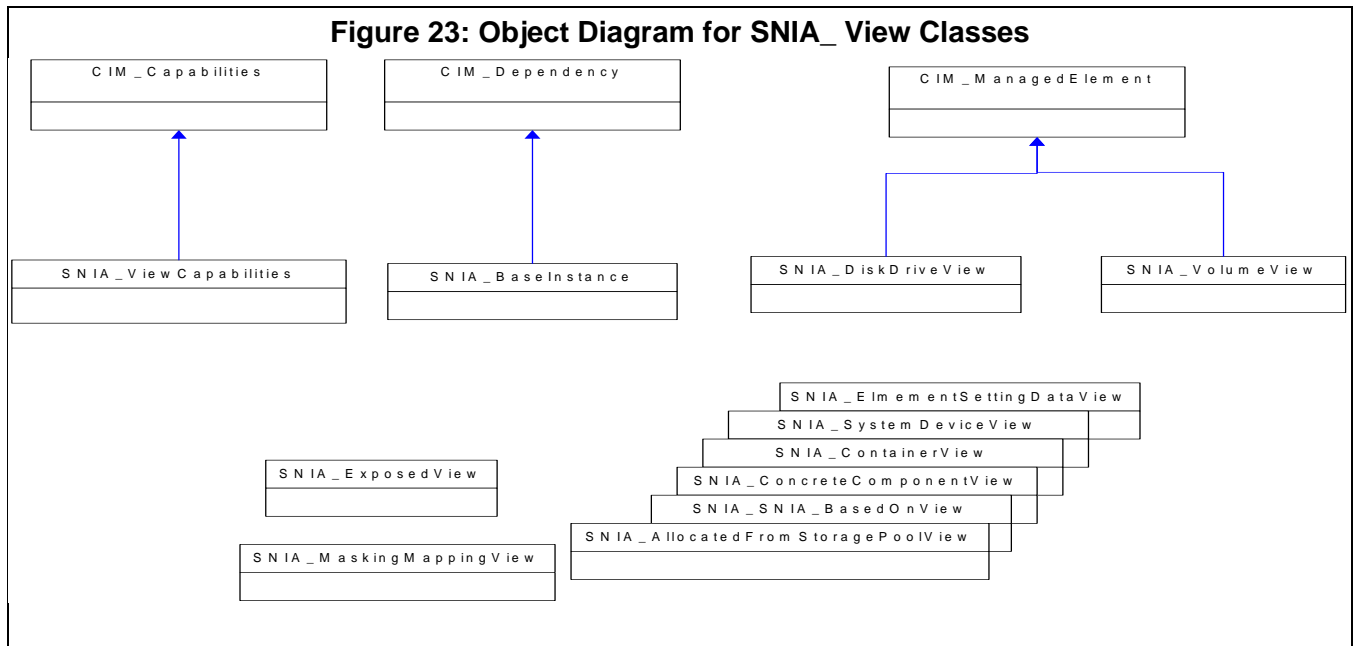
However, at this time "Write" intrinsic support is deferred and use in Extrinsics (as IN or OUT parameters) is not covered in this release of SMI-S.

6.1.2.2.3 Support Life Cycle Indications on View Classes

This requirement is being deferred for considered in a future release of SMI-S.

6.1.3 Object Diagram for SNIA View Classes

Figure 23 illustrates the object diagram for SNIA_ view classes.

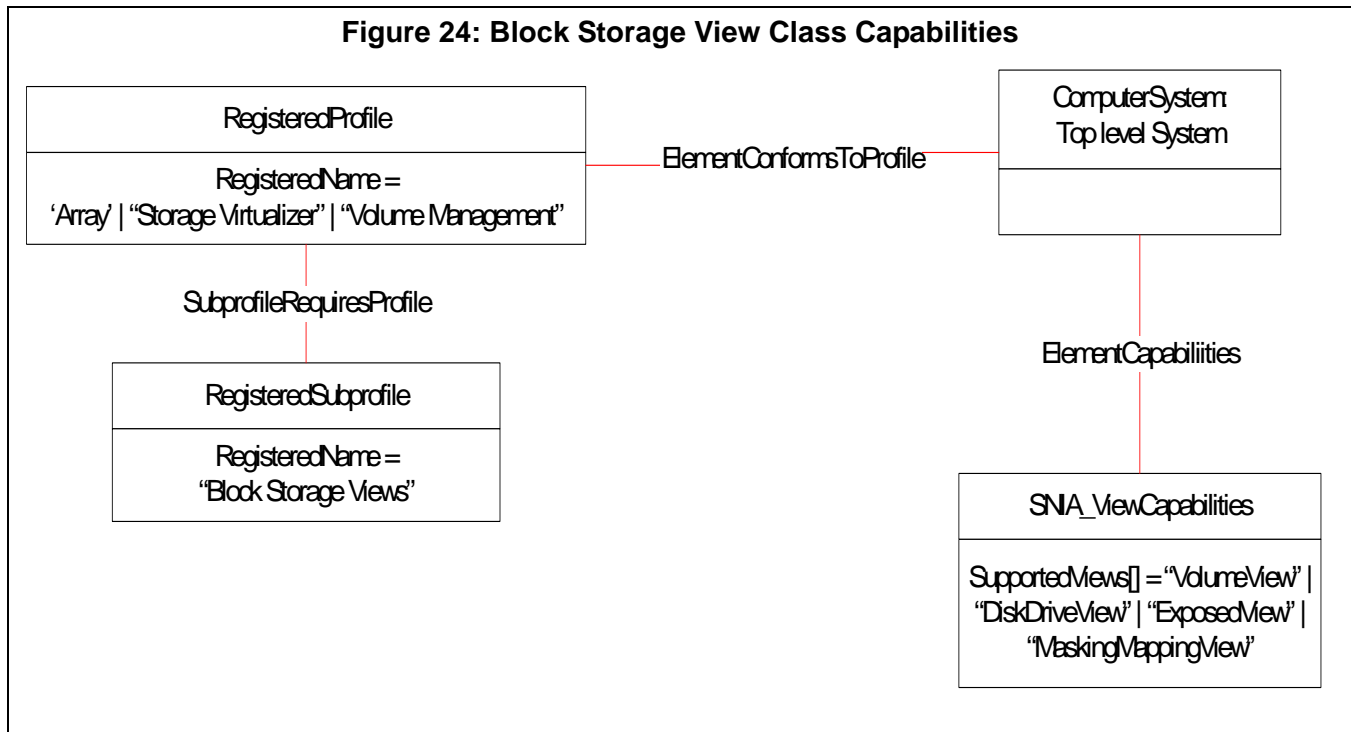


The **SNIA_ViewCapabilities** inherits from **CIM_Capabilities**. The **SNIA_VolumeView** and **SNIA_DiskDriveView** classes inherit from **CIM_ManagedElement**. The **SNIA_** association views (**SNIA_ExposedView** and **SNIA_MaskingMappingView**) do not inherit from anything.

6.1.4 Implementation

6.1.4.1 View Class Capabilities

The implementation of view classes shall be determined from a set of conditions. The model for determining whether or not the Block Storage Views Profile is supported and which views are supported is illustrated in Figure 24.

Figure 24: Block Storage View Class Capabilities

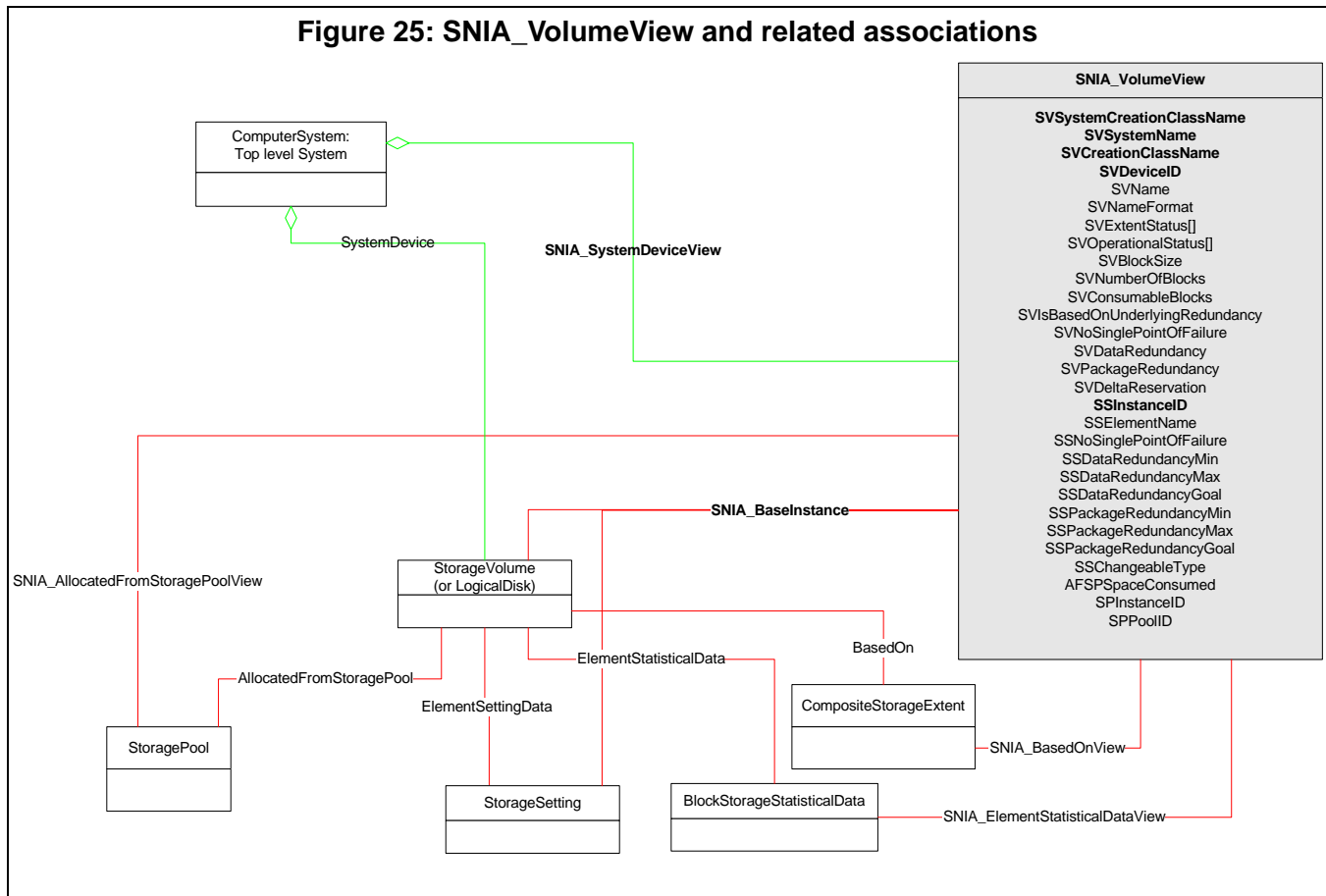
First a client shall be able to determine whether or not a profile implementation has implemented any view classes by looking for a `RegisteredSubprofile` with a `RegisteredName` of "Block Storage Views". If this `RegisteredSubprofile` exists then the profile supports some number of view classes.

Next a client would be able to determine which view classes are supported by an implementation by following the `ElementConformsToProfile` to the top level system and then following the `ElementCapabilities` from that system to the `SNIA_ViewCapabilities` instance. There shall be one instance of the `SNIA_ViewCapabilities` class if the profile supports the Block Storage Views Subprofile. The `SNIA_ViewCapabilities` instance shall have an array of strings (`SupportedViews`) that identify the view classes that are supported. For example, if the `SupportedViews` array includes the "VolumeView" string, then the `VolumeView` class shall be supported.

6.1.4.2 Block Services Views

6.1.4.2.1 SNIA_VolumeView and related associations

Figure 25 illustrates the `SNIA_VolumeView` and related associations.

Figure 25: SNIA_VolumeView and related associations

The **SNIA_VolumeView** is composed of information drawn from the following base classes:

- **StorageVolume (or LogicalDisk)**
- **StorageSetting**
- **AllocatedFromStoragePool**
- **StoragePool**

The keys for the **SNIA_VolumeView** are the **StorageVolume** and **StoragePool** keys from the base **CIM_StorageVolume** and **StoragePool** instances. There will be one instance of **SNIA_VolumeView** for each instance of **StorageVolume** if the **StorageVolume** is allocated from one **StoragePool**. If a **StorageVolume** is allocated from multiple **StoragePools** (e.g., **Composite Volumes**), there will be one instance of **SNIA_VolumeView** for each **StoragePool** from which the **StorageVolume** is allocated.

The information drawn from the **AllocatedFromStoragePool** association is the **SpaceConsumed** property. The properties from all other base classes shall be supported, but may be null.

6.1.4.2.2 Mandatory, Conditional and Optional Properties of **SNIA_VolumeView**

Properties that are mandatory in the mandatory base classes are mandatory in the **SNIA_VolumeView** class. Properties that are Conditional in the base classes are conditional in the **SNIA_VolumeView** class. Properties that are mandatory in optional (base) classes (**CompositeExtent**) are "conditional" in the **SNIA_VolumeView**. If an

optional base class is not supported by the Array or StorageVirtualizer implementation, these properties of those classes shall be present, but shall be null.

Properties in the base classes that are optional in the base class are optional in the SNIA_VolumeView.

6.1.4.2.3 Associations on SNIA_VolumeView

In this release of SMI-S the SNIA_VolumeView is "read only." Access to CIM class instances on which the view is based that can be updated (e.g., StorageVolume and StorageSetting) can be accessed from the SNIA_VolumeView instance via the SNIA_BaseInstance association.

In addition to the SNIA_VolumeView there are four associations that support association traversal to (or from) instances of the SNIA_VolumeView:

6.1.4.2.3.1 SNIA_SystemDeviceView

From the owning CIM_ComputerSystem a client will be able to find the SNIA_VolumeViews associated to the ComputerSystem via the SNIA_SystemDeviceView. This will return the VolumeViews that correspond to the StorageVolumes (or LogicalDisks) that would be found via association traversal from the ComputerSystem to the StorageVolumes (or LogicalDisks) via CIM_SystemDevice.

Similarly, if the client has a SNIA_VolumeView instance, the client can find the owning ComputerSystem by following the SNIA_SystemDeviceView association from the SNIA_VolumeView instance to the CIM_ComputerSystem instance for the ComputerSystem that scopes the StorageVolume (or LogicalDisk).

6.1.4.2.3.2 SNIA_AllocatedFromStoragePoolView

From the SNIA_VolumeView instance, the client can find the CIM_StoragePool instance by following the SNIA_AllocatedFromStoragePoolView association. Note that for one SNIA_VolumeView instance, there will be one CIM_StoragePool (that is, for Composite Volumes that draw from multiple StoragePools, there would be multiple SNIA_VolumeView instances that represent the composite volume.)

Similarly, from the CIM_StoragePool instance, a client can find all SNIA_VolumeView instances for StorageVolumes (or LogicalDisks) that are allocated from the StoragePool by following the AllocatedFromStoragePoolView association.

6.1.4.2.3.3 SNIA_BasedOnView

From the SNIA_VolumeView instance, the client can find the CIM_StorageExtent(s) on which the StorageVolume (or LogicalDisk) is based by following the BasedOnView.

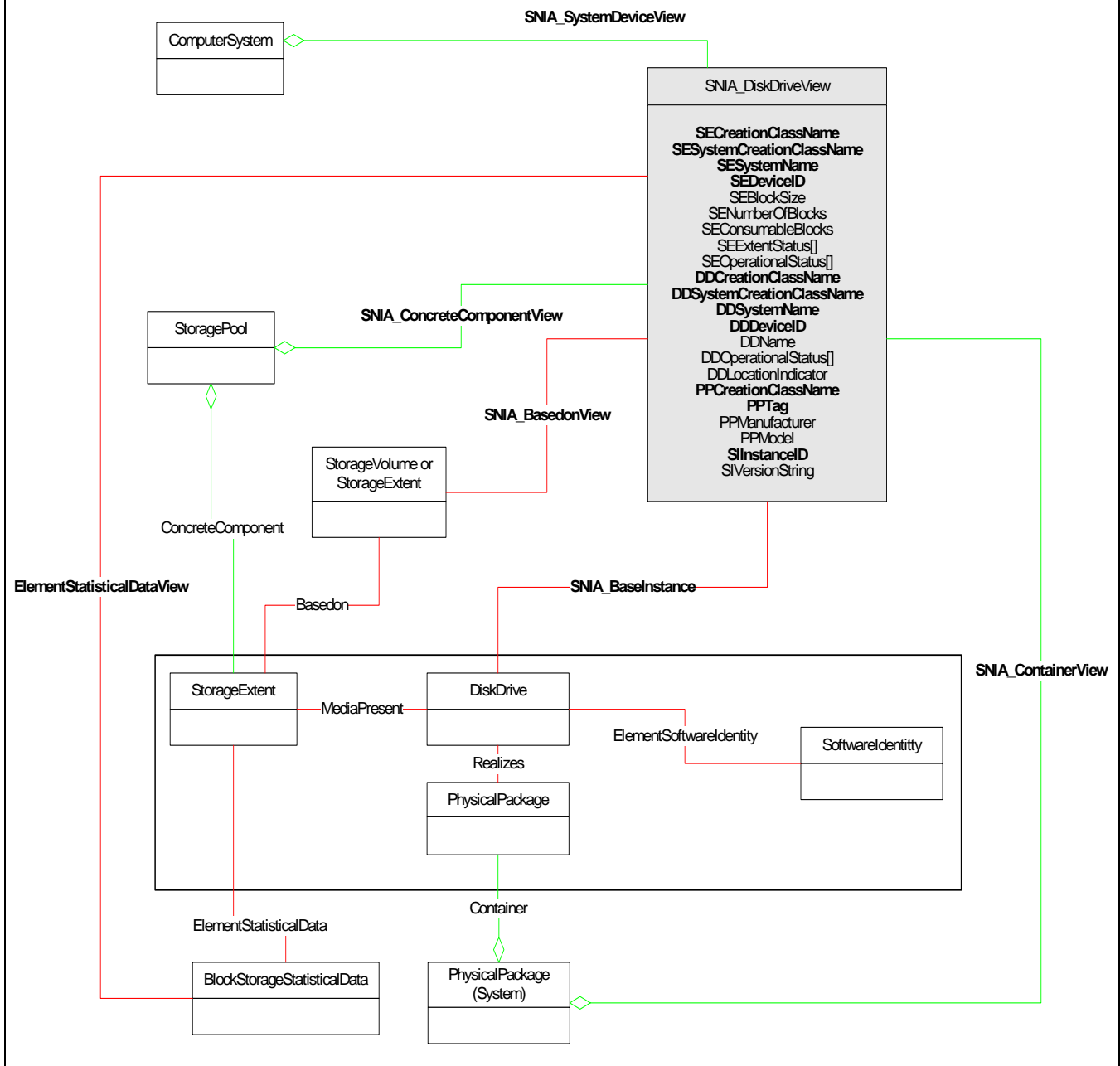
Similarly, from a "top level" CIM_StorageExtent instance, a client can find the SNIA_VolumeView instance(s) that are based on that StorageExtent.

6.1.4.2.3.4 SNIA_ElementStatisticalDataView

From the SNIA_VolumeView instance, the client can find the CIM_BlockStorageStatisticalData instance for the StorageVolume or LogicalDisk of the VolumeView by following the SNIA_ElementStatisticalDataView association.

6.1.4.3 Disk Drive Views

Figure 26 illustrates the DiskDriveView class and related associations.

Figure 26: SNIA_DiskDriveView and related associations

The SNIA_DiskDriveView is composed of information drawn from the following base classes:

- StorageExtent
- DiskDrive
- PhysicalPackage
- SoftwareIdentity (conditional)

The keys for the SNIA_DiskDriveView are the keys of the DiskDrive base class. There will be one instance of SNIA_DiskDriveView for each instance of a Disk Drive (primordial).

6.1.4.3.1 Mandatory, Conditional and Optional Properties of SNIA_DiskDriveView

The properties from base classes shall be supported, but may be null. Properties that are mandatory in mandatory base classes are mandatory in the SNIA_DiskDriveView class. Properties that are conditional in a base class are conditional in the SNIA_DiskDriveView class. Properties that are mandatory in optional (base) classes (BlockStorageStatisticalData and SoftwareIdentity) are also "conditional" in the SNIA_DiskDriveView. If an optional base class is not supported by the Array implementation, these properties of those classes shall be present but shall be null.

Properties in the base classes that are optional in the base class are optional in the SNIA_DiskDriveView.

6.1.4.3.2 Associations on SNIA_DiskDriveView

In this release of SMI-S, the SNIA_DiskDriveView is "read only." In order to support update of information in the SNIA_DiskDriveView instance, it would be necessary to update the class instances on which it is based. An association SNIA_BaseInstance is provided to the CIM_DiskDrive instance.

Note: The SNIA_BaseInstance association is only provided to base instances that can be modified.

In addition to the SNIA_DiskDriveView there are 5 associations that support association traversal to (or from) instances of the SNIA_DiskDriveView:

6.1.4.3.2.1 SNIA_ConcreteComponentView (mandatory if the DiskDriveView is implemented)

From a primordial CIM_StoragePool instance a client will be able to find the SNIA_DiskDriveViews associated to the StoragePool via the SNIA_ConcreteComponentView. This will return the DiskDriveView instances that correspond to the Disk Drive StorageExtents that would be found via association traversal from the StoragePool to the StorageExtents via CIM_ConcreteComponent association.

Similarly, if the client has a SNIA_DiskDriveView instance, the client can find the primordial StoragePool to which the drive is assigned by following the SNIA_ConcreteComponentView association from the SNIA_DiskDriveView instance to the CIM_StoragePool instance for the StoragePool that contains the Disk Drive StorageExtent.

6.1.4.3.2.2 SNIA_ContainerView (mandatory if the DiskDriveView is implemented)

From a system chassis (or other higher level physical package) instance a client will be able to find the SNIA_DiskDriveViews associated to the CIM_PhysicalPackage instance via the SNIA_ContainerView. This will return the DiskDriveView instances that correspond to the Disk Drive PhysicalPackage that would be found via association traversal from the system CIM_PhysicalPackage to the Disk Drive CIM_PhysicalPackage via CIM_Container association.

Similarly, if the client has a SNIA_DiskDriveView instance, the client can find the higher level system CIM_PhysicalPackage instance in which the drive resides by following the SNIA_ContainerView association from the SNIA_DiskDriveView instance to the CIM_PhysicalPackage instance for the higher level system physical package that contains the Disk Drive physical package.

6.1.4.3.2.3 SNIA_BasedOnView (mandatory if the DiskDriveView and Extent Composition are implemented)

From a higher level StorageExtent (e.g., CompositeExtent) instance a client will be able to find the SNIA_DiskDriveViews associated to the CIM_StorageExtent instance via the SNIA_BasedOnView. This will return the DiskDriveView instances that correspond to the Disk Drive StorageExtent that would be found via association traversal from the higher level CIM_StorageExtent to the Disk Drive CIM_StorageExtent via CIM_BasedOn association.

Similarly, if the client has a SNIA_DiskDriveView instance, the client can find the higher level CIM_StorageExtent instance that is based on the drive by following the SNIA_BasedOnView association from the SNIA_DiskDriveView

instance to the CIM_StorageExtent instance for the higher level storage extent that is based on the Disk Drive storage extent.

6.1.4.3.2.4 SNIA_SystemDeviceView (mandatory if the DiskDriveView is implemented)

From the owning CIM_ComputerSystem a client will be able to find the SNIA_VolumeViews associated to the ComputerSystem via the SNIA_SystemDeviceView. This will return the VolumeViews that correspond to the StorageVolumes (or LogicalDisks) that would be found via association traversal from the ComputerSystem to the StorageVolumes (or LogicalDisks) via CIM_SystemDevice.

Similarly, if the client has a SNIA_VolumeView instance, the client can find the owning ComputerSystem by following the SNIA_SystemDeviceView association from the SNIA_VolumeView instance to the CIM_ComputerSystem instance for the ComputerSystem that scopes the StorageVolume (or LogicalDisk).

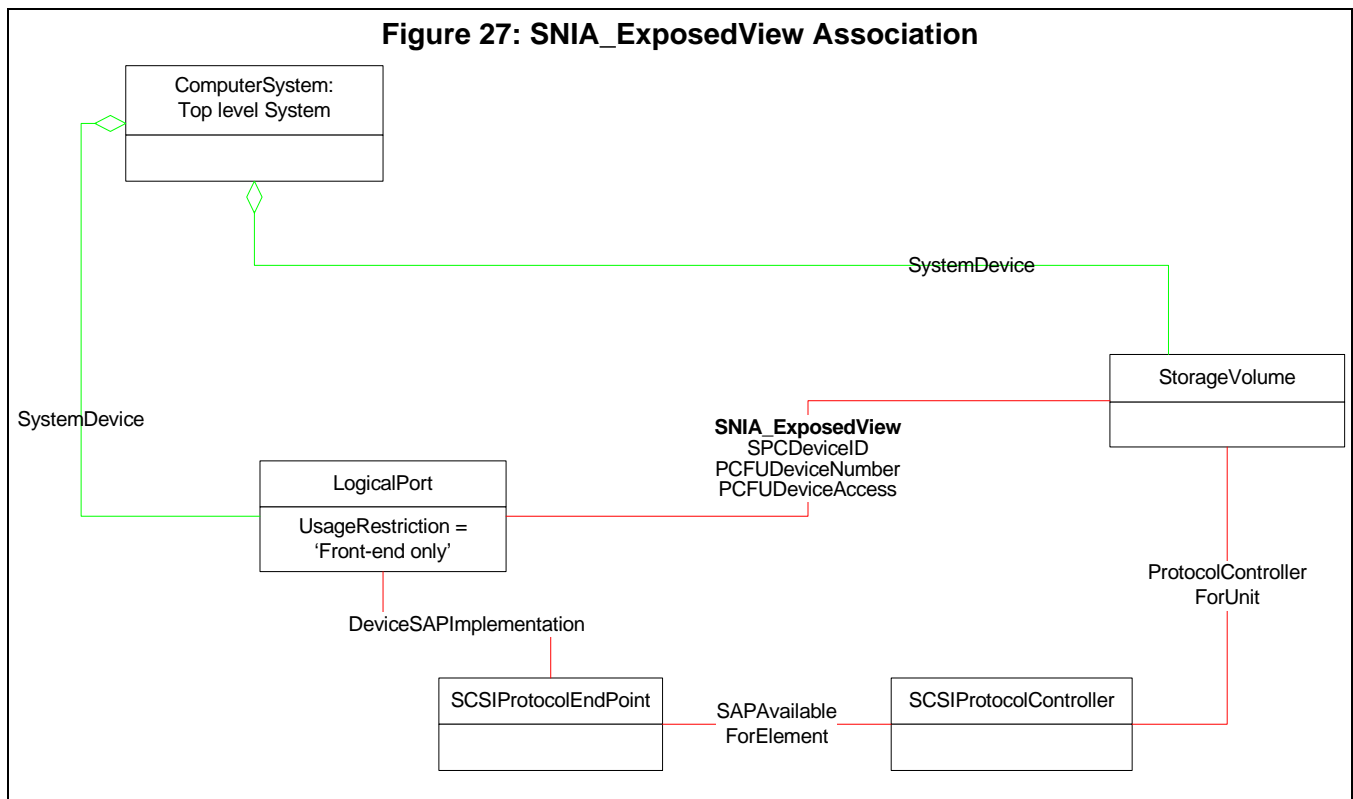
6.1.4.3.2.5 SNIA_ElementStatisticalDataView

From the SNIA_DiskDriveView instance, the client can find the CIM_BlockStorageStatisticalData instance for the Disk Drive StorageExtent of the DiskDriveView by following the SNIA_ElementStatisticalDataView association.

6.1.4.4 Masking and Mapping Views

6.1.4.4.1 The SNIA_ExposedView Association

Figure 27 illustrates the SNIA_ExposedView Association.



The SNIA_ExposedView association is composed of information drawn from the following base classes:

- SCSIProtocolEndpoint
- SCSIProtocolController

- DeviceSAPImplementation
- SAPAvailableForElement
- ProtocolControllerForUnit

The keys for the SNIA_ExposedView are the references to the LogicalDevice (a StorageVolume) and the reference to the LogicalPort. There will be one instance of SNIA_ExposedView for each unique combination of StorageVolume and LogicalPort through which the volume is exposed (in a Masking and Mapping model).

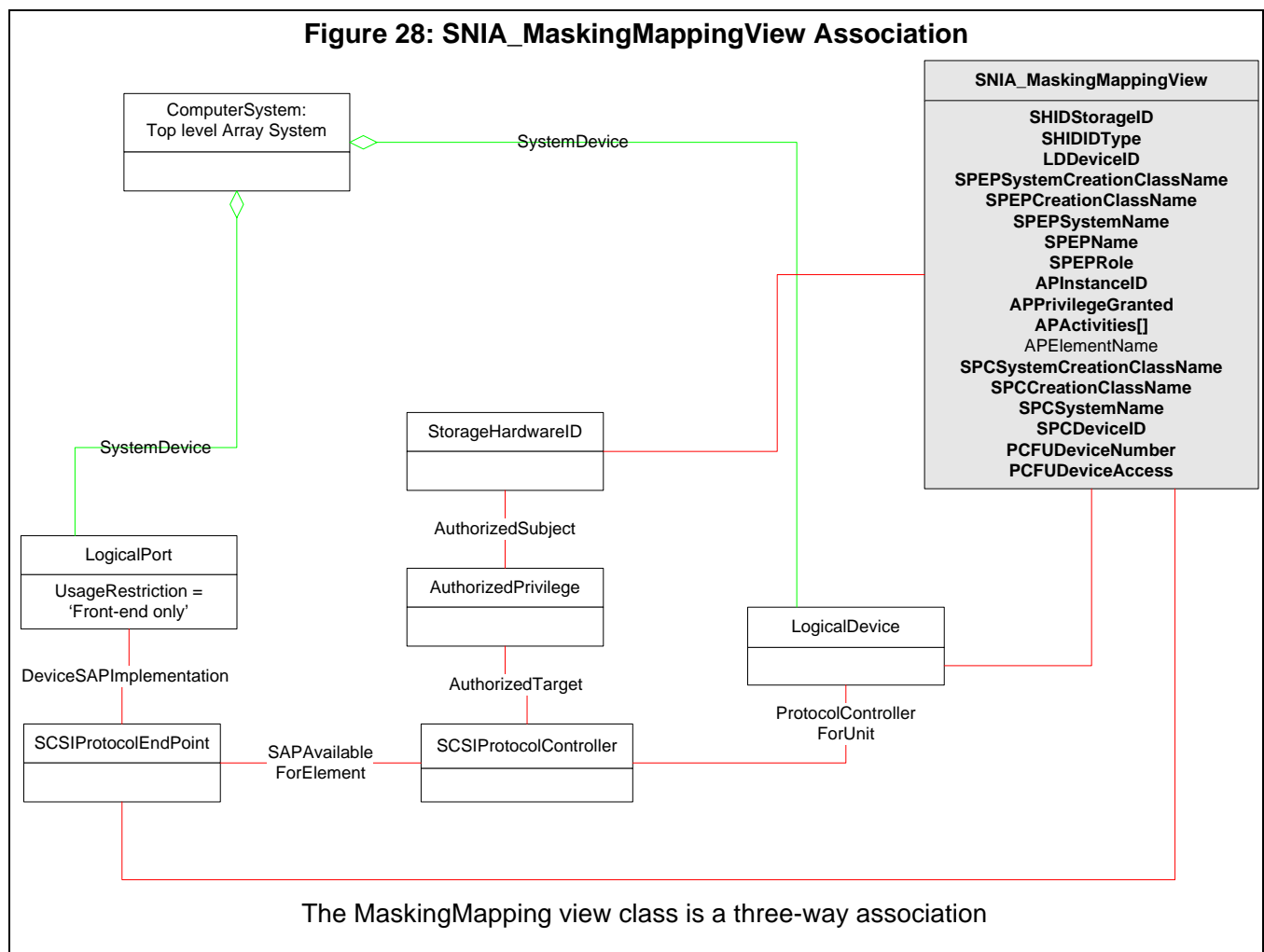
6.1.4.4.2 Mandatory, Conditional and Optional Properties of SNIA_ExposedView Association

In addition to the references to StorageVolume and the LogicalPort the SNIA_ExposedView association also carries the DeviceID of the SCSIProtocolController, the Name and ConnectionType properties of the SCSIProtocolEndPoint and the DeviceNumber and DeviceAccess properties from the ProtocolControllerForUnit association.

In this release of SMI-S, the SNIA_ExposedView is "read only." It would be used to do association traversal from StorageVolumes to LogicalPorts that expose the Volumes.

6.1.4.4.3 SNIA_ MaskingMapView Association

Figure 28 illustrates the SNIA_MaskingMapView Association.



The SNIA_MaskingMapView association is a three way association that is composed of information drawn from the following base classes:

- StorageHardwareID
- AuthorizedPrivilege
- SCSIProtocolController
- SCSIProtocolEndpoint
- ProtocolControllerForUnit
- LogicalDevice

The keys for the SNIA_MaskingMapView are the SHID reference, the SCSIProtocolEndpoint reference and the LogicalDevice reference. There will be one instance of SNIA_MaskingMapView for each unique combination of Storage Hardware ID (e.g., host), LogicalDevice (e.g., StorageVolume) and SCSIProtocolEndpoint (e.g., LogicalPort).

6.1.4.4.4 Mandatory, Conditional and Optional Properties of SNIA_MaskingMapView Association

In addition to the references to StorageHardwareID, LogicalDevice and the SCSIProtocolEndpoint the SNIA_MaskingMapView association also carries their properties and the AuthorizedPrivilege properties, DeviceID of the SCSIProtocolController and the DeviceNumber and DeviceAccess properties from the ProtocolControllerForUnit association. Also, for the convenience to clients, identifying properties from the LogicalDevice, StorageHardwareID and SCSIProtocolEndpoint are also pulled into the MaskingMapView. This allows a client to enumerate the SNIA_MaskingMapView association and get the identifiers for the endpoints in the association.

In this release of SMI-S, the SNIA_MaskingMapView is "read only." It would be used to do associate the StorageHardwareIDs, StorageVolumes to SCSIProtocolEndpoints.

6.2 Health and Fault Management Consideration

Health and Fault Management considerations are defined in terms of the base classes (no View Classes). However, it should be noted that OperationalStatus of view classes shall be the same as the OperationalStatus of the underlying CIM classes on which the view classes are defined.

6.3 Cascading Considerations

Not defined in this standard.

6.4 Supported Profiles, Subprofiles, and Packages

Table 47: Supported Profiles for Block Storage Views

Registered Profile Names	Mandatory	Version
Block Services	No	1.2.0
Block Server Performance	No	1.2.0
Disk Drive Lite	No	1.2.0
Masking and Mapping	No	1.2.0

Table 47: Supported Profiles for Block Storage Views

Registered Profile Names	Mandatory	Version
Extent Composition	No	1.2.0

6.5 Methods of the Profile

6.5.1 Extrinsic Methods of the Profile

None

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

SNIA_ View classes are modified by creating, deleting and modifying the base classes.

In the case of the SNIA_VolumeView, the only class instances that can be modified are the CIM_StorageVolume (or CIM_LogicalDisk) and CIM_StorageSetting instances. These instances may be accessed via association traversal through the SNIA_BaseInstance association.

In the case of the SNIA_DiskDriveView, the only class instance that can be modified is the CIM_DiskDrive class instance. This instance may be accessed via association traversal through the SNIA_BaseInstance association.

6.6 Client Considerations and Recipes

6.6.1 Use Cases

6.6.2 Recipes

6.7 Registered Name and Version

Block Storage Views version 1.2.0

6.8 CIM Elements

Table 48: CIM Elements for Block Storage Views

Element Name	Requirement	Description
CIM_ElementCapabilities (View Capabilities) (6.8.1)	Mandatory	Associates the top level ComputerSystem to the SNIA_ViewCapabilities supported by the implementation
SNIA_AllocatedFromStoragePoolView (6.8.2)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView". This associates a SNIA_VolumeView instance to a CIM_StoragePool. This is required if the SNIA_VolumeView is implemented.
SNIA_BaseInstance (StorageSetting) (6.8.3)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView". This associates a SNIA_VolumeView class instance to a base CIM_StorageSetting class instance that can be modified. This is required if the SNIA_VolumeView is implemented.
SNIA_BaseInstance (Volume) (6.8.4)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView". This associates a SNIA_VolumeView instance to a base CIM_StorageVolume (or CIM_LogicalDisk) instance that can be modified. This is required if the SNIA_VolumeView is implemented.
SNIA_BaseInstance (DiskDrive) (6.8.5)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView". This associates a SNIA_DiskDriveView instance to a base CIM_DiskDrive instance that can be modified. This is required if the SNIA_DiskDriveView is implemented.
SNIA_BasedOnView (ExtentOnDriveExtent) (6.8.6)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView" and Extent Composition is implemented. This associates a CIM_StorageExtent (or StorageVolume or LogicalDisk) instance to a SNIA_DiskDriveView instance. This is required if the SNIA_DiskDriveView and ExtentComposition are implemented.

Table 48: CIM Elements for Block Storage Views

Element Name	Requirement	Description
SNIA_BasedOnView (VolumeOnExtent) (6.8.7)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView" and Extent Composition is implemented. This associates a SNIA_VolumeView instance to a base CIM_StorageExtent instance on which the volume is based. This is required if the SNIA_VolumeView and ExtentComposition are implemented.
SNIA_ConcreteComponentView (6.8.8)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView". The SNIA_ConcreteComponentView associates the SNIA_DiskDriveView instance to the primordial StoragePool to which the disk drive StorageExtent is assigned. This is required if the SNIA_DiskDriveView is implemented.
SNIA_ContainerView (6.8.9)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView". The SNIA_ContainerView associates the SNIA_DiskDriveView instance to the higher level physical package (e.g., System physical package) that contains the physical package of the disk drive. This is required if the SNIA_DiskDriveView is implemented.
SNIA_DiskDriveView (6.8.10)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView". The SNIA_DiskDriveView instance represents a Disk Drive and its associated information. This is required if SNIA_ViewCapabilities.SupportedViews includes "DiskDriveView"
SNIA_ElementStatisticalDataView (VolumeView) (6.8.11)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView" and Block Server Performance is implemented. This associates a SNIA_VolumeView instance to the CIM_BlockStorageStatisticalData instance for the StorageVolume (or LogicalDisk). This is required if the SNIA_VolumeView and the Block Server Performance Subprofile are implemented.

Table 48: CIM Elements for Block Storage Views

Element Name	Requirement	Description
SNIA_ElementStatisticalDataView (DiskDriveView) (6.8.12)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView" and Block Server Performance is implemented. This associates a SNIA_DiskDriveView instance to the CIM_BlockStorageStatisticalData instance for the Disk Drive. This is required if the SNIA_DiskDriveView and the Block Server Performance Subprofile are implemented.
SNIA_ExposedView (6.8.13)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "ExposedView". This view associates a LogicalPort and a LogicalDevice (e.g., StorageVolume). This is required if the SNIA_ViewCapabilities.SupportedViews includes "ExposedView".
SNIA_MaskingMappingView (6.8.14)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "MaskingMappingView". This three way association associates a CIM_LogicalDevice, CIM_StorageHardwareID and CIM_SCSIProtocolEndpoint instances to each other and derived from the Masking and Mapping subprofile model. This is required if SNIA_ViewCapabilities.SupportedViews contains "MaskingMappingView".
SNIA_ViewCapabilities (6.8.15)	Mandatory	The SNIA_ViewCapabilities identifies the capabilities of the implementation of view classes.
SNIA_VolumeView (6.8.16)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView". The SNIA_VolumeView represents the storage (LogicalDisks or StorageVolumes) of a block storage profile. This is required if the SNIA_ViewCapabilities.SupportedViews includes "VolumeView".

Table 48: CIM Elements for Block Storage Views

Element Name	Requirement	Description
SNIA_SystemDeviceView (DiskDriveViews) (6.8.17)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "DiskDriveView". This association links SNIA_DiskDriveView instances to the scoping system. This is required if the SNIA_DiskDriveView is implemented.
SNIA_SystemDeviceView (VolumeViews) (6.8.18)	Conditional	Conditional requirement: Required if the array property SNIA_ViewCapabilities.SupportedViews contains the string "VolumeView". This association links SNIA_VolumeView instances to the scoping system. This is required if the SNIA_VolumeView is implemented.

6.8.1 CIM_ElementCapabilities (View Capabilities)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 49 describes class CIM_ElementCapabilities (View Capabilities).

Table 49: SMI Referenced Properties/Methods for CIM_ElementCapabilities (View Capabilities)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The ViewCapabilities.
ManagedElement		Mandatory	The top level ComputerSystem that has the ViewCapabilities.

6.8.2 SNIA_AllocatedFromStoragePoolView

The SNIA_AllocatedFromStoragePoolView instance is a view that is derived from the
CIM_AllocatedFromStoragePool association between the StorageVolume or LogicalDisk (of the
SNIA_VolumeView) and the StoragePool from which the StorageVolume (or LogicalDisk is allocated. Note that if
the StorageVolume (or LogicalDisk) is allocated from multiple StoragePools there will be multiple
AllocatedFromStoragePoolView instances for the StorageVolume (or LogicalDisk). The
SNIA_AllocatedFromStoragePoolView is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeView

Table 50 describes class SNIA_AllocatedFromStoragePoolView.

Table 50: SMI Referenced Properties/Methods for SNIA_AllocatedFromStoragePoolView

Properties	Flags	Requirement	Description & Notes
AFSPSpaceConsumed		Mandatory	The space consumed from the StoragePool by the StorageVolume (or LogicalDisk). This value is the same as the AllocatedFromStoragePool.SpaceConsumed value for the base CIM_StorageVolume on the antecedent StoragePool.
Antecedent		Mandatory	A StoragePool from which the StorageVolume of the SNIA_VolumeView is allocated.
Dependent		Mandatory	The SNIA_VolumeView instance that is allocated from the StoragePool. There is only one VolumeView instance for the combined StorageVolume (or LogicalDisk) - StoragePool pair.

6.8.3 SNIA_BaseInstance (StorageSetting)

The SNIA_BaseInstance instance is an association between the SNIA_VolumeView and the CIM_StorageSetting instance for the base StorageVolume (or LogicalDisk) on which the view is based. This association is provided to accommodate update operations on the CIM_StorageSetting instance (e.g., ModifyInstance), since the properties cannot be updated in the view class. The SNIA_BaseInstance is subclassed from CIM_Dependency.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeView

Table 51 describes class SNIA_BaseInstance (StorageSetting).

Table 51: SMI Referenced Properties/Methods for SNIA_BaseInstance (StorageSetting)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The base CIM_StorageSetting instance on which the SNIA_VolumeView instance is based.
Dependent		Mandatory	The SNIA_VolumeView instance that is based on the CIM_StorageSetting instance.

6.8.4 SNIA_BaseInstance (Volume)

The SNIA_BaseInstance instance is an association between a SNIA_VolumeView instance and a base CIM_StorageVolume (or CIM_LogicalDisk) instance on which the view is based. This association is provided to

accommodate update operations on the base CIM_StorageVolume (or CIM_LogicalDisk) instances, since the properties cannot be updated in the view class. The SNIA_BaseInstance is subclassed from CIM_Dependency.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeView

Table 52 describes class SNIA_BaseInstance (Volume).

Table 52: SMI Referenced Properties/Methods for SNIA_BaseInstance (Volume)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The base CIM_StorageVolume (or CIM_LogicalDisk) instance on which the SNIA_VolumeView instance is based.
Dependent		Mandatory	The SNIA_VolumeView instance that is based on the CIM_StorageVolume (or CIM_LogicalDisk) instance.

6.8.5 SNIA_BaseInstance (DiskDrive)

The SNIA_BaseInstance instance is an association between a SNIA_DiskDriveView instance and a base CIM_DiskDrive instance on which the view is based. This association is provided to accommodate update operations on the base CIM_DiskDrive instances, since the properties cannot be updated in the view class. The SNIA_BaseInstance is subclassed from CIM_Dependency.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: DiskDriveView

Table 53 describes class SNIA_BaseInstance (DiskDrive).

Table 53: SMI Referenced Properties/Methods for SNIA_BaseInstance (DiskDrive)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The base CIM_DiskDrive instance on which the SNIA_DiskDriveView instance is based.
Dependent		Mandatory	The SNIA_DiskDriveView instance that is based on the CIM_DiskDrive instance.

6.8.6 SNIA_BasedOnView (ExtentOnDriveExtent)

The SNIA_BasedOnView instance is a view that is derived from CIM_BasedOn between the CIM_StorageExtent (or StorageVolume or LogicalDisk) instance and the primordial CIM_StorageExtent under it. The SNIA_BasedOnView is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: DiskDriveViewComp

Table 54 describes class SNIA_BasedOnView (ExtentOnDriveExtent).

Table 54: SMI Referenced Properties/Methods for SNIA_BasedOnView (ExtentOnDriveExtent)

Properties	Flags	Requirement	Description & Notes
BOStartingAddress		Optional	This is derived from the BasedOn.StartingAddress
BOEndingAddress		Optional	This is derived from the BasedOn.EndingAddress
BOOrderIndex		Optional	When the association is used in a concatenation composition, indicates the order in which the extents(and thus their block ranges) are concatenated.
Antecedent		Mandatory	The SNIA_DiskDriveView on which a StorageExtent (or StorageVolume or LogicalDisk) is based.
Dependent		Mandatory	The CIM_StorageExtent instance that is dependent on the SNIA_DiskDriveView

6.8.7 SNIA_BasedOnView (VolumeOnExtent)

The SNIA_BasedOnView instance is a view that is derived from CIM_BasedOn between the CIM_StorageVolume instance and the first CIM_StorageExtent it is based on. The SNIA_BaseOnView is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeViewComp

Table 55 describes class SNIA_BasedOnView (VolumeOnExtent).

Table 55: SMI Referenced Properties/Methods for SNIA_BasedOnView (VolumeOnExtent)

Properties	Flags	Requirement	Description & Notes
BOStartingAddress		Optional	This is derived from the BasedOn.StartingAddress
BOEndingAddress		Optional	This is derived from the BasedOn.EndingAddress
BOOrderIndex		Optional	When the association is used in a concatenation composition, indicates the order in which the extents(and thus their block ranges) are concatenated.
Antecedent		Mandatory	The lower level StorageExtent on which the SNIA_VolumeView StorageVolume is based.

Table 55: SMI Referenced Properties/Methods for SNIA_BasedOnView (VolumeOnExtent)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	The SNIA_VolumeView instance

6.8.8 SNIA_ConcreteComponentView

The SNIA_ConcreteComponentView instance is a view that is derived from the CIM_ConcreteComponent between the base CIM_StorageExtent of the Disk Drive and its primordial CIM_StoragePool. The SNIA_ConcreteComponentView is not subclassed from anything.

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: DiskDriveView

Table 56 describes class SNIA_ConcreteComponentView.

Table 56: SMI Referenced Properties/Methods for SNIA_ConcreteComponentView

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The CIM_StoragePool to which the StorageExtent of the Disk Drive is assigned.
PartComponent		Mandatory	A SNIA_DiskDriveView instance that is assigned to the StoragePool.

6.8.9 SNIA_ContainerView

The SNIA_ContainerView instance is a view that is derived from the CIM_Container between the base CIM_PhysicalPackage of the Disk Drive and the CIM_PhysicalPackage of the ComputerSystem. The SNIA_ContainerView is not subclassed from anything.

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: DiskDriveView

Table 57 describes class SNIA_ContainerView.

Table 57: SMI Referenced Properties/Methods for SNIA_ContainerView

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The CIM_PhysicalPackage for the ComputerSystem instance that groups the CIM_PhysicalPackage of the Disk Drive

Table 57: SMI Referenced Properties/Methods for SNIA_ContainerView

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	A SNIA_DiskDriveView instance that includes CIM_PhysicalPackage information for the CIM_DiskDrive

6.8.10 SNIA_DiskDriveView

The SNIA_DiskDriveView instance is a view that is derived from CIM_StorageExtent, CIM_MediaPresent, CIM_DiskDrive, CIM_Realizes, CIM_PhysicalPackage, CIM_ElementSoftwareIdentity and CIM_SoftwareIdentity. The SNIA_DiskDriveView is subclassed from CIM_ManagedElement.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: DiskDriveView

Table 58 describes class SNIA_DiskDriveView.

Table 58: SMI Referenced Properties/Methods for SNIA_DiskDriveView

Properties	Flags	Requirement	Description & Notes
SESystemCreationClassName		Mandatory	The SystemCreationClassName for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SESystemName		Mandatory	The SystemName for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SECreationClassName		Mandatory	The CreationClassName for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SEDeviceID		Mandatory	The DeviceID for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SEBlockSize		Mandatory	The BlockSize for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SENumberOfBlocks		Mandatory	The NumberOfBlocks for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SEConsumableBlocks		Mandatory	The ConsumableBlocks for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
SEExtentStatus		Mandatory	The ExtentStatus for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.

Table 58: SMI Referenced Properties/Methods for SNIA_DiskDriveView

Properties	Flags	Requirement	Description & Notes
SEOperationalStatus		Mandatory	The OperationalStatus for the StorageExtent of the Disk Drive as reported in the underlying primordial StorageExtent instance for the Disk Drive.
DDSystemCreationClassName		Mandatory	The SystemCreationClassName for the Disk Drive as reported in the underlying DiskDrive instance.
DDSystemName		Mandatory	The SystemName for the Disk Drive as reported in the underlying DiskDrive instance.
DDCreationClassName		Mandatory	The CreationClassName for the Disk Drive as reported in the underlying DiskDrive instance.
DDDeviceID		Mandatory	The DeviceID for the Disk Drive as reported in the underlying DiskDrive instance.
DDName		Mandatory	The Name for the Disk Drive as reported in the underlying DiskDrive instance.
DDOperationalStatus		Mandatory	The OperationalStatus for the Disk Drive as reported in the underlying DiskDrive instance.
PPCreationClassName		Mandatory	The CreationClassName for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.
PPTag		Mandatory	The Tag for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.
PPManufacturer		Mandatory	The Manufacturer for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.
PPModel		Mandatory	The Model for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.
SIInstanceID		Mandatory	The InstanceID for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
SIVersionString		Mandatory	The VersionString for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
DDLLocationIndicator		Optional	The LocationIndicator for the Disk Drive as reported in the underlying DiskDrive instance.
PPSerialNumber		Optional	The SerialNumber for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.
PPPartNumber		Optional	The PartNumber for the PhysicalPackage of the Disk Drive as reported in the underlying PhysicalPackage instance for the Disk Drive.

Table 58: SMI Referenced Properties/Methods for SNIA_DiskDriveView

Properties	Flags	Requirement	Description & Notes
SIManufacturer		Optional	The Manufacturer for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
SIBuildNumber		Optional	The BuildNumber for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
SIMajorVersion		Optional	The MajorVersion for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
SIRevisionNumber		Optional	The RevisionNumber for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.
SIMinorVersion		Optional	The MinorVersion for the SoftwareIdentity of the Disk Drive as reported in the underlying SoftwareIdentity instance for the Disk Drive.

6.8.11 SNIA_ElementStatisticalDataView (VolumeView)

The SNIA_ElementStatisticalDataView is an association between a SNIA_VolumeView instance and the CIM_BlockStorageStatisticalData instance for the StorageVolume (or LogicalDisk). The SNIA_BaseInstance is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeViewPerf

Table 59 describes class SNIA_ElementStatisticalDataView (VolumeView).

Table 59: SMI Referenced Properties/Methods for SNIA_ElementStatisticalDataView (Volume-View)

Properties	Flags	Requirement	Description & Notes
Stats		Mandatory	The CIM_BlockStorageStatisticalData instance for the StorageVolume (or LogicalDisk) instance.
ManagedElement		Mandatory	The SNIA_VolumeView instance that has the CIM_BlockStorageStatisticalDatainstance.

6.8.12 SNIA_ElementStatisticalDataView (DiskDriveView)

The SNIA_ElementStatisticalDataView is an association between a SNIA_DiskDriveView instance and the CIM_BlockStorageStatisticalData instance for the DiskDrive. The SNIA_BaseInstance is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: DiskDriveViewPerf

Table 60 describes class SNIA_ElementStatisticalDataView (DiskDriveView).

Table 60: SMI Referenced Properties/Methods for SNIA_ElementStatisticalDataView (DiskDriveView)

Properties	Flags	Requirement	Description & Notes
Stats		Mandatory	The CIM_BlockStorageStatisticalData instance for the DiskDrive (StorageExtent) instance.
ManagedElement		Mandatory	The SNIA_DiskDriveView instance that has the CIM_BlockStorageStatisticalData instance.

6.8.13 SNIA_ExposedView

The SNIA_ExposedView instance is a view that is derived from CIM_DeviceSAPImplementation, CIM_SCSIProtocolEndpoint, CIM_SAPAvailableForElement, CIM_SCSIProtocolController and CIM_ProtocolControllerForUnit. The SNIA_ExposedView is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: ExposedView

Table 61 describes class SNIA_ExposedView.

Table 61: SMI Referenced Properties/Methods for SNIA_ExposedView

Properties	Flags	Requirement	Description & Notes
PEPSystemCreationClassName		Mandatory	The SystemCreationClassName for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPSystemName		Mandatory	The SystemName for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPCreationClassName		Mandatory	The CreationClassName for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
SPCSystemCreationClassName		Mandatory	The SystemCreationClassName for the SCSIProtocolController used with the LogicalPort as reported in the underlying SCSIProtocolController instance for the LogicalPort and StorageVolume.

Table 61: SMI Referenced Properties/Methods for SNIA_ExposedView

Properties	Flags	Requirement	Description & Notes
SPCSystemName		Mandatory	The SystemName for the SCSIProtocolController used with the LogicalPort as reported in the underlying SCSIProtocolController instance for the LogicalPort and StorageVolume.
SPCCreationClassName		Mandatory	The CreationClassName for the SCSIProtocolController used with the LogicalPort as reported in the underlying SCSIProtocolController instance for the LogicalPort and StorageVolume.
SPCDeviceID		Mandatory	The DeviceID for the SCSIProtocolController used with the LogicalPort as reported in the underlying SCSIProtocolController instance for the LogicalPort and StorageVolume.
PCFUDeviceNumber		Mandatory	The DeviceNumber (LUN) for the StorageVolume when accessed through the LogicalPort as reported in the underlying ProtocolControllerForUnit instance for the StorageVolume.
PCFUDeviceAccess		Mandatory	The DeviceAccess value for the StorageVolume when accessed through the LogicalPort as reported in the underlying ProtocolControllerForUnit instance for the StorageVolume.
PEPName		Optional	The Name for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPProtocolIFType		Optional	The ProtocolIFType value for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPOtherTypeDescription		Optional	The OtherTypeDescription value for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPConnectionType		Optional	The ConnectionType value for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
PEPRole		Optional	The Role value for the SCSIProtocolEndpoint used with the LogicalPort as reported in the underlying SCSIProtocolEndpoint instance for the LogicalPort.
LogicalPort		Mandatory	The LogicalPort through which the LogicalDevice is exposed.
LogicalDevice		Mandatory	The LogicalDevice (e.g., StorageVolume) that is exposed through the LogicalPort

6.8.14 SNIA_MaskingMapView

The SNIA_MaskingMapView instance is a view that is derived from CIM_StorageHardwareID, CIM_AuthorizedSubject, CIM_AuthorizedPrivilege, CIM_AuthorizedTarget, CIM_SCSIProtocolController, CIM_SAPAvailableForElement, CIM_SCSIProtocolEndpoint, CIM_ProtocolControllerForUnit and CIM_LogicalDevice. The SNIA_MaskingMapView is not subclassed from anything.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: MaskingMapView

Table 62 describes class SNIA_MaskingMapView.

Table 62: SMI Referenced Properties/Methods for SNIA_MaskingMapView

Properties	Flags	Requirement	Description & Notes
SHIDStorageID		Mandatory	The StorageID from the referenced CIM_StorageHardwareID instance.
SHIDIDType		Mandatory	The IDType from the referenced CIM_StorageHardwareID instance.
LDDeviceID		Mandatory	The DeviceID from the referenced CIM_LogicalDevice instance.
SPEPSystemCreationClassName		Mandatory	The SystemCreationClassName from the referenced CIM_SCSIProtocolEndpoint instance.
SPEPCreationClassName		Mandatory	The CreationClassName from the referenced CIM_SCSIProtocolEndpoint instance.
SPEPSystemName		Mandatory	The SystemName from the referenced CIM_SCSIProtocolEndpoint instance.
SPEPName		Mandatory	The Name from the referenced CIM_SCSIProtocolEndpoint instance.
SPEPRole		Mandatory	The Role from the referenced CIM_SCSIProtocolEndpoint instance.
APIInstanceID		Mandatory	The InstanceID of the CIM_AuthorizedPrivilege instance.
APPrivilegeGranted		Mandatory	The PrivilegeGranted of the CIM_AuthorizedPrivilege instance.
APActivities		Mandatory	The Activities array of the CIM_AuthorizedPrivilege instance.
APElementName		Optional	The ElementName of the CIM_AuthorizedPrivilege instance.
SPCSystemCreationClassName		Mandatory	The SystemCreationClassName of the CIM_SCSIProtocolController instance.

Table 62: SMI Referenced Properties/Methods for SNIA_MaskingMapView

Properties	Flags	Requirement	Description & Notes
SPCCreationClassName		Mandatory	The CreationClassName of the CIM_SCSIProtocolController instance.
SPCSystemName		Mandatory	The SystemName of the CIM_SCSIProtocolController instance.
SPCDeviceID		Mandatory	The DeviceID of the CIM_SCSIProtocolController instance.
PCFUDeviceNumber		Mandatory	The DeviceNumber (LUN) of the CIM_ProtocolControllerForUnit association instance.
PCFUDeviceAccess		Mandatory	The DeviceAccess value of the CIM_ProtocolControllerForUnit association instance.
StorageHardwareID		Mandatory	The CIM_StorageHardwareID instance that is associated to the CIM_LogicalDevice and CIM_ProtocolEndpoint instances.
LogicalDevice		Mandatory	The CIM_LogicalDevice instance that is associated to the CIM_StorageHardwareID and CIM_ProtocolEndpoint instances.
ProtocolEndpoint		Mandatory	The CIM_ProtocolEndpoint instance that is associated to the CIM_StorageHardwareID and CIM_LogicalDevice instances.

6.8.15 SNIA_ViewCapabilities

The SNIA_ViewCapabilities instance defines the capabilities of an implementation support for SNIA_view classes. The SNIA_ViewCapabilities is subclassed from CIM_Capabilities.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 63 describes class SNIA_ViewCapabilities.

Table 63: SMI Referenced Properties/Methods for SNIA_ViewCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	An opaque, unique id for the view class capability of an implementation.
ElementName		Optional	A provider supplied user-Friendly Name for this SNIA_ViewCapabilities element.
SupportedViews		Mandatory	This array of strings lists the view classes that are supported by the implementation. Valid string values are "VolumeView" "DiskDriveView" "ExposedView" "MaskingMapView".

6.8.16 SNIA_VolumeView

The SNIA_VolumeView instance is a view that is derived from CIM_StorageVolume, CIM_ElementSettingData, CIM_StorageSetting, CIM_AllocatedFromStoragePool and CIM_StoragePool. The SNIA_VolumeView is subclassed from CIM_ManagedElement.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: VolumeView

Table 64 describes class SNIA_VolumeView.

Table 64: SMI Referenced Properties/Methods for SNIA_VolumeView

Properties	Flags	Requirement	Description & Notes
SVSystemCreationClassName		Mandatory	
SVSystemName		Mandatory	
SVCreationClassName		Mandatory	
SVDeviceID		Mandatory	An opaque identifier of the underlying StorageVolume (or LogicalDisk).
SVName		Mandatory	The identifier for the underlying StorageVolume (or LogicalDisk).
SVNameFormat		Mandatory	The format of the identifier for the underlying StorageVolume (or LogicalDisk).
SVExtentStatus		Mandatory	The ExtentStatus as reported in the underlying StorageVolume (or LogicalDisk).
SVOperationalStatus		Mandatory	The OperationalStatus as reported in the underlying StorageVolume (or LogicalDisk).
SVBlockSize		Mandatory	
SVNumberOfBlocks		Mandatory	The number of blocks that make up the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVConsumableBlocks		Mandatory	The number of usable blocks in the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVIsBasedOnUnderlyingRedundancy		Mandatory	Whether or not redundancy is supported for the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVNoSinglePointOfFailure		Mandatory	Whether or not NoSinglePointOfFailure is supported for the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVDataRedundancy		Mandatory	The DataRedundancy supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).

Table 64: SMI Referenced Properties/Methods for SNIA_VolumeView

Properties	Flags	Requirement	Description & Notes
SVPackageRedundancy		Mandatory	The PackageRedundancy supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVDeltaReservation		Mandatory	The DeltaReservation supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).
SSInstanceID		Mandatory	The InstanceID of the StorageSetting for the volume as reported in its underlying StorageSetting.
SSElementName		Mandatory	The ElementName of the StorageSetting for the volume as reported in its underlying StorageSetting.
SSNoSinglePointOfFailure		Mandatory	Whether or not NoSinglePointOfFailure was requested in the StorageSetting for the volume as reported in its underlying StorageSetting.
SSDataRedundancyMin		Mandatory	The DataRedundancyMin value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSDataRedundancyMax		Mandatory	The DataRedundancyMax value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSDataRedundancyGoal		Mandatory	The DataRedundancyGoal supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSPackageRedundancyMin		Mandatory	The PackageRedundancyMin value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSPackageRedundancyMax		Mandatory	The PackageRedundancyMax value supported by the StorageSetting for the volume as reported in the underlying StorageSetting.
SSPackageRedundancyGoal		Mandatory	The PackageRedundancyGoal supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSChangeableType		Mandatory	The ChangeableType defined for the StorageSetting for the volume as reported in the underlying StorageSetting.
AFSPSpaceConsumed		Mandatory	The SpaceConsumed from the StoragePool by the volume as reported in its underlying AllocatedFromStoragePool association to the StoragePool.
SPIInstanceID		Mandatory	The InstanceID of the StoragePool for the volume as reported in the underlying StoragePool.
SPPoolID		Mandatory	The PoolID of the StoragePool for the volume as reported in the underlying StoragePool.
SVOtherIdentifyingInfo		Optional	Other identifiers for the StorageVolume (or LogicalDisk) as reported in the underlying StorageVolume (or LogicalDisk).

Table 64: SMI Referenced Properties/Methods for SNIA_VolumeView

Properties	Flags	Requirement	Description & Notes
SVIdentifyingDescriptions		Optional	The description of the other identifiers for the StorageVolume (or LogicalDisk) as reported in the underlying StorageVolume (or LogicalDisk).
SVElementName		Optional	The user friendly name for the underlying StorageVolume (or LogicalDisk).
SVUsage		Optional	The Usage supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVOtherUsageDescription		Optional	The OtherUsageDescription supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).
SVClientSettableUsage		Optional	The ClientSettableUsage supported by the volume as reported in the underlying StorageVolume (or LogicalDisk).
SSExtentStripeLength		Optional	The ExtentStripeLength value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSExtentStripeLengthMin		Optional	The ExtentStripeLengthMin value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSExtentStripeLengthMax		Optional	The ExtentStripeLengthMax supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSParityLayout		Optional	The ParityLayout defined by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSUserDataStripeDepth		Optional	The UserDataStripeDepth value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSUserDataStripeDepthMin		Optional	The UserDataStripeDepthMin value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSUserDataStripeDepthMax		Optional	The UserDataStripeDepthMax value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.
SSStorageExtentInitialUsage		Optional	The StorageExtentInitialUsage value supported by the StorageSetting for the volume as reported in its underlying StorageSetting.

6.8.17 SNIA_SystemDeviceView (DiskDriveViews)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: DiskDriveView

Table 65 describes class SNIA_SystemDeviceView (DiskDriveViews).

Table 65: SMI Referenced Properties/Methods for SNIA_SystemDeviceView (DiskDriveViews)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this DiskDriveView instance.
PartComponent		Mandatory	The SNIA_DiskDriveView instance that is a device on the computer system.

6.8.18 SNIA_SystemDeviceView (VolumeViews)

Created By: External

Modified By: Static

Deleted By: External

Class Mandatory: VolumeView

Table 66 describes class SNIA_SystemDeviceView (VolumeViews).

Table 66: SMI Referenced Properties/Methods for SNIA_SystemDeviceView (VolumeViews)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	The Computer System that contains this VolumeView instance.
PartComponent		Mandatory	The SNIA_VolumeView instance that is a device on the computer system.

EXPERIMENTAL

STABLE**Clause 7: Block Server Performance Subprofile****7.1 Description****7.1.1 Synopsis**

Profile Name: Block Server Performance

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.11

Central Class: BlockStatisticsService

Scoping Class: ComputerSystem

7.1.2 Overview

The Block Server Performance Subprofile defines classes and methods for managing performance information in block servers (e.g., Arrays, Storage Virtualizers and Volume Management). Not all of the objects for which statistics are defined apply to all these profiles. For example, Storage Virtualizers don't have Disk Drives and Volume Management Profiles don't have Ports. In these cases, the profile would not support the statistics for the object that does not apply to it.

Note: Performance analysis is broader than just Arrays, Storage Virtualizers and Volume Managers. Complete analysis requires performance information from hosts and fabric. These are (or will be) addressed separately as part of the appropriate profiles.

One of the key SRM disciplines for managing block servers (e.g., arrays) is Performance Management. Currently, there are no common statistics defined that can be used to manage multiple vendor arrays from a performance perspective. Some of the key tasks commonly performed in the discipline of Performance Management are:

- Performance Capacity Planning,
- Performance Problem Isolation,
- Peak Window Analysis,
- Block server Workload Analysis,
- Block server Performance Tuning.

In order to manage performance, a number of processes need to be in place:

- Ability to measure the performance and saturation points of components within the storage network. This subprofile describes the first increment of measurement, that of the storage system. Examples of this include:
 - Read and Write I/O counts for a LUN or a disk,
 - Number of blocks transferred per unit time,
 - Cache hit ratios.

Both specific measurements and methods to make these measurements available to SRM applications will be part of this subprofile.

- Ability to understand the relationship of facilities within the storage network and their relationship to the actual application: This is provided by mapping functions which are described in the standard SMI specification. Mapping functions are listed within the specification today. As new objects (like cache which is currently not defined) and new relationships between objects are defined, these parts of the SMI specification will have to be upgraded;
- Ability to understand the status and configuration of the storage network components: There is some level of this information within the SMI specification today, and there are expected future improvements to this area that will be in future releases. Examples of this include:
 - Cache status on or off for read or write cache,
 - How much Cache is installed,
 - Storage Volume (LUN) status, normal or degraded,
 - Cache configuration parameters,
 - LUN status,
 - Error counts on a port.

Methods to be able to tune the configuration of a storage network component. This would include setting RAID levels, setting stripe widths, setting cache tunable parameters, etc. This is an area for future development. Given that there is a wide diversity of storage architectures, this may be an area where SMI provides a framework and vendors supply the custom extensions required for their systems.

Performance Management is optimized when all four components are in place. Performance Measurement is the key deliverable that is the focus of this subprofile.

Block storage devices usually have one or more of the following elements:

- Block Server (Top level ComputerSystem),
- I/O Ports (e.g., FCPorts),
- Front-end Ports,
- Back-end Ports,

Note: Port Statistics in block servers need to be coordinated with Port statistics in the Fabric Profile by applications.

- Individual Controllers (ComponentCS),
- Front-end controller(s) (ComponentCS),
- Back-end controller(s) (ComponentCS),
- Exported Elements (e.g., Volumes or Logical Disks),
- Imported Elements (e.g., Extents with ConcreteComponent association to Pools),
- Disk Drives.

In order to monitor and manage these components, it is necessary to identify performance counters for each of the above elements in the block server and externalize an interface to obtain these counters at some SRM-determined periodicity. An SRM product will also need to be able to associate these counters to the appropriate block server elements as defined in the appropriate SMI-S profiles in order to complete the full picture of the performance analysis (e.g., what disks are part of this LUN and what other LUNs have portions on this disk).

The function of this subprofile is to support the aforementioned SRM applications.

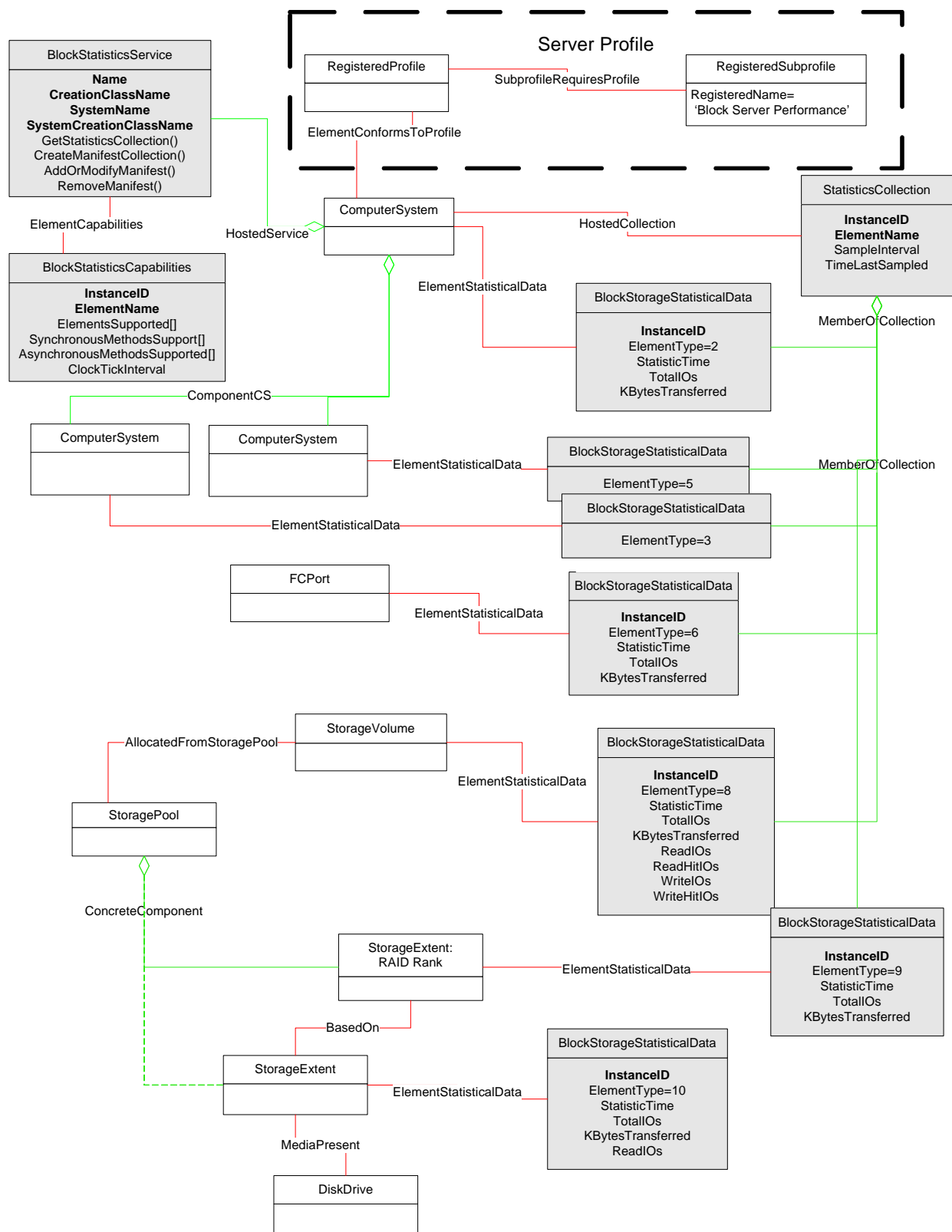
The Block Server Performance Subprofile augments the profiles and subprofiles for Arrays, Storage Virtualizers and Volume Management Profiles. Instead of being an isolated subprofile, it adds modeling constructs to existing profiles and subprofiles. Together these enhancements make up the Block Server Performance Subprofile (as would be registered in the Server Profile as a RegisteredSubprofile).

7.2 Implementation

7.2.1 Performance Additions Overview

Figure 29 provides an overview of the model (independent of profiles and subprofiles). The new classes added by the Block Server Performance Subprofile are the shaded grey boxes.

Figure 29: Block Server Performance Subprofile Summary Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

What this figure shows is a single instance of `StatisticsCollection` for the entire profile. This is the anchor point from which all statistics being kept by the profile can be found. Block statistics are defined as a `BlockStorageStatisticalData` class, instances of which hold the statistics for particular elements (e.g., `StorageVolumes`, `ComputerSystems`, `Ports`, `Extents` and `Disk Drives`). The type of element is recorded in the instance of `BlockStorageStatisticalData` in the `ElementType` property.

All the statistics instances are related to the elements they meter via the `ElementStatisticalData` association (e.g., `BlockStorageStatisticalData` for a `StorageVolume` can be found from the `Volume` by traversing the `ElementStatisticalData` association).

All the statistics instances kept in the profile are associated to the one `StatisticsCollection` instance. Access to all the statistics for the profile is through the `StatisticsCollection`. The `StatisticsCollection` has a `HostedCollection` association to the "top level" computer system of the profile.

Note that statistics may be kept for a number of elements in the profile, including elements in subprofiles. The elements that are metered are:

The Top Level ComputerSystem – This provides a summary of all statistics for the whole profile (e.g., `ReadIOs` are all read IOs handled by the array, storage virtualizer or volume manager).

Component ComputerSystems – This provides a summary of all statistics that derive from a particular processor in the system cluster (e.g., all `ReadIOs` handled by a particular processor). These statistics are kept in `BlockStorageStatisticalData` instances (one for each component computer system).

Port – This provides a summary of all the statistics that derive from a particular Port on the Array or Storage Virtualizer (e.g., all `ReadIOs` that go through the particular port). These statistics are kept in `BlockStorageStatisticalData` instances (one for each Port in the system).

Note: This element does not apply to the Volume Management Profile. Volume managers do not have front-end ports. The back-end ports for volume managers are HBAs. Statistics for volume manager back end ports would be kept by the HBAs.

StorageVolume – (or `LogicalDisk`). This provides a summary of statistics for a particular `StorageVolume` (or `LogicalDisk`). For example, all the `ReadIOs` to the particular `StorageVolume` (or `LogicalDisk`). These statistics are kept in `BlockStorageStatisticalData` instances (one for each `StorageVolume` or `LogicalDisk` in the system).

StorageExtent – This provides a summary of statistics that derive from access to a particular `StorageExtent`. Note: `StorageExtent` support is ONLY PROVIDED for extents with a `ConcreteComponent` association to a concrete `StoragePool`. That is, this is not offered for intermediate extents. These statistics are kept in `BlockStorageStatisticalData` instances (one for each Extent that is modeled in the system).

SCSI Arbitrary Logical Units – This provides summary of statistics that derive from access to LUNs that are not `StorageVolumes` (e.g., controller commands).

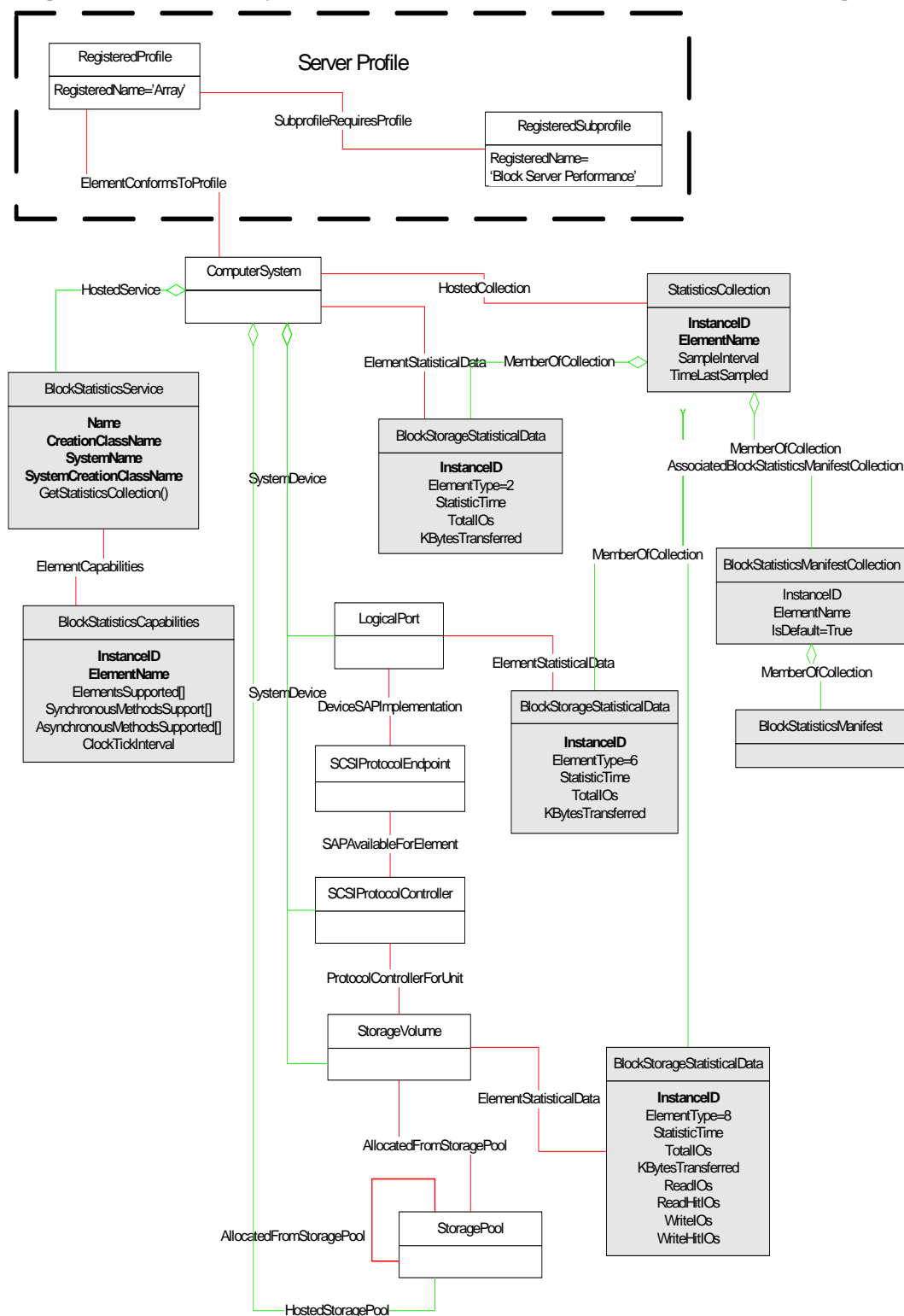
Finally, Figure 30 illustrates the `BlockStatisticsService` for Bulk retrieval of all the statistics data and creation of manifest collections. These methods will be discussed later. They are shown here for completeness. Associated with the `BlockStatisticsService` is a `BlockStatisticsCapabilities` instance that identifies the specific capabilities implemented by the performance support. Specifically, it includes an "ElementsSupported" property that identifies the elements for which statistics are kept and the various retrieval mechanisms that are implemented (e.g., `Extrinsic`, `Association Traversal`, `Indications` and/or `Query`).

7.2.2 Performance Additions to base Array Profile

Figure 30 illustrates the class instances that would be supported if an Array only implemented the base Array Profile and the Block Server Performance Subprofile. Only the `StatisticsCollection`, the `BlockStorageStatisticalData`

instance for the top level computer system, BlockStorageStatisticalData instances for front end ports and BlockStorageStatisticalData instances for Storage Volumes would be supported.

And only the GetStatisticsCollection method of the BlockStatisticsService would be supported. The actual elements for which the statistics would be kept would be reported in the "ElementsSupported" property of the BlockStatisticsCapabilities instance.

Figure 30: Base Array Profile Block Server Performance Instance Diagram

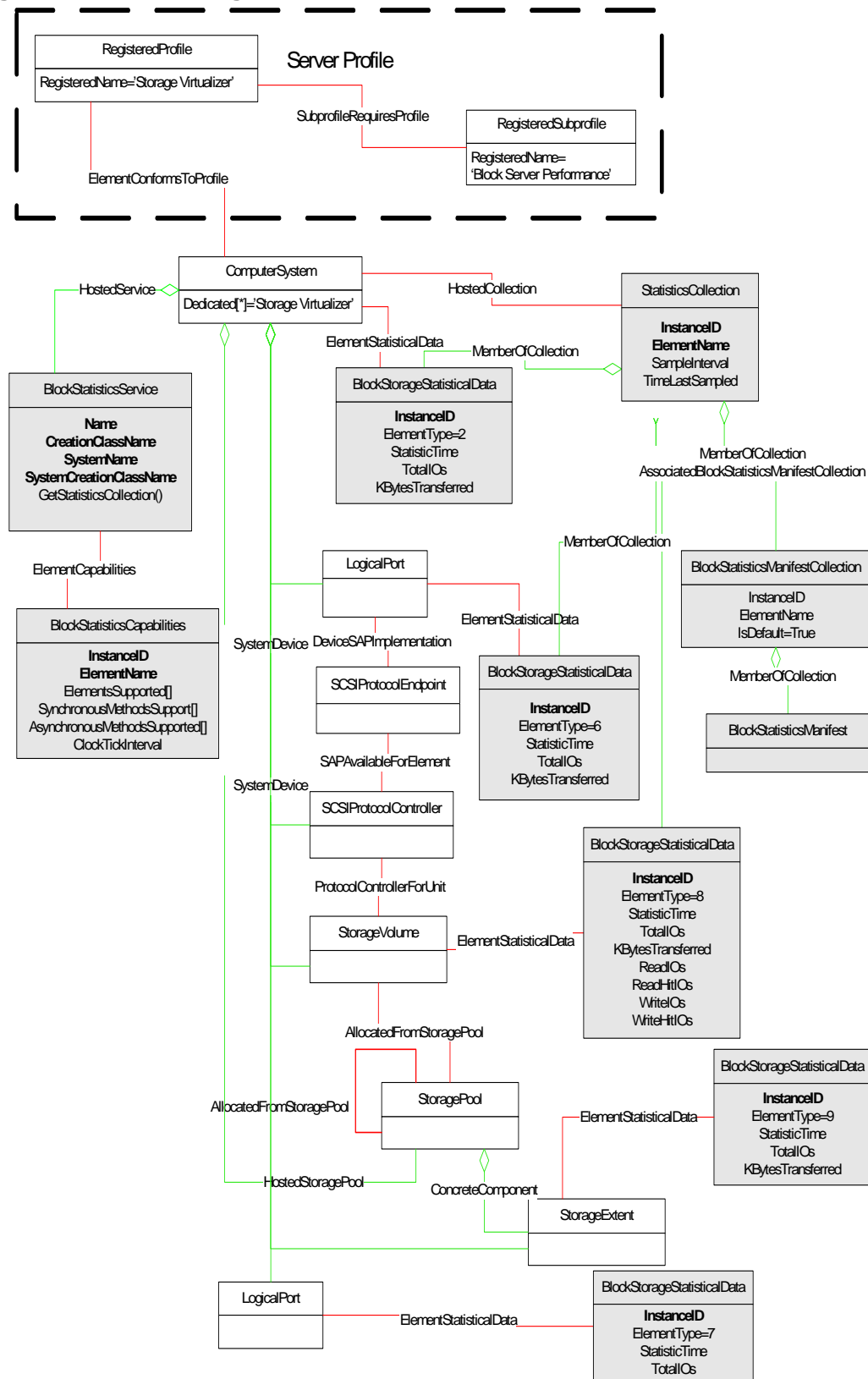
Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

7.2.3 Performance Additions to base Storage Virtualizer Profile

Figure 31 illustrates the class instances that would be supported if a Storage Virtualizer only implemented the base Storage Virtualizer Profile and the Block Server Performance Subprofile. Only the StatisticsCollection, the BlockStorageStatisticalData instance for the top level computer system, BlockStorageStatisticalData instances for front-end and back-end ports, BlockStorageStatisticalData instances for Storage Volumes and BlockStorageStatisticalData for StorageExtents would be supported.

Only the GetStatisticsCollection method of the BlockStatisticsService would be supported. The actual elements for which the statistics would be kept would be reported in the "ElementsSupported" property of the BlockStatisticsCapabilities instance.

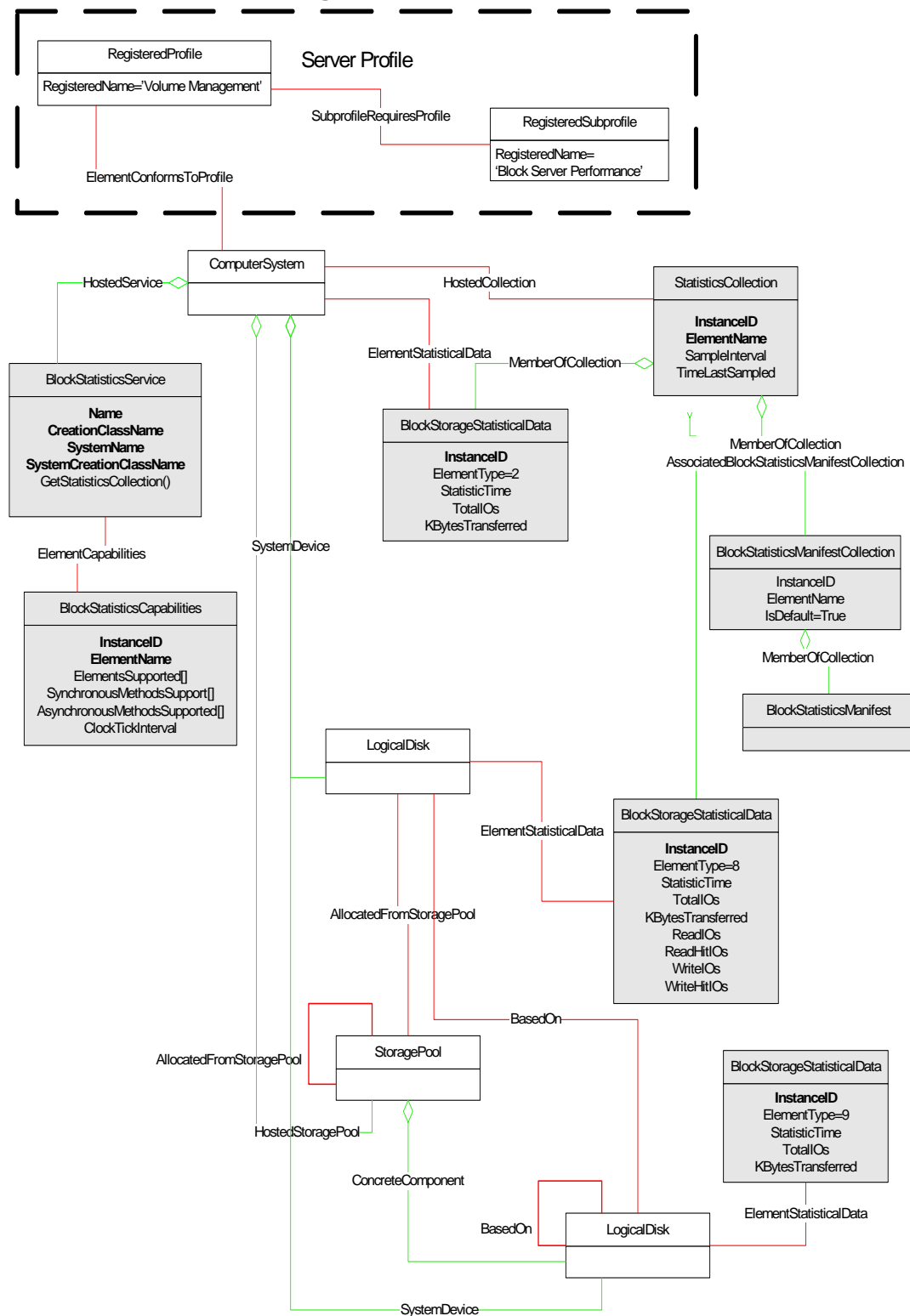
Figure 31: Base Storage Virtualizer Profile Block Server Performance Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

7.2.4 Performance Additions to base Volume Management Profile

Figure 32 illustrates the class instances that would be supported if the volume manager only implemented the base Volume Management Profile and the Block Server Performance Subprofile. Only the StatisticsCollection, the BlockStorageStatisticalData instance for the top level computer system, BlockStorageStatisticalData instances for LogicalDisks (lower extents) and BlockStorageStatisticalData instances for LogicalDisks (exported Logical Disks) would be supported.

Figure 32: Base Volume Management Profile Block Server Performance Instance Diagram

Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

7.2.5 Summary of BlockStorageStatisticsData support by Profile

Table 67 defines the Element Types (for BlockStorageStatisticalData instances) that may be supported by profile.

Table 67: Summary of Element Types by Profile

ElementType	Array	Storage Virtualizer	Volume Management
Computer System	YES	YES	YES
Front-end Computer System	YES	YES	YES
Peer Computer System	YES	YES	YES
Back-end Computer System	YES	YES	YES
Front-end Port	YES	YES	NO
Back-end Port	YES	YES	NO
Volume	YES	YES	YES
Extent	YES	YES	YES
Disk Drive	YES	YES	NO
Arbitrary LUs	YES	YES	NO
Remote Replica Group	YES	YES	YES

YES means that this specification defines the element type for the profile. Actual support by any given implementation would be implementation dependent. But the specification covers defining the element type for the profile. NO means that this specification does not specify this element type for the profile.

7.2.6 Server Profile Support for the Block Server Performance Subprofile

At the top of Figure 30 is a dashed box that illustrates a part of the Server Profile for the Array. A similar dashed box appears for Storage Virtualizer and Volume Management Profiles. The part illustrated is the particulars for the Block Server Performance Subprofile. If performance support has been implemented, then there shall be a RegisteredSubprofile instance for the Block Server Performance Subprofile.

7.2.7 Default Manifest Collection

Associated with the instance of the StatisticsCollection shall be a provider supplied (Default) CIM_BlockStatisticsManifestCollection that represents the statistics properties that are kept by the profile. The default manifest collection is indicated by the IsDefault property (=True) of the CIM_BlockStatisticsManifestCollection. For each metered object of the profile implementation the default manifest collection will have exactly one manifest that will identify which properties are included for that metered object. If an object is not metered, then there shall not be a manifest for that element type. If an element type (e.g., StorageVolume) is metered, then there shall be a manifest for that element type.

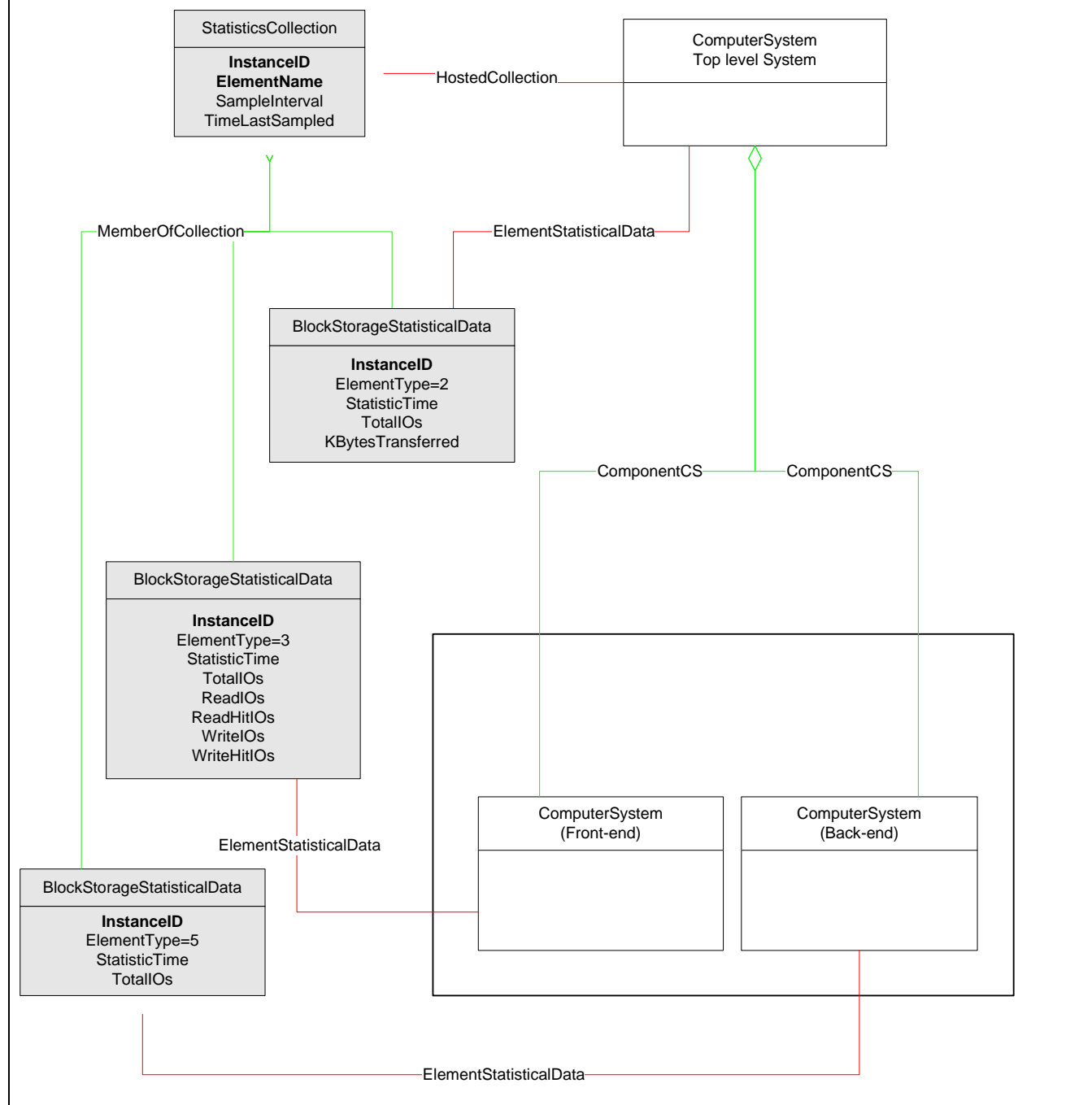
7.2.8 Performance Additions applied to Multiple Computer Systems

Figure 33 illustrates the class instances that would be supported if an Array, Storage Virtualizer or Volume Management Profile also implemented the Multiple Computer System Subprofile (and the Block Server Performance Subprofile). In this case, additional BlockStorageStatisticalData instances would exist for the component computer systems, as well as the top level computer system.

The “ElementsSupported” property of the BlockStatisticsCapabilities instance would include “Front-end Computer System”, “Back-end Computer System” and/or “Peer Computer System”.

Note: Support for both the Multiple Computer System Subprofile and the Block Server Performance Subprofile does not imply support for statistics at the Component Computer System level. This support is ONLY implied by the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

Figure 33: Multiple Computer System Subprofile Block Server Performance Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

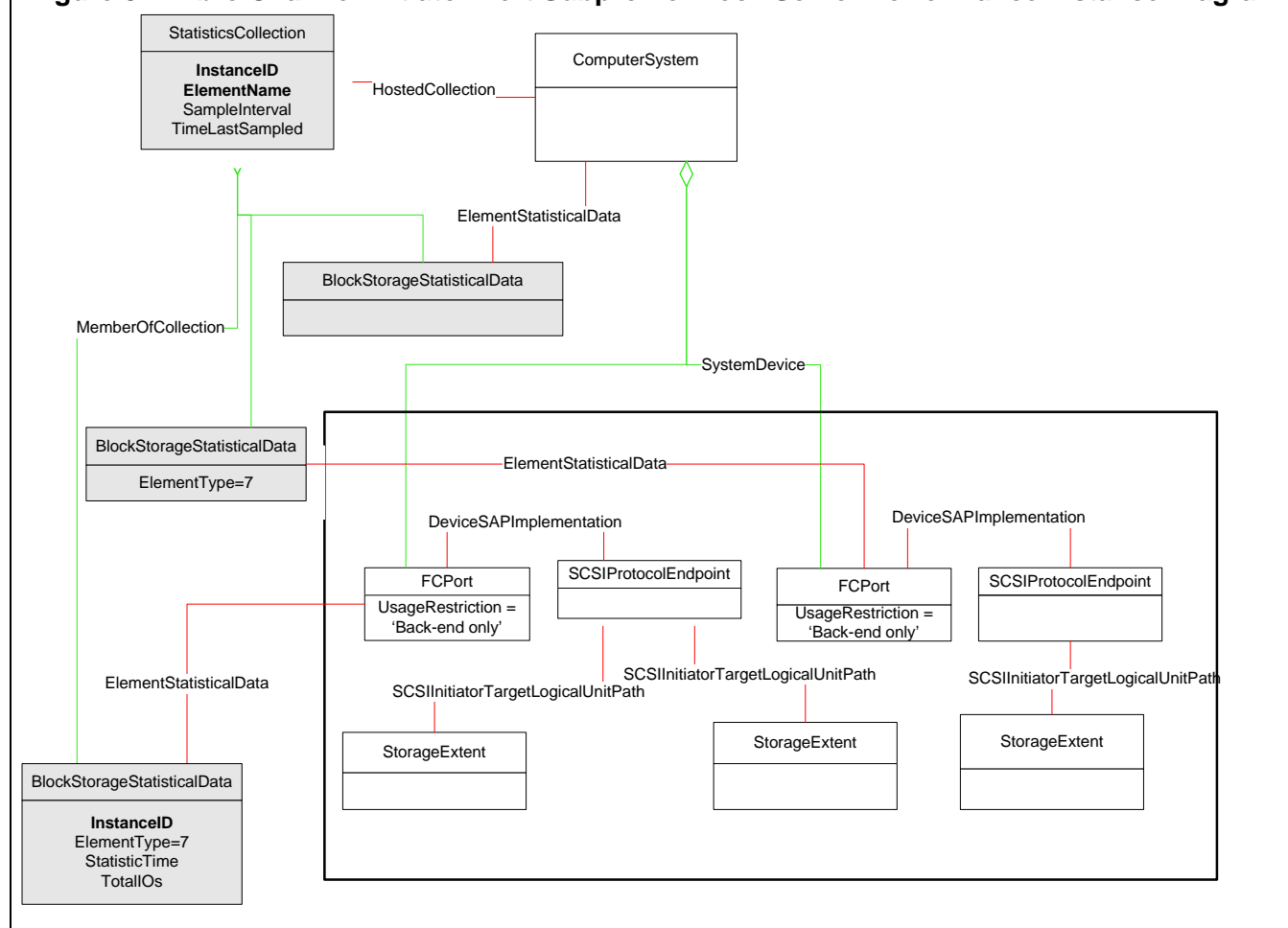
7.2.9 Performance Additions to Backend Ports

Figure 34 illustrates the class instances that would be supported if an Array also implemented the Fibre Channel Initiator Port Subprofile (and the Block Server Performance Subprofile). In this case, additional BlockStorageStatisticalData instances would exist for the back-end ports, as well as the front-end ports.

The “ElementsSupported” property of the BlockStatisticsCapabilities instance would include “Back-end Ports”.

Note: Support for both the Fibre Channel Initiator Port Subprofile and the Block Server Performance Subprofile DOES not imply support for statistics at the Back-end Port level. This support is ONLY implied by the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

Figure 34: Fibre Channel Initiator Port Subprofile Block Server Performance Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

7.2.10 Performance Additions to Extent Composition

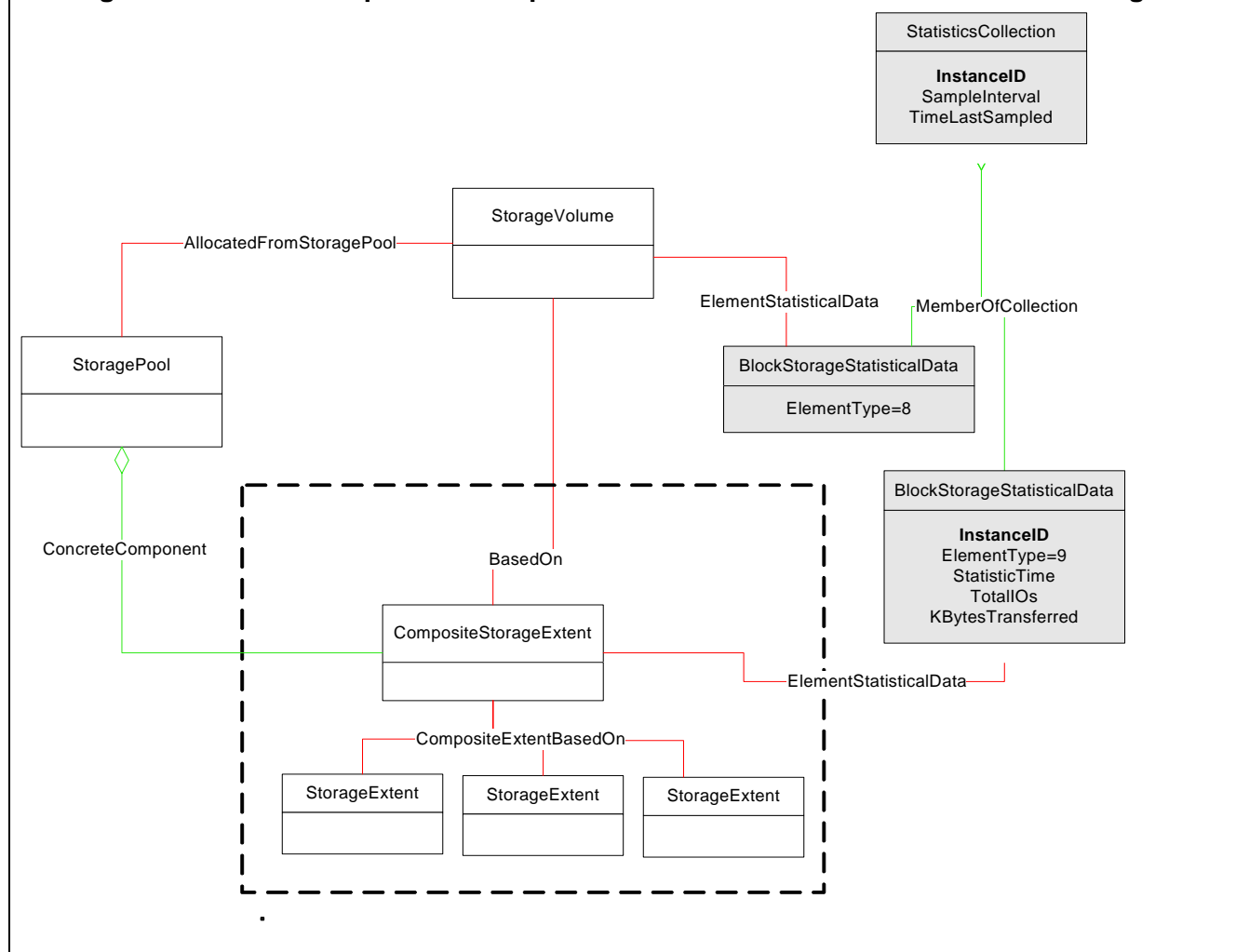
Figure 35 illustrates the class instances that would be supported if an Array also implemented the Extent Composition Subprofile (and the Block Server Performance Subprofile). In this case, BlockStorageStatisticalData instances would exist for the Extents that are modeled.

The “ElementsSupported” property of the BlockStatisticsCapabilities instance would include “Extents”.

Note: The Storage Virtualizer and Volume Management Profiles would use the “Extents” statistics for Storage Volumes (or LogicalDisks) that are imported instead of Disk extent statistics (since they do not have disk drives). Also note that an Array may model both “Extents” and “Disks” extents.

Note: Support for both the Extent Composition Subprofile and the Block Server Performance Subprofile DOES not imply support for statistics at the Extent level. This support is ONLY implied by the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

Figure 35: Extent Composition Subprofile Block Server Performance Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

Note: The low level extents represent Disk Drive Extents and they would not be part of the Storage Virtualizer or Volume Management Profiles.

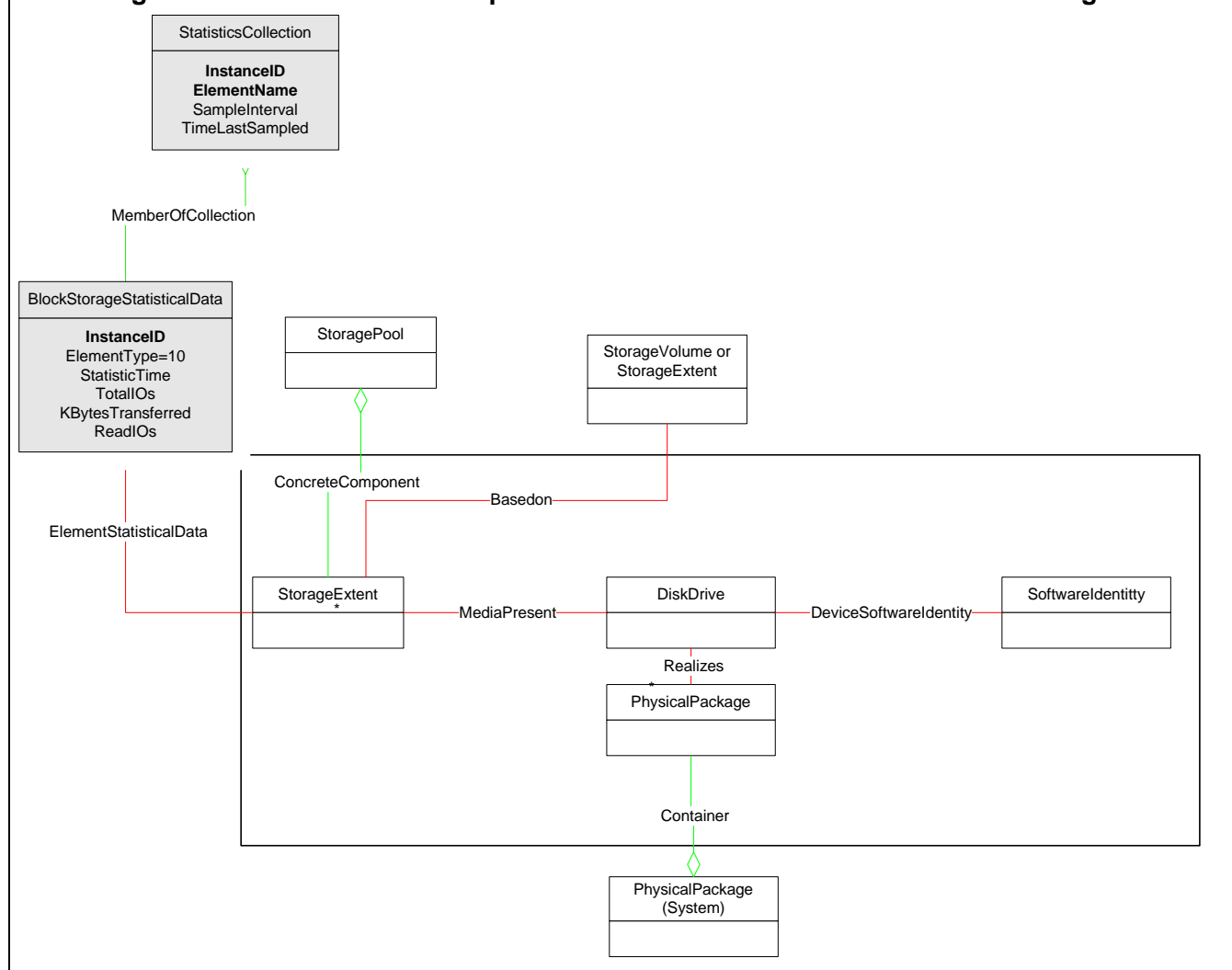
7.2.11 Performance Additions to Disk Drives

Figure 36 illustrates the class instances that would be supported if an Array also implemented the Disk Drive Lite (or Disk Drive) Subprofile (and the Block Server Performance Subprofile). In this case, **BlockStorageStatisticalData** instances would exist for each of the Disk Drives in the Array.

The “ElementsSupported” property of the **BlockStatisticsCapabilities** instance would include “Disks”.

Note: The Volume Management Profiles would NEVER show the “Disks” statistics. Also note that an Array or Storage Virtualizer may model both “Extents” and “Disks”. Note: Support for both the Disk Drive Lite Subprofile and the Block Server Performance Subprofile DOES not imply support for statistics at the Disk Drive level. This support is ONLY implied by the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

Figure 36: Disk Drive Lite Subprofile Block Server Performance Instance Diagram



Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, "CIM Elements".

7.2.12 Performance Additions to SCSIArbitraryLogicalUnits (Controller LUNs)

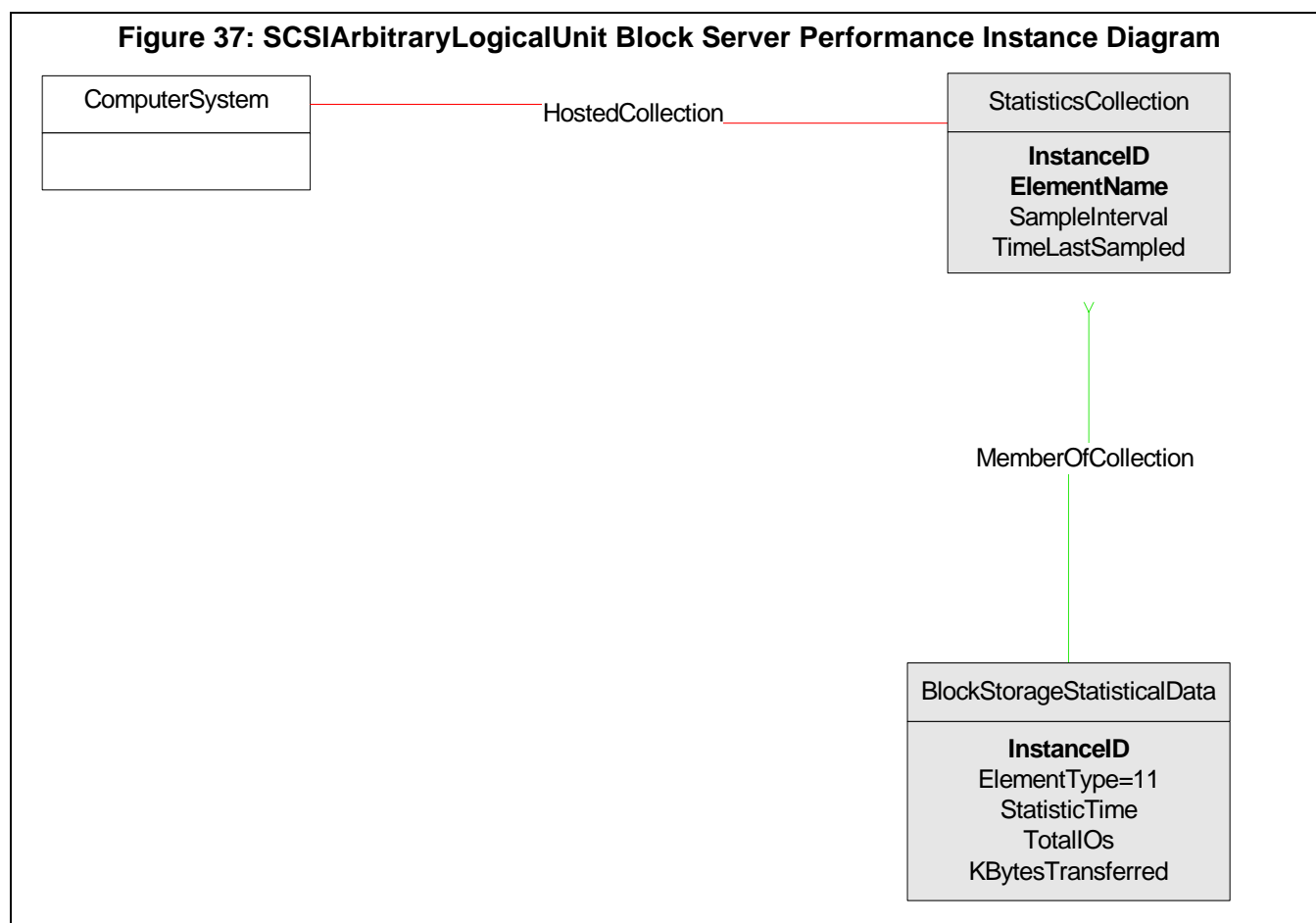


Figure 37 illustrates the class instances that would be supported if an Array (or Storage Virtualizer) has Controller LUNs (e.g., SCSIArbitraryLogicalUnits). In this case, BlockStorageStatisticalData instances would exist for each of the Controller LUNs (LogicalDevices or SCSIArbitraryLogicalUnits) supported by the Array (or Storage Virtualizer).

Note: There is no ElementStatisticalData association to any element. This is because the Controller LUNs are not actually part of the Array or Storage Virtualizer Profiles. But the statistics may still be collected in and kept in BlockStorageStatisticalData instances with ElementType=11.

The “ElementsSupported” property of the BlockStatisticsCapabilities instance would include “Arbitrary LUs”.

Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, “CIM Elements”.

EXPERIMENTAL

7.2.13 Performance Additions for Remote Mirrors

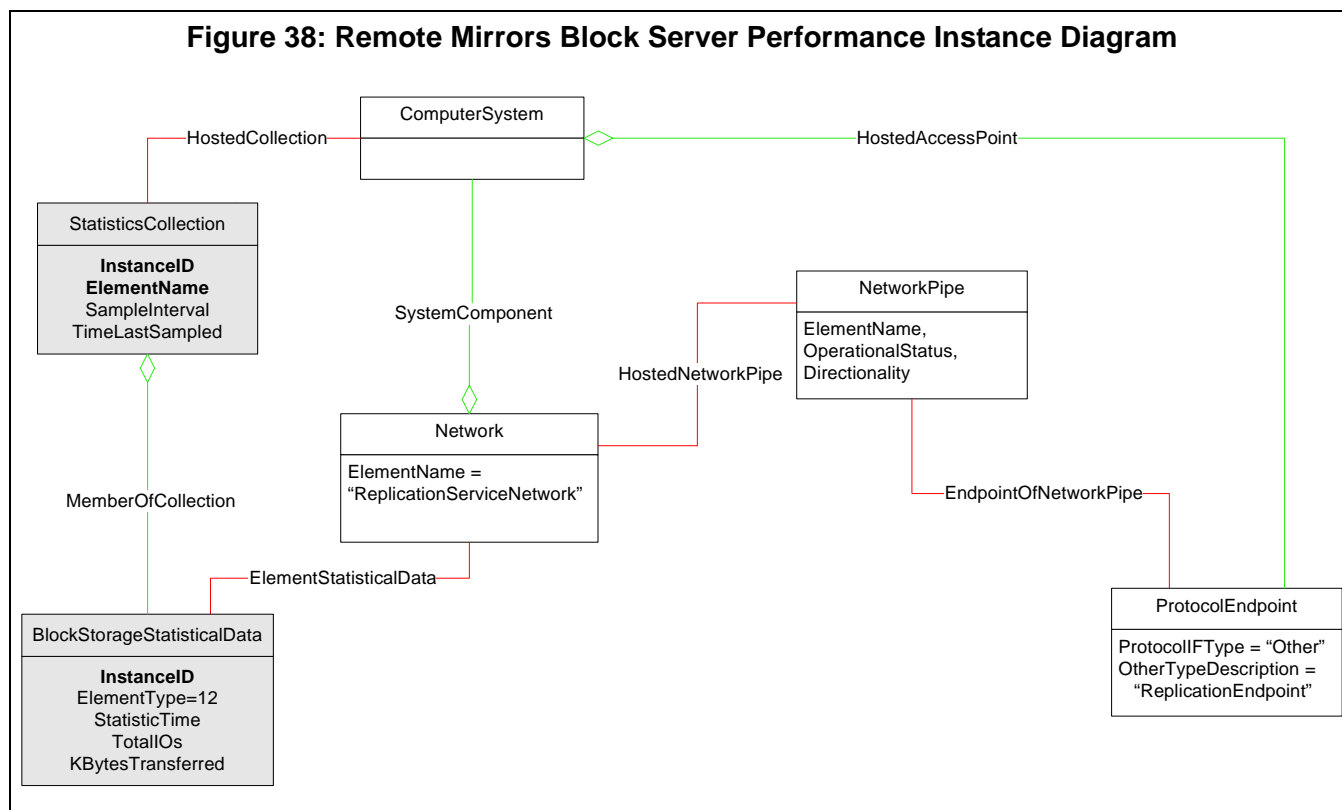


Figure 38 illustrates the class instances that would be supported if an Array also implemented the Remote Mirroring of the Copy Services Subprofile (and the Block Server Performance Subprofile). In this case, BlockStorageStatisticalData instances would exist for non-volume (e.g., meta data) IO requests. In this case, the BlockStorageStatisticalData instance is associated with the Network instance that represents the connection to the remote system. **Note:** Statistics attributed to the Network are control IOs between the mirroring arrays. Statistics that actually move data to the remote mirror are attributed to the targeted StorageVolume (or logical disk).

The “ElementsSupported” property of the BlockStatisticsCapabilities instance would include “Remote Replica Group”.

Note: Support for both the Copy Services Subprofile and the Block Server Performance Subprofile DOES not imply support for statistics at the Remote Replica Group level. This support is ONLY implied by the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

Note: The properties listed for the statistics classes are the mandatory properties. Optional Properties are not listed in order to save space in the diagram. Optional properties can be found in 7.9, “CIM Elements”.

EXPERIMENTAL

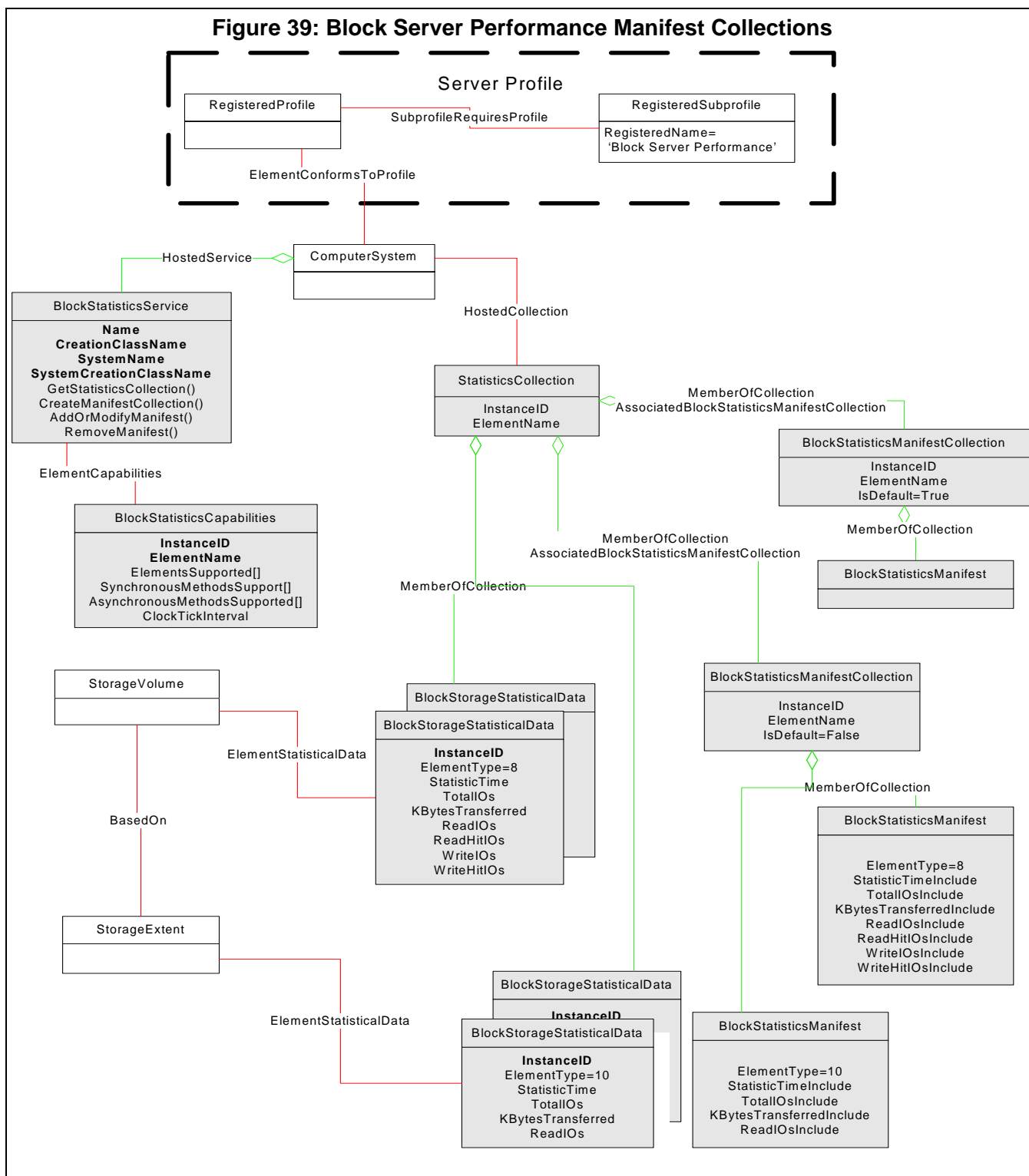
7.2.14 Client Defined Manifest Collections

Manifest collections are either provider supplied (CIM_BlockStatisticsManifestCollection.IsDefault=True) for the profile implementation or client defined collections (CIM_BlockStatisticsManifestCollection.IsDefault=False) that

indicate what statistics properties the client would like to retrieve using the `GetStatisticsCollection` method. For a discussion of provider supplied manifest collections, see 7.2.7.

Client defined manifest collections are a mechanism for restricting the amount of data returned on a `GetStatisticsCollection` request. A client defined manifest collection is identified by the `IsDefault` property of the collection is set to `False`. For each block statistics class (e.g., Computer System, Volume, Disk, etc.) a manifest can be defined which identifies which properties of the particular statistics class are to be returned on a `GetStatisticsCollection` request. Each of the classes of block statistic may have 0 or 1 manifest in any given manifest collection. This is illustrated in Figure 39.

Figure 39: Block Server Performance Manifest Collections



In this figure, manifest classes are defined for Volumes (StorageVolumes or LogicalDisks) and Disk Drives. Each property of the manifest is a Boolean that indicates whether the property is to be returned (true) or omitted (false).

Multiple client defined manifest collections can be defined in the profile. So different clients or different client applications can define different manifests for different application needs. A manifest collection can completely omit a whole class of statistics (e.g., no ComputerSystem statistics are shown in Figure 39). Since manifest collections are “client objects”, they are named (ElementName) by the client for the client’s convenience. The CIM server will generate an instance ID to uniquely identify the manifest collection in the CIM Server.

Client defined manifest collections are created using the CreateManifestCollection method. Manifests are added or modified using the AddOrModifyManifest method. And a manifest may be removed from the manifest collection using the RemoveManifest method.

Note: Use of manifest collections is optional with the GetStatisticsCollection method. If NULL for the manifest collection is passed on input, then all statistics instances are assumed.

7.2.15 Capabilities Support for Block Server Performance Subprofile

There are two dimensions to determining what is supported with a Block Server Performance Subprofile implementation. First, there are the RegisteredSubprofiles supported by the Block server (Array, Storage Virtualizer or Volume Management Profile). In order to support statistics for a particular class of metered element, the corresponding object shall be modeled. So, if an Array has not implemented the Disk Drive Lite (or Disk Drive) Subprofile, then it shall not implement the BlockStorageStatisticalData for Disk Drives in the Block Server Performance Subprofile (and implementation of the Disk Drive Lite or Disk Drive Subprofile does not guarantee implementation of the BlockStorageStatisticalData for disk drives).

Both of these dimensions are captured in the BlockStatisticsCapabilities class instance. This is populated by the provider (not created or modified by Clients) and it has three properties of interest. The second dimension is techniques supported for retrieving statistics and manipulating manifest collections.

7.2.15.1 ElementsSupported

The values of interest are “Computer System”, “Front-end Computer System”, “Peer Computer System”, “Back-end Computer System”, “Front-end Port”, “Back-end Port”, “Volume”, “Extent”, “Disk Drive”, “Arbitrary LUs”, “Remote Replica Group”

7.2.15.2 SynchronousMethodsSupported

The values of interest are “Exec Query”, “Indications”, “Query Collection”, “GetStatisticsCollection”, “Manifest Creation”, “Manifest Modification”, and “Manifest Removal”

7.2.15.3 AsynchronousMethodsSupported

For the current version of SMI-S this should be NULL.

7.2.15.4 ClockTickInterval

An internal clocking interval for all timer counters kept in the subsystem, measured in microseconds (Unit of measure in the timers, measured in microseconds). Time counters are monotonically increasing counters that contain 'ticks'. Each tick represents one ClockTickInterval.

To be a valid implementation of the Block Server Performance Subprofile, at least one of the values listed for ElementsSupported shall be supported. ElementsSupported is an array, such that all of the values can be identified.

For the methods supported properties any or all of these values can be missing (e.g., the arrays can be NULL). If all the methods supported are NULL, this means that manifest collections are not supported and neither GetStatisticsCollection nor Query are supported for retrieval of statistics. This leaves enumerations or association traversals as the only methods for retrieving the statistics.

7.3 Health and Fault Management Considerations

Not defined in this standard.

7.4 Cascading Considerations

Not applicable.

7.5 Supported Subprofiles and Packages

Table 68: Supported Profiles for Block Server Performance

Registered Profile Names	Mandatory	Version
Multiple Computer System	No	1.2.0
Extent Composition	No	1.2.0
Generic Target Ports	No	1.2.0
SPI Target Ports	No	1.2.0
FC Target Ports	No	1.2.0
iSCSI Target Ports	No	1.2.0
DA Target Ports	No	1.2.0
Generic Initiator Ports	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
Disk Drive Lite	No	1.2.0
Copy Services	No	1.2.0

Note: Each of these subprofiles is mandatory if the element in question is to be metered. For example, in order to keep statistics on Disk Drives, it will be necessary for Disk Drives to be modeled.

7.6 Methods of the Profile

7.6.1 Extrinsic Methods of the Profile

7.6.1.1 Overview

The methods supported by this subprofile are summarized in Table 69, and detailed in the sections that follow it.

Table 69: Creation, Deletion and Modification Methods in Block Server Performance Subprofile

Method	Created Instances	Deleted Instances	Modified Instances
GetStatisticsCollection	None	None	None

Table 69: Creation, Deletion and Modification Methods in Block Server Performance Subprofile

Method	Created Instances	Deleted Instances	Modified Instances
CreateManifestCollection	BlockStatisticsManifestCollection AssociatedBlockStatisticsManifestCollection	None	None
AddOrModifyManifest	BlockStatisticsManifest (subclass) MemberOfCollection	None	BlockStatisticsManifest (subclass)
RemoveManifest	None	BlockStatisticsManifest (subclass) MemberOfCollection	None

7.6.1.2 GetStatisticsCollection

This method retrieves statistics in a well-defined bulk format. The set of statistics returned by this list is determined by the list of element types passed in to the method and the manifests for those types contained in the supplied manifest collection. The statistics are returned through a well-defined array of strings that can be parsed to retrieve the desired statistics as well as limited information about the elements that those metrics describe.

GetStatisticsCollection(

[IN (false), OUT, Description(Reference to the job (shall be null in the current version of SMI-S).)]

CIM_ConcreteJob REF **Job**,

[IN, Description(Element types for which statistics should be returned.)

ValueMap { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "..", "32768..65535" },

Values { "Unknown", "Reserved", "Computer System", "Front-end Computer System",
"Peer Computer System", « Back-end Computer System" "Front-end Port", "Back-end Port",
"Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group",
"DMTF Reserved", "Vendor Specific" }}

uint16 **ElementTypes**[],

[IN, Description(The manifest collection that contains the manifests that list the metrics that should be returned for each element type.)]

CIM_BlockStatisticsManifestCollection REF **ManifestCollection**,

[IN, Description("Specifies the format of the Statistics output parameter.")

ValueMap { "2" },

Values ("CSV")]

UInt16 **StatisticsFormat**,

[OUT, Description(The statistics for all the elements as determined by the Elements and

ManifestCollection parameters.))

string **Statistics[]**);

Error returns are:

{ "Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Method Reserved", "Method Parameters Checked - Job Started", "Element Not Supported", "Statistics Format Not Supported", "Method Reserved", "Vendor Specific" }

Note: In this version of the standard, Job Control is not supported for the GetStatisticsCollection method. This method should always return NULL for the Job parameter.

If the ElementTypes[] array is empty, then no data is returned. If the ElementTypes[] array is NULL, then all data specified in the manifest collection is returned.

If the manifest collection is empty, then no data is returned. If the manifest collection parameter is NULL, then the default manifest collection is used (Note: In SMI-S, a default manifest collection shall exist if the GetStatisticalCollection method is supported).

Note: The ElementTypes[] and ManifestCollection parameters may identify different sets of element types. The effect of this will be for the implementation to return statistics for the element types that are in both lists (that is, the intersection of the two lists). This intersection could be empty. In this case, no data will be returned.

For the current version of SMI-S, the only recognized value for StatisticsFormat is "CSV". The method may support other values, but they are not specified by SMI-S (i.e., they would be vendor specific).

Given a client has an inventory of the metered objects with Statistics InstanceIDs that may be used to correlate with the BlockStorageStatisticalData instances, a simple CSV format is sufficient and the most efficient human-readable format for transferring bulk statistics. More specifically, the following rules constrain that format and define the content of the String[] Statistics output parameter to the GetStatisticsCollection() method:

- The Statistics[] array may contain multiple statistics records per array entry. In such cases, the total length of the concatenated record strings will not exceed 64K bytes. And a single statistics record will not span Array entries.
- There shall be exactly one statistics record per line in the bulk Statistics parameter. A line is terminated by:
 - a line-feed character
 - the end of a String Array Element (i.e. a statistics record cannot overlap elements of the String[] Statistics output parameter).
- Each statistics record shall contain the InstanceID of the BlockStorageStatisticalData instance, the value map (number) of the ElementType of the metered object and one value for each property that the relevant BlockStatisticsManifest specifies as "true".
- Each value in a record shall be separated from the next value by a Semi-colon (";"). This is to support internationalization of the CSV format. A provider creating a record in this format should not include white space between values in a record. A client reading a record it has received would ignore white-space between values.
- The InstanceID value is an opaque string that shall correspond to the InstanceID property from BlockStorageStatisticalData instance. The ElementType value shall be a decimal string representation of the Element Type number (e.g., "8" for StorageVolume). The StatisticTime shall be a string representation of DateTime. All other values shall be decimal string representations of their statistical values.
- NULL values shall be included in records for which a statistic is returned (specified by the manifest or by a lack of manifest for a particular element type) but there is no meaningful value available for the statistic. A NULL

statistic is represented by placing a comma in the record without a value in the position the value would have otherwise been included. A record in which the last statistic has a NULL value shall end in a comma.

- The first three values in a record shall be the InstanceID, ElementType and StatisticTime values from the BlockStorageStatisticalData instance. The remaining values shall be returned in the order in which they are defined by the MOF for the BlockStatisticsManifest class or subclass the record describes.

As an additional convention, a provider should return all the records for a particular element type in consecutive String elements, and the order of the element types should be the same as the order in which the element types were specified in the input parameter to GetStatisticsCollection().

Example output as it might be transmitted in CIM-XML. It shows records for 5 Volumes and 5 disks, assuming that 6 statistics were specified in the BlockStatisticsManifest instance for both disks and volumes. The sixth statistic is unavailable for volumes, and the fourth statistic is unavailable for disks:

```
<METHODRESPONSE NAME="GetStatisticsCollection">
  <RETURNVALUE PARAMTYPE="uint32">
    <VALUE>
      0
    </VALUE>
  </RETURNVALUE>
  <PARAMVALUE NAME="Statistics" PARAMTYPE="string">
    <VALUE.ARRAY>
      <VALUE>
        STORAGEVOLUMESTATS1;7;20040811133015.0000010-300;11111;22222;33333;44444;55555;
        STORAGEVOLUMESTATS2;7;20040811133015.0000020-300;11111;22222;33333;44444;55555;
        STORAGEVOLUMESTATS3;7;20040811133015.0000030-300;11111;22222;33333;44444;55555;
        STORAGEVOLUMESTATS4;7;20040811133015.0000040-300;11111;22222;33333;44444;55555;
        STORAGEVOLUMESTATS5;7;20040811133015.0000050-300;11111;22222;33333;44444;55555;
      </VALUE>
      <VALUE>
        DISKSTATS1;9;20040811133015.0000100-300;11111;22222;33333;;55555;66666;
        DISKSTATS2;9;20040811133015.0000110-300;11111;22222;33333;;55555;66666;
        DISKSTATS3;9;20040811133015.0000120-300;11111;22222;33333;;55555;66666;
        DISKSTATS4;9;20040811133015.0000130-300;11111;22222;33333;;55555;66666;
        DISKSTATS5;9;20040811133015.0000140-300;11111;22222;33333;;55555;66666;
      </VALUE>
    </VALUE.ARRAY>
  </PARAMVALUE>
</METHODRESPONSE>
```

7.6.1.3 CreateManifestCollection

Creates a new manifest collection whose members serve as a filter for metrics retrieved through the GetStatisticsCollection method.

CreateManifestCollection(

[IN, Description(The collection of statistics that will be filtered using the new
manifest collection.)]

CIM_StatisticsCollection REF **Statistics**,

[IN, Description(Client-defined name for the new manifest collection)]

string **ElementName**,

[OUT, Description(Reference to the new manifest collection.)]

CIM_BlockStatisticsManifestCollection REF **ManifestCollection**);

Error returns are:

{ "Ok", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Method Reserved", "Vendor Specific" }

7.6.1.4 AddOrModifyManifest

This is an extrinsic method that either creates or modifies a statistics manifest for this statistics service. A client supplies a manifest collection in which the new manifest collection will be placed or an existing manifest will be modified, the element type of the statistics that the manifest will filter, and a list of statistics that should be returned for that element type using the GetStatisticsCollection method.

AddOrModifyManifest(

[IN, Description(Manifest collection that the manifest is or should be a member of.)]

CIM_BlockStatisticsManifestCollection REF **ManifestCollection**,

[IN, Description(The element type whose statistics the manifest will filter.)

ValueMap { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "..", "32768..65535" },

Values { "Unknown", "Reserved", "Computer System", "Front-end Computer System",

"Peer Computer System", « Back-end Computer System" "Front-end Port", "Back-end Port",

"Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group",

"DMTF Reserved", "Vendor Specific" }}

uint16 **ElementType**,

[IN, Description(The client-defined string that identifies the manifest created or modified by this method.)]

string **ElementName**,

[IN, Description(The statistics that will be supplied through the GetStatisticsCollection method.)]

string **StatisticsList**[],

[OUT, Description(The Manifest that is created or modified on successful execution of the method.)]

CIM_BlockStatisticsManifest REF **Manifest**);

Error returns are:

{ "Success", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Method Reserved", "Element Not Supported", "Metric not supported", "ElementType Parameter Missing", "Method Reserved", "Vendor Specific" }

If the `StatisticsList[]` array is empty, then only `InstanceID` and `ElementType` will be returned when the manifest is referenced. If the `StatisticsList[]` array parameter is `NULL`, then all supported properties is assumed

Note: This would be the `BlockStatisticsManifest` from the default manifest collection.

7.6.1.5 RemoveManifest

This is an extrinsic method that removes manifests from a manifest collection.

RemoveManifest(

[IN, Description(Manifest collection from which the manifests will be removed.)]

CIM_BlockStatisticsManifestCollection REF **ManifestCollection**,

[IN, Description(List of manifests to be removed from the manifest collection.)]

CIM_BlockStatisticsManifest REF **Manifests[]**);

Error returns are:

{ "Success", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Method Reserved", "Manifest not found", "Method Reserved", "Vendor Specific" }

7.6.2 Intrinsic Methods of the Profile

Note: Basic Write intrinsic methods are not specified for `StatisticsCollection`, `HostedCollection`, `BlockStorageStatisticalData`, `MemberOfCollection` or `ElementStatisticalData`.

7.6.2.1 DeleteInstance (of a CIM_BlockStatisticsManifestCollection)

This will delete the `CIM_BlockStatisticsManifestCollection` where `IsDefault=False`, the `CIM_AssociatedBlockStatisticsManifestCollection` association to the `StatisticsCollection` and all manifests collected by the manifest collection (and the `MemberOfCollection` associations to the `CIM_BlockStatisticsManifestCollection`).

7.6.2.2 Association Traversal

One of the ways of retrieving statistics is through association traversal from the `StatisticsCollection` to the individual `Statistics` following the `MemberOfCollection` association. This shall be supported by all implementations of the Block Server Performance Subprofile and would be available to clients if the provider does not support `EXEC QUERY` or `GetStatisticsCollection` approaches.

EXPERIMENTAL

7.6.2.3 CreateInstance (of a ListenerDestinationCIMXML, IndicationSubscription and possibly IndicationFilters)

`CreateInstance` would be required to establish subscriptions and `ListenerDestinations` for retrieval of statistics via indications. Depending on the support in the profile, it may also be required to create the `IndicationFilter`.

7.6.2.4 DeleteInstance (of a ListenerDestinationCIMXML, IndicationSubscription and possibly IndicationFilters)

`DeleteInstance` would be required to delete subscriptions and `ListenerDestinations` that were defined for retrieval of statistics via indications. Depending on the support in the profile, it may also be required to delete the `IndicationFilter`.

7.6.2.5 ModifyInstance (of an IndicationFilter)

ModifyInstance may also be supported for modifying IndicationFilters, assuming the profile supports client defined filters. It would not be supported for “pre-defined” filters.

7.6.2.6 EXEC QUERY

This is one of the ways of retrieving statistics.

7.6.2.7 GetInstance on QueryStatisticalCollection

This is yet another means of retrieving statistics. In this technique an instance of the QueryStatisticalCollection class is created that defines a Query for statistics and the format in which the query results are to be represented. The key properties of the QueryStatisticalCollection class are:

- **Query** - This is a query string that defines the statistics to be populated in the QueryStatisticalCollection instance.
- **QueryLanguage** - This defines the query language that is used in the query. For the current version of SMI-S, only CQL should be encoded.
- **SelectedEncoding** - This defines the encoding of the data that is to be populated in the QueryStatisticalCollection instance. For the current version of SMI-S, this should be CSV (for Comma Separated Values).
- **SelectedNames** - This is the list of statistics property names to be retrieved. These correspond to the Select List of the Query. The encoding of these names is as defined by the SelectedEncoding (for the current version of SMI-S, this would be CSV).
- **SelectedTypes** - This is the list of data types for the columns of the query result. Each data type specified corresponds to a column in the SelectedValues property.
- **SelectedValues** - This is a table of values that correspond to the query results (for the query specified in the Query property). The data types of the column of values are defined by SelectedTypes. The name of each column in the table is defined by SelectedNames. And the values are encoded as defined by SelectedEncoding (i.e., CSV for the current version of SMI-S).

An example CQL query would be:

```
SELECT Stats.*
FROM CIM_BlockStorageStatisticalData Stats, CIM_QueryStatisticsCollection
      QSC,
      CIM_MemberOfCollection MoC
WHERE ObjectPath(QSC) = ObjectPath(SELF)
      AND ObjectPath(QSC) = MoC.Collection
      AND ObjectPath(Stats) = MoC.Member
      AND CurrentDateTime() >=
          Stats.StatisticTime + Stats.SampleInterval
```

A client would define a QueryStatisticalCollection instance as means of specifying what the client wants. This would be done with the CreateInstance intrinsic method. The client would delete such an instance using the DeleteInstance method. If the client wishes to change the query, the client would use the ModifyInstance intrinsic method.

Retrieving the data would be done via the `GetInstance` intrinsic. This would retrieve the `QueryStatisticalCollection` instance, which includes the table of comma separated values which are the statistics.

EXPERIMENTAL

7.7 Client Considerations and Recipes

7.7.1 Bulk Performance Statistics Gathering

```
// DESCRIPTION
//
// This recipe describes how to determine what elements are metered, what
// retrieval methods are supported and what statistics are kept for each
// metered element in Arrays, Storage Virtualizers or Volume Managers that
// support the Block Server Performance Subprofile and how to retrieve the
// statistical data.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
// 1. The names of the top-level ComputerSystem instances for Array, Storage
// Virtualizer, or Volume Manager implementations supporting the Block Server
// Performance Subprofile have previously been discovered via SLP and are known
// as $StorageSystems->[].
//

// Function GetNumStatsIncluded
//
// This function counts of the number of metrics that should be included in a
// statistics record built using the supplied BlockManifest instance.
//
sub GetNumStatsIncluded($BlockManifest) {
    #numIncluded = 0
    if ($BlockManifest.IncludeStatisticTime)
        #numIncluded++
    if ($BlockManifest.IncludeTotalIOs)
        #numIncluded++
    if ($BlockManifest.IncludeKBytesTransferred)
        #numIncluded++
    if ($BlockManifest.IncludeIOTimeCounter)
        #numIncluded++
    if ($BlockManifest.IncludeReadIOs)
        #numIncluded++
    if ($BlockManifest.IncludeReadHitIOs)
        #numIncluded++
    if ($BlockManifest.IncludeReadIOTimeCounter)
        #numIncluded++
    if ($BlockManifest.IncludeReadHitIOTimeCounter)
```

Block Server Performance Subprofile

```
#numIncluded++
if ($BlockManifest.IncludeKBytesRead)
    #numIncluded++
if ($BlockManifest.IncludeWriteIOs)
    #numIncluded++
if ($BlockManifest.IncludeWriteHitIOs)
    #numIncluded++
if ($BlockManifest.IncludeWriteIOTimeCounter)
    #numIncluded++
if ($BlockManifest.IncludeWriteHitIOTimeCounter)
    #numIncluded++
if ($BlockManifest.IncludeKBytesWritten)
    #numIncluded++
if ($BlockManifest.IncludeIdleTimeCounter)
    #numIncluded++
if ($BlockManifest.IncludeMaintOp)
    #numIncluded++
if ($BlockManifest.IncludeMaintTimeCounter)
    #numIncluded++
return #numIncluded
}

// Function ValidateRecords
//
// This function validates the records of a set of statistics supplied in the
// Bulk Statistics Format defined in the Block Server Performance Subprofile.
// Every statistics record should contain an InstanceID, ElementType and the
// number of statistics indicated by the BlockManifest. This functional
// verifies that a non-empty InstanceID was specified and that the format of
// metrics populated is appropriate for the data type defined each supported
// metric. It also checks if the metrics are null, which could occur if a
// client included a metric in the BlockManifest used by the
// GetStatisticsCollection function that cannot be populated.
sub ValidateRecords(#BulkStatistics[],
    $BlockManifests[],
    $BSSDs[]) {

    for (#i in #BulkStatistics[]) {

        // The function split() below parses the content of an element in
        // #BulkStatistics[] into multiple sub-strings based on occurrences
        // of carriage return. (i.e. "\n")
        #Records[] = #BulkStatistics[#i].split("\n")
        for (#j in #Records[]) {

            // The function split() below further parses the content of an
            // element in #Records[] into multiple sub-strings based on
```

```

// occurrences of semi-colon. The resulting elements contain (in
// order) the InstanceID and ElementType properties followed by the
// metrics supported.
#RecordElements[] = #Records[#j].split(";")

// Each element MUST contain at least InstanceID and ElementType.
if (#RecordElements[].length < 2) {
<ERROR! Statistics Record does not contain InstanceID and/or
    ElementType>
}
// The InstanceID in the record MUST match the InstanceID of a BSSD.
$StatsData = null
for (#k in $BSSDs[]) {
if ($BSSDs[#k]->InstanceID == #RecordElements[0]) {
    $StatsData = $BSSDs[#k]
    break
}
}
if ($StatsData == null) {
<ERROR! Statistics instance could not be found for record>
}

// The function Integer() below is used to convert a string
// representation of an integer to an int value.
#elementType = Integer(#RecordElements[1])
if (#elementType != $StatsData.ElementType) {
<ERROR! ElementTypes for statistics record and instance do not
    match>
}

// Get the BlockManifest that describes this record. If none exists
// then the record does not contain a valid ElementType.
$BlockManifest = &GetBlockManifestByType($BlockManifests[],
    #elementType)
if ($BlockManifest == null) {
<ERROR! ElementType in Statistics Record not recognized>
}

// There MUST be two elements in the record (i.e. InstanceID and
// ElementType) in addition to one element for each supported
// metric.
if (#RecordElements.length !=
    &GetNumStatsIncluded($BlockManifest) + 2) {
<ERROR! Statistics record does not contain the expected number
    of metrics>
}

```

Block Server Performance Subprofile

```
// All default manifests MUST contain StatisticTime
if (!$BlockManifest.IncludeStatisticTime) {
<ERROR! Default manifest does not specify required property
    value IncludeStatisticTime=true>
}

// The function Datetime() below is used to convert a string
// representation of a DateTime value into a DateTime object
#statisticTime = Datetime(#RecordElements[2])

// Copy instance for local modification
$Manifest = $BlockManifest
// Validate the metrics in each record
#CurrentProperty = 0
#CurrentPropertyName = "Unknown"
#k = 3
while (#k < #RecordElements[.length]) {
// The remaining record elements should be integral values
// Parse the next element in the record and save the relevant
// property from the BSSD instance (and its name for inclusion
// in error codes)
#CurrentElement = Integer(#RecordElements[#k])
if ($Manifest.IncludeTotalIOs) {
    #CurrentProperty = $StatsData.TotalIOs
    #CurrentPropertyName = "TotalIOs"

    // Avoid double-checking for inclusion of this metric
    $Manifest.IncludeTotalIOs = false
} else if ($Manifest.IncludeKBytesTransferred) {
    #CurrentProperty = $StatsData.KBytesTransferred
    #CurrentPropertyName = "KBytesTransferred"

    // Avoid double-checking for inclusion of this metric
    $Manifest.IncludeKBytesTransferred = false
} else if ($Manifest.IncludeIOTimeCounter) {
    #CurrentProperty = $StatsData.IOTimeCounter
    #CurrentPropertyName = "IOTimeCounter"

    // Avoid double-checking for inclusion of this metric
    $Manifest.IncludeIOTimeCounter = false
} else if ($Manifest.IncludeReadIOs) {
    #CurrentProperty = $StatsData.ReadIOs
    #CurrentPropertyName = "ReadIOs"

    // Avoid double-checking for inclusion of this metric
    $Manifest.IncludeReadIOs = false
} else if ($Manifest.IncludeReadHitIOs) {
```


Block Server Performance Subprofile

```
#CurrentProperty = $StatsData.ReadHitIOs
#CurrentPropertyName = "ReadHitIOs"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeReadHitIOs = false
} else if ($Manifest.IncludeReadIOTimeCounter) {
    #CurrentProperty = $StatsData.ReadIOTimeCounter
    #CurrentPropertyName = "ReadIOTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeReadIOTimeCounter = false
} else if ($Manifest.IncludeReadHitIOTimeCounter) {
    #CurrentProperty = $StatsData.ReadHitIOTimeCounter
    #CurrentPropertyName = "ReadHitIOTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeReadHitIOTimeCounter = false
} else if ($Manifest.IncludeKBytesRead) {
    #CurrentProperty = $StatsData.KBytesRead
    #CurrentPropertyName = "KBytesRead"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeKBytesRead = false
} else if ($Manifest.IncludeWriteIOs) {
    #CurrentProperty = $StatsData.WriteIOs
    #CurrentPropertyName = "WriteIOs"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeWriteIOs = false
} else if ($Manifest.IncludeWriteHitIOs) {
    #CurrentProperty = $StatsData.WriteHitIOs
    #CurrentPropertyName = "WriteHitIOs"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeWriteHitIOs = false
} else if ($Manifest.IncludeWriteIOTimeCounter) {
    #CurrentProperty = $StatsData.WriteIOTimeCounter
    #CurrentPropertyName = "WriteIOTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeWriteIOTimeCounter = false
} else if ($Manifest.IncludeWriteHitIOTimeCounter) {
    #CurrentProperty = $StatsData.WriteHitIOTimeCounter
    #CurrentPropertyName = "WriteHitIOTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeWriteHitIOTimeCounter = false
```

Block Server Performance Subprofile

```
} else if ($Manifest.IncludeKBytesWritten) {
#CurrentProperty = $StatsData.KBytesWritten
#CurrentPropertyName = "KBytesWritten"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeKBytesWritten = false
} else if ($Manifest.IncludeIdleTimeCounter) {
#CurrentProperty = $StatsData.IdleTimeCounter
#CurrentPropertyName = "IdleTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeIdleTimeCounter = false
} else if ($Manifest.IncludeMaintOp) {
#CurrentProperty = $StatsData.MaintOp
#CurrentPropertyName = "MaintOp"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeMaintOp = false
} else if ($Manifest.IncludeMaintTimeCounter) {
#CurrentProperty = $StatsData.MaintTimeCounter
#CurrentPropertyName = "MaintTimeCounter"

// Avoid double-checking for inclusion of this metric
$Manifest.IncludeMaintTimeCounter = false
}

if (#statisticTime == $BlockStats.StatisticTime) {
    // record and instance property should be equal
    if (#CurrentElement != #CurrentProperty) {
        <ERROR! Record Element inconsistent with BSSD
        property #CurrentPropertyName>
    }
} else if (#statisticTime > $BlockStats.StatisticTime) {
    // record should be >= instance property
    if (#CurrentElement < #CurrentProperty) {
        <ERROR! Record Element inconsistent with BSSD property
        #CurrentPropertyName. The counter may have
        rolled back to 0>
    }
} else {
    // record should be <= instance property
    if (#CurrentElement > #CurrentProperty) {
        <ERROR! Record Element inconsistent with BSSD property
        #CurrentPropertyName. The counter may have
        rolled back to 0>
    }
}
}
```

```

        k++

        } // while (#k < #RecordElements[.length)...
    } // for (#j in #Records[])
} // for (#i in #BulkStatistics[])
}

// This function takes a container of BlockManifest instances and locates the
// instance that represents the specified element type. Null is returned if
// the specified instance cannot be located in the container.
sub CIMInstance GetBlockManifestByType($BlockManifests[], #elementType) {
    for (#i in $BlockManifests[]) {
        if ($BlockManifests[#i].ElementType == #elementType) {
            return $BlockManifests[#i]
        }
    }
    return null
}

// MAIN
//
// 1. Loop through the top-level ComputerSystems and retrieve the
// hosted BlockStatisticsService.
for (#i in $StorageSystems->[]) {

    // Step 1. Retrieve the hosted BlockStatisticsService.
    $StorageSystem-> = $StorageSystems->[#i]
    $StatServices->[] = AssociatorNames($StorageSystem->,
        "CIM_HostedService",
        "CIM_BlockStatisticsService",
        "Antecedent",
        "Dependent")
    // There should be one and only one BlockStatisticsService.
    $StatService-> = $StatServices->[0]

    // Step 2. Retrieve the capabilities describing the BlockStatisticService.
    $StatCapabilities[] = Associators($StatService->,
        "CIM_ElementCapabilities",
        "CIM_BlockStatisticsCapabilities",
        "ManagedElement",
        "Capabilities",
        false,
        false,
        {"ElementTypesSupported", "SynchronousMethodsSupported"})
    // There should be one and only one BlockStatisticsCapabilities.
    $Capabilities = $StatCapabilities[0]

```

Block Server Performance Subprofile

```
#SynchCollectionRetrieval = contains(5,      // "GetStatisticsCollection"
    $Capabilities.SynchronousMethodsSupported)

// Step 3. Locate the StatisticsCollection
$StatCollections->[] = AssociatorNames($StorageSystem->,
    "CIM_HostedCollection",
    "CIM_StatisticsCollection",
    "Antecedent",
    "Dependent")
// There should be one and only one StatisticsCollection.
$StatCollection-> = $StatCollections->[0]
// Step 4. Locate the default ManifestCollection
$ManifestCollections[] = Associators($StatCollection->,
    "CIM_AssociatedBlockStatisticsManifestCollection",
    "CIM_BlockStatisticsManifestCollection",
    "Statistics",
    "ManifestCollection",
    false,
    false,
    {"IsDefault"})
$DefaultManifestCollection = null
for (#j in $ManifestCollections[]) {
    if ($ManifestCollections[#j].IsDefault) {
        $DefaultManifestCollection = $ManifestCollections[#j]
        break
    }
}
if ($DefaultManifestCollection == null) {
    <ERROR! A default ManifestCollection MUST exist>
}
// Step 5. Locate the default BlockManifests which identify what statistical
// data is supported for each element type. (e.g. disk, volume, etc.)
#PropList = {"InstanceID", "ElementName", "ElementType",
    "IncludeStatisticTime", "IncludeTotalIOs",
    "IncludeKBytesTransferred", "IncludeIOTimeCounter",
    "IncludeReadIOs", "IncludeReadHitIOs", "IncludeReadIOTimeCounter",
    "IncludeReadHitIOTimeCounter", "IncludeKBytesRead",
    "IncludeWriteIOs", "IncludeWriteHitIOs",
    "IncludeWriteIOTimeCounter", "IncludeWriteHitIOTimeCounter",
    "IncludeKBytesWritten", "IncludeIdleTimeCounter", "IncludeMaintOp",
    "IncludeMaintTimeCounter"}
$DefaultBlockManifests[] = Associators(
    $DefaultManifestCollection.GetObjectPath(),
    "CIM_MemberOfCollection",
    "CIM_BlockStatisticsManifest",
    "Collection",
    "Member",
```

```

        false,
        false,
        #PropList)
// There MUST be one default Block Manifest for each element type supported.
if ($Capabilities.ElementTypesSupported[].length
    != $DefaultBlockManifest[].length) {
    <ERROR! Required default BlockManifests do not exist>
}

// Step 6. Traverse from the StatisticsCollection to the
// BlockStorageStatisticalData. If SyncCollectionRetrieval is supported,
// then this is necessary for validation of the Manifest data retrieved
// through the GetStatisticsCollection method. If it is not supported,
// then these instances must be used to retrieve the statistics directly.
$BlockStats[] = Associators($StatCollection->,
    "CIM_MemberOfCollection",
    "CIM_BlockStorageStatisticalData",
    "Collection",
    "Member",
    false,
    false,
    #PropList)

if (#SynchCollectionRetrieval) {

    // Step 7a. Retrieve the data specified by the default
    // ManifestCollection in bulk.
    %InArguments["ElementTypes"] = $Capabilities.ElementTypesSupported[]
    %InArguments["ManifestCollection"] =
        $DefaultManifestCollection.getObjectPath()
    %InArguments["StatisticsFormat"] = 2// "CSV"
    #MethodReturn = InvokeMethod($StatService->,
        "GetStatisticsCollection",
        %InArguments,
        %OutArguments)
    if (#MethodReturn == 0) {
        #Statistics[] = %OutArguments["Statistics"]
        // Step 8. Parse the bulk statistical data retrieved to validate
        // the values (at least as much as is feasible)
        &ValidateRecords(#Statistics[], $DefaultBlockManifests[#j],
            $BlockStats[])
    } else {
        <ERROR! Bulk statistical data retrieval failed>
    }
} else {

    // Step 7b. Since bulk statistics retrieval is not supported, the

```

```

// statistical data must be retrieved directly.
for (#j in $BlockStats[]) {
    $BlockStat = $BlockStats[#j]
    $BlockManifest = GetBlockManifestByType($DefaultBlockManifests[],
        $BlockStat.ElementType)
    if ($BlockManifest == null) {
        <ERROR! The required default BlockManifest does not exist for
        this element type>
    }
    // Determine the supported statistical properties specified by
    // $BlockManifest, and retrieve the corresponding property values
    // for this element type from $BlockStat.
}
}
}

```

EXPERIMENTAL

7.7.2 Building an Object Map of Metered Elements

```

// DESCRIPTION
//
// This recipe describes how to build a record of all metered object instances
// and a topology of how the instances are related. (e.g. volume mapping to
// disk drives, ports used to access volumes, etc.)
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
// 1. The name of a top-level ComputerSystem instance for an Array, Storage
// Virtualizer, or Volume Manager implementation supporting the Block Server
// Performance Subprofile has previously been discovered via SLP and is known
// as $StorageSystem->.
// 2. The element types that support performance statistics are known as
// #ElementTypes[] whose content is populated from the property value of
// CIM_BlockStatisticsCapabilities.ElementTypesSupported.
// 3. The performance statistics properties supported for each element type are
// know as #<ElementType>DataPropList[]. (e.g. #VolumeDataPropList[],
// #DiskDataPropList[], etc.) The content of the property lists is determine
// from the default instance of CIM_BlockStatisticsManifest for each element type.
// 4. The required properties for each element type are know as
// #<ElementType>PropList[]. (e.g. #VolumePropList[], #DiskDataPropList[], etc.)

// Function GetAssociatedStats
//
// This function retrieves the instance data of BlockStorageStatisticalData
// associated to the specified metered object. If there is no instance data
// associated, null is returned.
//

```

```

sub CIMInstance[] GetAssociatedStats(CIMObjectPath $MeteredObject->,
    string[] #PropList) {
    $StatData[] = Associators($MeteredObject->,
        "CIM_ElementStatisticalData",
        "CIM_BlockStorageStatisticalData",
        "ManagedElement",
        "Stats",
        false,
        false,
        #PropList)
    return $StatData[]
}

// This function retrieves the performance statistics of a CompositeExtent
// then recursively traverses the hierarchy beneath it.
sub void traverseComposition(REF $Composite->) {

    // Retrieve the performance statistics of the Composite Extent.
    $CompositeExtentStatData[] = &GetAssociatedStats($Composite->,
        #ExtentDataPropList[])
    // There may not be BlockStorageStatisticalData for each and every level
    // of Composite Extents.
    if ($CompositeExtentStatData[] != null) {
        $CompositeExtentStats = $CompositeExtentStatData[0]
    }

    // Retrieve the associations in which this Composite Extent is the
    // Dependent reference. The association instances retrieved should be
    // either BasedOn or CompositeExtentBasedOn.
    $Associations[] = References($Composite->,
        "CIM_BasedOn",
        "Dependent",
        false,
        false,
        NULL)

    // There must be one or more associations involving the Composite Extent
    // as the Antecedent reference.
    if ($Associations[] == null || $Associations[].length == 0) {
        <EXIT! Required associations not found>
    }

    // Determine which association class was discovered.
    #AssocClass = "CIM_BasedOn"
    if ($Associations[0] ISA CIM_CompositeExtentBasedOn) {
        #AssocClass = "CIM_CompositeExtentBasedOn"
    }
}

```

```

// Retrieve the underlying Extents.
$TargetExtents->[] = AssociatorNames($Composite->,
    #AssocClass,
    NULL,
    "Dependent",
    "Antecedent")

// Examine the QOS of the current level's Composite Extent
$CompositeExtent = GetInstance($Composite->,
    false,
    false,
    false,
    {"IsConcatenated", "ExtentStripeLength"})

// For each underlying extent at this level, traverse the sub-tree it is
// the sub-root of. If the extent is a CompositeExtent, then this is part
// of a complex RAID level; recursively invoke the Composition Algorithm.
// Otherwise it is just a regular StorageExtent and thus must be decomposed
// from it's Antecedent, so invoke the recursive Decomposition Algorithm.
for (#i in $TargetExtents->[]) {
    if ($TargetExtents->[#i] ISA CIM_CompositeExtent) {
        &traverseComposition($TargetExtents->[#i++])
    } else {
        &traverseDecomposition($TargetExtents->[#i++])
    }
}

// This function recursively traverses the hierarchy below a non-Composite
// StorageExtent.
sub void traverseDecomposition(REF $StartingExtent->) {

    // The Starting Extent is allocated partially or in full from the
    // Antecedent Extent, so a single BasedOn is expected.
    $TargetExtents[] = Associators($StartingExtent->,
        "CIM_BasedOn",
        "CIM_StorageExtent",
        "Dependent",
        "Antecedent",
        false,
        false,
        {"Primordial"})

    // Since the Starting Extent is allocated from the Antecedent, there must
    // be only one Antecedent.
    if ($TargetExtents[] == null || $TargetExtents[].length != 1) {

```



```

    <ERROR! Extent allocated from multiple Antecedents>
}
$TargetExtent = $TargetExtents[0]

if ($TargetExtent ISA CIM_CompositeExtent) {
    // This is a Composite Extent representing a RAID Level. Since we
    // encountered the Composite in a decomposition, it is the "top"
    // extent in a pool and the Dependent/Antecedent relationship falls
    // into one of the following scenarios:
    //
    // o The Starting Extent is a StorageVolume that is one-to-one with
    //   the Target Composite Extent.
    //
    // o The Starting Extent is a StorageVolume partially allocated from
    //   the Target Composite Extent, where the Composite is one-to-one
    //   with the Storage Pool which is a RAID Group.
    //
    // o The Starting Extent is a ComponentExtent of a Child Concrete
    //   pool and is partially allocated from the Target Composite Extent
    //   where the Composite is one-to-one with the parent RAID Group pool.
    //
    // Call the (recursive) function to analyze the sub-hierarchy
    // composed by the Target Extent.
    //
    &traverseComposition($TargetExtent.GetObjectPath())
} else {
    // Check here to see if we have reached the leaves of the hierarchy
    if ($TargetExtent.Primordial == true) {
        // Recursion ends with each Primordial Extent.
        return
    } else {
        // Since the Dependent was a regular StorageExtent, and not
        // Primordial, it must be decomposed from an Antecedent, so invoke
        // ourselves recursively.
        &traverseDecomposition($TargetExtent.GetObjectPath())
    }
}
}

// This function locates the logical devices on the specified ComputerSystem
// and retrieves the supported statistical information. Note that the
// ComputerSystem specified may be a top-level, peer, front-end or back-end
// system.
sub void discoverSupportedDeviceStats(REF $System->) {

    // Retrieve all ports on the system.
    $Ports[] = Associators($System.GetObjectPath(),

```

Block Server Performance Subprofile

```

        "CIM_SystemDevice",
        "CIM_LogicalPort",
        "GroupComponent",
        "PartComponent",
        false,
        false,
        #PortPropList[])
if ($Ports[] != null && $Ports[].length > 0) {

    // Determine if performance statistics are supported for any type of
    // port.
    #SupportsPortStats = contains(6, #ElementTypes[]) // "Front-end Port"
    || contains(7, #ElementTypes[]) // "Back-end Port"
    for (#j in $Ports[]) {
        if (#SupportsPortStats) {

            // Retrieve the performance statistics of the system's port.
            $PortStatData[] = &GetAssociatedStats(
                $Ports[#j].getObjectPath(),
                #PortDataPropList[])
            // NOTE: Performance statistics may not be supported for
            // this particular type of port. (i.e. Front-end vs. Back-end)
            if ($PortStatData[] != null && $PortStatData[].length > 0) {
                // There should be one and only one
                // BlockStorageStatisticalData.
                $PortStats[#j] = $PortStatData[0]

                // Determine the type of this port.
                #PortType[#j] = $PortStats.ElementType
            }
        }
    }
}

// Retrieve all volumes on the system.
$Volumes[] = Associators($System.getObjectPath(),
    "CIM_SystemDevice",
    "CIM_StorageVolume",
    "GroupComponent",
    "PartComponent",
    false,
    false,
    #VolumePropList[])
if ($Volumes[] != null && $Volumes[].length > 0) {

    // Determine if performance statistics are supported for volume.
    #SupportsVolumeStats = contains(8, #ElementTypes[]) // "Volume"

```

```

for (#k in $Volumes[]) {
    if (#SupportsVolumeStats) {

        // Retrieve the performance statistics of the volumes
        $VolumeStatData[] = &GetAssociatedStats(
            $Volumes[#k].getObjectPath(),
            #VolumeDataPropList[])
        // There should be one and only one BlockStorageStatisticalData.
        $VolumeStats = $VolumeStatData[0]
    }

    // Retrieve the protocol controllers through which the volume is
    // visible.
    $ProtocolControllers[] = Associators($Volumes[#k].getObjectPath(),
        "CIM_ProtocolControllerForUnit",
        "CIM_SCSIProtocolController",
        "Dependent",
        "Antecedent",
        false,
        false,
        #ProtocolControllerPropList[])
    if ($ProtocolControllers[] != null
        && $ProtocolControllers[].length > 0) {
        for (#l in ($ProtocolControllers[])) {

            // Retrieve the protocol controller's endpoint.
            $ProtocolEndpoints[] = Associators(
                $ProtocolControllers[#l].getObjectPath(),
                "CIM_SAPAvailableForElement",
                "CIM_SCSIProtocolEndpoint",
                "ManagedElement",
                "AvailableSAP",
                false,
                false,
                #ProtocolControllerPropList[])
            if ($ProtocolEndpoints[] != null ) {
                for (#pe in (#ProtocolEndpoints[])) {
                    // There should be one and only one ProtocolEndpoint
                    $ProtocolEndpoint = $ProtocolEndpoints[#pe]

                    // Retrieve the ports that access this ProtocolEndpoint.
                    $AccessingPorts[] = Associators(
                        $ProtocolEndpoint.getObjectPath(),
                        "CIM_DeviceSAPImplementation",
                        "CIM_LogicalPort",
                        "Dependent",
                        "Antecedent",

```

Block Server Performance Subprofile

```

        false,
        false,
        #PortPropList[])
    }
}

// Determine if performance statistics are supported for Extents.
#SupportsExtentStats = contains(9, #ElementTypes[])// "Extent"

// NOTE: StorageExtents are investigated ONLY if performance
// statistics are supported for "Extent" and/or "Disk Drive".
// Performance statistics support for "composite" StorageExtents
// is indicated by the "Extent" capability. Performance statistics
// support for "primordial" StorageExtents is indicated by the
// "Disk Drive" capability.
//
// StorageExtents may not be present if the Extent Composition
// Subprofile is not supported.
if (#SupportsExtentStats) {

// Retrieve the StorageExtents that comprise the StorageVolume.
$ComponentExtents[] = Associators(
    $Volumes[#k].getObjectPath(),
    "CIM_BasedOn",
    "CIM_StorageExtent",
    "Dependent",
    "Antecedent",
    false,
    false,
    #ExtentPropList)

// Retrieve the performance statistics of the composite
// Storage Extent(s).
if ($ComponentExtents[] != null
    && $ComponentExtents[].length > 0) {
    &traverseComposition($ComponentExtents[0].getObjectPath())
}
}

// Determine if performance statistics are supported for Disk Drive.
#SupportsDiskStats = contains(10, #ElementTypes[])// "Disk Drive"
if (#SupportsDiskStats) {

// Retrieve the primordial StorageExtents to which the disk
// performance statistics will be associated.
$DiskExtents[] = &findPrimordials(

```

```

        $Volumes[#k].getObjectPath())
    if ($DiskExtents[] == null || $DiskExtents[].length == 0) {
        <ERROR! Required primordial StorageExtents not found>
    }
    for (#m in $DiskExtents[]) {
        $DiskExtentStatData[] = &GetAssociatedStats(
            $DisExtents[#m].getObjectPath(),
            #DiskDataPropList[])
        // There should be one and only one
        // BlockStorageStatisticalData.
        $DiskExtentStats = $DiskExtentStatData[0]
    }
}
}
}

// MAIN
//
// Step 1. Retrieve the performance statistics for the top-level system.
if (contains(2, // "Computer System"
    #ElementTypes[]) {
    $TopSystemStatData[] = &GetAssociatedStats($StorageSystem->,
        #TopSystemDataPropList[])
    // There should be one and only one BlockStorageStatisticalData.
    $TopSystemStats = $TopSystemStatData[0]
}

// Step 2. Discover the logical devices on the top-level system and their
// related performance statistics
&discoverSupportedDeviceStats($StorageSystem->)

// Step 3. Retrieve the component systems in a multiple system device.
// NOTE: Traversing ComponentCS from the top-level system to its component
// systems will retrieve ALL component systems. In the case of a device that
// supports 2-tier redundancy, the relationship between the component systems
// (i.e. first redundancy tier) to the sub-component systems would be determined
// by investigating the ConcreteIdentity and MemberOfCollection relationships
// to a RedundancySet. See the Multiple Computer System Subprofile for more
// detail.
$ComponentSystems[] = Associators($StorageSystem->,
    "CIM_ComponentCS",
    "CIM_ComputerSystem",
    "GroupComponent",
    "PartComponent",
    false,
    false,

```

Block Server Performance Subprofile

```
#ComponentSystemPropList[])
if ($ComponentSystems[] != null && $ComponentSystems[].length > 0) {

    // Step 4. Determine if performance statistics are supported for any type
    // of component system.
    #SupportsComponentSystemStats =
        contains(3, #ElementTypes[])// "Front-end Computer System"
        || contains(4, #ElementTypes[])// "Peer Computer System"
        || contains(5, #ElementTypes[])// "Back-end Computer System"
    for (#i in $ComponentSystems[]) {
        $ComponentSystemPath = $ComponentSystems[#i].getObjectPath()
        if (#SupportsComponentSystemStats) {

            // Step 5. Retrieve the performance statistics of the component
            // system.
            $ComponentSystemStatData[] = &GetAssociatedStats(
                $ComponentSystemPath,
                #ComponentSystemDataPropList[])
            // NOTE: Performance statistics may not be supported for this
            // particular type of component system. (i.e. Front-end vs.
            // Back-end vs. Peer Computer Systems)
            if ($ComponentSystemStatData[] != null
                && $ComponentSystemStatData[].length > 0) {
                // There should be one and only one BlockStorageStatisticalData.
                $ComponentSystemStats[#i] = $ComponentSystemStatData[0]

                // Step 6. Determine the type of this component system.
                #ComponentSystemType[#i] = $ComponentSystemStats.ElementType
            }

            // Step 7. Discover the Topology of the component computer systems by
            // finding the RedundancySet that each of the ComponentSystems belong
            // to (if any), and the ComputerSystem that has a concrete identity
            // relationship with that RedundancySet. The computer system that is
            // one tier above the current component system is stored in an array
            // of ParentComputerSystems, with each entry corresponding to the
            // component system at the same index in the ComponentSystems array.

            $RedundancySets->[] = AssociatorNames($ComponentSystemPath->,
                "CIM_MemberOfCollection",
                "CIM_RedundancySet",
                "Member",
                "Collection")
            if(RedundancySets->[] != null && $RedundancySets->[].length > 0)
            {
                if($RedundancySets->[].length > 1)
                {
```

```

        <ERROR! Component System belongs to more than one Redundancy
        Set>
    }
    $AggregateSystems->[] = AssociatorNames($RedundancySets->[0],
        "CIM_ConcreteIdentity",
        "CIM_ComputerSystem",
        "SameElement",
        "SystemElement")
    if($AggregateSystems->[] == null ||
        $AggregateSystems->[].length != 1)
    {
        <ERROR! Could not find Concrete Computer System for Redundancy
        Set>
    }
    $ParentComputerSystems->[#i] = $AggregateSystems->[0]
}
}
// Step 8. Discover the logical devices on the component system and
// their related performance statistics
&discoverSupportedDeviceStats($ComponentSystemPath->)
}
}

```

EXPERIMENTAL

7.7.3 Retrieving Statistics for a Specific Volume

```

// DESCRIPTION
//
// This recipe describes how to retrieve the supported performance statistics
// for a specific set of StorageVolumes.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
// 1. The name of a top-level ComputerSystem instance for an Array, Storage
// Virtualizer, or Volume Manager implementation supporting the Block Server
// Performance Subprofile has previously been discovered via SLP and is known
// as $StorageSystem->.
// 2. A specific set of StorageVolumes is known as $StorageVolume->[].
//
// MAIN
//
// Step 1. Retrieve the hosted BlockStatisticsService.
$StatServices->[] = AssociatorNames($StorageSystem->,
    "CIM_HostedService",
    "CIM_BlockStatisticsService",

```

Block Server Performance Subprofile

```
        "Antecedent",
        "Dependent")
// There should be one and only one BlockStatisticsService.
$StatService-> = $StatServices->[0]

// Step 2. Retrieve the capabilities describing the BlockStatisticService.
$StatCapabilities[] = Associators($StatService->,
    "CIM_ElementCapabilities",
    "CIM_BlockStatisticsCapabilities",
    "ManagedElement",
    "Capabilities",
    false,
    false,
    {"ElementTypesSupported"})
// There should be one and only one BlockStatisticsCapabilities.
$Capabilities = $StatCapabilities[0]
if !contains(8, // "Volume"
    $Capabilities.ElementTypesSupported) {

    <EXIT! StorageVolume performance statistics not supported>
}

// Step 3. Locate the default ManifestCollection
$ManifestCollections[] = Associators($StatCollection->,
    "CIM_AssociatedBlockStatisticsManifestCollection",
    "CIM_BlockStatisticsManifestCollection",
    "Statistics",
    "ManifestCollection",
    false,
    false,
    {"IsDefault"})
$DefaultManifestCollection = null
for #i in $ManifestCollections[] {
    if ($ManifestCollections[#i].IsDefault) {
        $DefaultManifestCollection = $ManifestCollections[#i]
        break
    }
}
if ($DefaultManifestCollection == null) {
    <ERROR! A default ManifestCollection MUST exist>
}

// Step 4. Locate the default BlockManifest which identifies the statistical
// data supported for StorageVolumes.
$VolumeManifest = null
string[] #PropList = {"ElementType", "IncludeStatisticTime", "IncludeTotalIOs",
    "IncludeKBytesTransferred", "IncludeIOTimeCounter", "IncludeReadIOs",
```



```

    "IncludeReadHitIOs", "IncludeReadIOTimeCounter",
    "IncludeReadHitIOTimeCounter", "IncludeKBytesRead", "IncludeWriteIOs",
    "IncludeWriteHitIOs", "IncludeWriteIOTimeCounter",
    "IncludeWriteHitIOTimeCounter", "IncludeKBytesWritten",
    "IncludeIdleTimeCounter", "IncludeMaintOp", "IncludeMaintTimeCounter"}
$DefaultBlockManifests[] = Associators(
    $DefaultManifestCollection.GetObjectPath(),
    "CIM_MemberOfCollection",
    "CIM_BlockStatisticsManifest",
    "Collection",
    "Member",
    false,
    false,
    #PropList)
for #i in $DefaultBlockManifests[] {
    if ($DefaultBlockManifests[#i].ElementType == 8) {
        $VolumeManifest = $DefaultBlockManifests[#i]
        break
    }
}
if ($VolumeManifest == null) {
    <ERROR! Required default BlockManifest for StorageVolume not found>
}

// Step 5. Retrieve the performance statistics for each specified StorageVolume.
for (#i in $StorageVolume->[]) {
    $VolumeStatData[] = Associators($StorageVolume->[#i],
        "CIM_ElementStatisticalData",
        "CIM_BlockStorageStatisticalData",
        "ManagedElement",
        "Stats",
        false,
        false,
        null)
    // There should be one and only one BlockStorageStatisticalData.
    if ($VolumeStatData[] == null || $VolumeStatData[].length != 1) {
        <ERROR! The required staticistics were not found>
    }
    $VolumeStats = $VolumeStatData[0]

    // Step 6. Extract the performance statistics supported by the
    // StorageVolume.
    if ($VolumeManifest.IncludeStatisticTime) {
        #StatisticTime = VolumeStats.StatisticTime
    } else {
        <ERROR! StatisticTime is a required property for Volumes and should
        be set to "true" in the default BlockManifest>
    }
}

```

Block Server Performance Subprofile

```
}
if ($VolumeManifest.IncludeTotalIOs) {
    #TotalIOs = VolumeStats.TotalIOs
} else {
    <ERROR! TotalIOs is a required property for Volumes and should
        be set to "true" in the default BlockManifest>
}
if ($VolumeManifest.IncludeKBytesTransferred) {
    #KBytesTransferred = VolumeStats.KBytesTransferred
} else {
    <ERROR! KBytesTransferred is a required property for Volumes and
        should be set to "true" in the default BlockManifest>
}
if ($VolumeManifest.IncludeIOTimeCounter) {
    #IOTimeCounter = VolumeStats.IOTimeCounter
}
if ($VolumeManifest.IncludeReadIOs) {
    #ReadIOs = VolumeStats.ReadIOs
} else {
    <ERROR! ReadIOs is a required property for Volumes and should
        be set to "true" in the default BlockManifest>
}
if ($VolumeManifest.IncludeReadHitIOs) {
    #ReadHitIOs = VolumeStats.ReadHitIOs
} else {
    <ERROR! ReadHitIOs is a required property for Volumes and should
        be set to "true" in the default BlockManifest>
}
if ($VolumeManifest.IncludeReadIOTimeCounter) {
    #ReadIOTimeCounter = VolumeStats.ReadIOTimeCounter
}
if ($VolumeManifest.IncludeReadHitIOTimeCounter) {
    #ReadHitIOTimeCounter = VolumeStats.ReadHitIOTimeCounter
}
if ($VolumeManifest.IncludeKBytesRead) {
    #KBytesRead = VolumeStats.KBytesRead
}
if ($VolumeManifest.IncludeWriteIOs) {
    #WriteIOs = VolumeStats.WriteIOs
} else {
    <ERROR! WriteIOs is a required property for Volumes and should
        be set to "true" in the default BlockManifest>
}
if ($VolumeManifest.IncludeWriteHitIOs) {
    #WriteHitIOs = VolumeStats.WriteHitIOs
} else {
    <ERROR! WriteHitIOs is a required property for Volumes and should
```

```

        be set to "true" in the default BlockManifest>
    }
    if ($VolumeManifest.IncludeWriteIOTimeCounter) {
        #WriteIOTimeCounter = VolumeStats.WriteIOTimeCounter
    }
    if ($VolumeManifest.IncludeWriteHitIOTimeCounter) {
        #WriteHitIOTimeCounter = VolumeStats.WriteHitIOTimeCounter
    }
    if ($VolumeManifest.IncludeKBytesWritten) {
        #KBytesWritten = VolumeStats.KBytesWritten
    }
    if ($VolumeManifest.IncludeIdleTimeCounter) {
        #IdleTimeCounter = VolumeStats.IdleTimeCounter
    }
}

```

7.7.4 Summary of Statistics Support by Element

Not all statistics properties are kept for all elements. Table 70 illustrates the statistics properties that are kept for each of the metered elements.

Table 70: Summary of Statistics Support by Element

Statistic Property	Top Level Computer System	Component Computer System (Front-end)	Component Computer System (Peer)	Component Computer System (Back-end)	Front-end Port	Back-end Port	Volume (Logical Disk)	Composite Extent	Disk
StatisticTime	R	R	R	R	R	R	R	R	R
TotalIOs	R	R	R	R	R	R	R	R	R
KBytes Transferred	R	O	O	O	R	O	R	R	R
IOTimeCounter	O	O	O	O	O	O	O	N	O
ReadIOs	O	R	R	N	N	N	R	N	R
ReadHitIOs	O	R	R	N	N	N	R	N	N
ReadIOTimeCounter	O	O	O	N	N	N	O	N	O
ReadHitIOTimeCounter	O	O	O	N	N	N	O	N	N
KBytesRead	O	O	O	O	N	N	O	N	O
WriteIOs	O	R	R	N	N	N	R	N	O
WriteHitIOs	O	R	R	N	N	N	R	N	N
WriteIOTimeCounter	O	O	O	N	N	N	O	N	O

Statistic Property	Top Level Computer System	Component Computer System (Front-end)	Component Computer System (Peer)	Component Computer System (Back-end)	Front-end Port	Back-end Port	Volume (Logical Disk)	Component Extent	Disk
WriteHitIO TimeCounter	O	O	O	N	N	N	O	N	N
KBytesWritten	O	O	O	O	N	N	O	N	O
IdleTimeCounter	N	N	N	O	O	N	O	O	O
MaintOp	N	N	N	N	N	N	N	O	O
MaintTimeCounter	N	N	N	N	N	N	N	O	O

The legend is:

R – Required

O – Optional

N – Not specified

Notice that there is a difference between the “front-end” port and “back-end” port elements. And there is a difference between the Top Level Computer System (i.e., the Array, Storage Virtualizer or Volume Management Profile) and the component computer systems. Furthermore, there can be variations in the component computer systems. This is based on how component computer systems are configured. In some cases, these computer systems are “front-end” and “back-end” controllers. In other subsystems, they are “peer” controllers.

Note: Controller LUNs (SCSIArbitraryLogicalUnits) and RemoteReplicaGroup are not shown in Table 705: Summary of Block Statistics Support by Element. They only require StatisticTime, TotalIOs and KBytesTransferred. All other properties are not SPECIFIED.

A complete list of definitions of the metered elements as defined by the ElementType property of BlockStorageStatisticalData are below:

- ElementType = 2 (Computer System) - These are statistics for the whole Array (virtualizer or volume manager).
- ElementType = 3 (Front-end Computer System) - This is the Computer System (controller) that provides the support for receiving the IO from host systems. The Front-end function acts as a target of IO.
- ElementType = 4 (Peer Computer System) - This is a Computer System that acts as both a front-end and back-end Computer System.
- ElementType = 5 (Back-end Computer System) - This is the Computer System (controller) that provides the support for driving the IO to the back-end storage (disk drives or external volumes). The back-end function acts as an initiator of IO.
- ElementType = 6 (Front-end Port) - A port in a disk array that connects the disk array (or Storage Virtualizer) to hosts using the storage. The Front End port is usually connected to either the Peer Computer System (controller) or to the Front-end Computer System (controller) in some Disk Arrays or Storage Virtualizers.
- ElementType = 7 (Back-end Port) - A port that can be inside the disk array housing that connects to the disk drives. This is connected to either the Peer Computer system (controller) or to the Back-end Computer System (controller) in some Disk Arrays or Storage Virtualizers.

- **ElementType = 8 (Volume)** - This is a Logical Unit that is the target of data IOs for storing or retrieving data. This would be a StorageVolume for Arrays or Storage Virtualizers. It would be a LogicalDisk for Volume Management Profiles.
- **ElementType = 9 (Extent)** - This is an intermediate Storage Extent. That is, it is not a Volume and it is not a Disk Drive. An example of the use of an Extent would be a RAID rank that creates a logical storage extent from multiple disk drives. In the case of Storage Virtualizers, this is used to represent the volumes that are imported from Arrays.
- **ElementType = 10 (Disk Drive)** - This is a disk drive.
- **ElementType = 11 (Arbitrary LUs)** - This is a Logical Unit that is the target of “control” IO functions. The Logical Unit does not contain data, but supports invocation of control functions in an Array or Storage Virtualizer.
- **ElementType = 12 (Remote Replica Group)** - Replication requires a local disk array and a remote disk array (in some “safe” location). The remote replica group is a group of disk drives in the remote disk array used to replicated defined data from the local disk array.

7.7.5 Formulas and Calculations

Table 70 identifies the set of statistics that are recommended for the various storage components in the array. These metrics, once collected, can be further enhanced through the definition of formulas and calculations that create additional ‘derived’ statistics.

Table 71 defines a set of such derived statistics. They are by no means the only possible derivations but serve as examples of the most commonly asked for statistics.

Table 71: Formulas and Calculations

Calculated Statistics	
New statistic	Formula
TimeInterval	delta StatisticTime
% utilization	$100 * (\text{delta StatisticTime} - \text{delta IdleTime}) / \text{delta StatisticTime}$
I/O rate	$\text{delta TotalIOs} / \text{delta StatisticTime}$
I/O response time	$\text{delta IOTime} / \text{delta TotalIOs}$
Queue depth	$\text{delta I/O rate} * \text{delta I/O response time}$
Service Time	$\text{utilization} / \text{I/O rate}$
Wait Time	$\text{Response Time} - \text{Service Time}$
Average Read Size	$\text{delta KBytesRead} / \text{delta ReadIOs}$
Average Write Size	$\text{delta KBytesWritten} / \text{delta WriteIOs}$
% Read	$100 * (\text{delta ReadIOs} / \text{delta TotalIOs})$
% Write	$100 * (\text{delta WriteIOs} / \text{delta TotalIOs})$
% Hit	$100 * ((\text{delta ReadHitIOs} + \text{delta WriteHitIOs}) / \text{delta TotalIOs})$

7.7.6 Block Server Performance Supported Capabilities Patterns

The Capabilities patterns summarized in Table 72 are formally recognized by the Block Server Performance Subprofile of the current version of SMI-S

Table 72: Block Server Performance Subprofile Supported Capabilities Patterns

ElementSupported	SynchronousMethods Supported	AsynchronousMethods Supported
Any (at least one)	NULL	NULL
Any (at least one)	Neither GettatisticsCollection nor Exec Query	NULL
Any (at least one)	GetStatisticsCollection	NULL
Any (at least one)	Any	NULL
Any (at least one)	Exec Query	NULL
Any (at least one)	GetStatisticsCollection, Query	NULL
Any (at least one)	Exec Query	NULL
Any (at least one)	"Manifest Creation", "Manifest Modification", and "Manifest Removal"	NULL
Any (at least one)	"Indications", "Query Collection"	NULL

An implementation will support GetStatisticsCollection, Query, GetStatisticsCollection and Query or neither. But if the implementation supports GetStatisticsCollection, it will shall support Synchronous execution.

If manifest collections are supported, then ALL three methods shall be supported (Creation, modification and removal).

7.8 Registered Name and Version

Block Server Performance version 1.2.0

7.9 CIM Elements

Table 73: CIM Elements for Block Server Performance

Element Name	Requirement	Description
CIM_AssociatedBlockStatisticsManifestCollection (Provider defined collection) (7.9.1)	Mandatory	This is an association between the StatisticsCollection and a provider supplied (pre-defined) manifest collection that defines the statistics properties supported by the profile implementation.
CIM_AssociatedBlockStatisticsManifestCollection (Client defined collection) (7.9.2)	Conditional	Conditional requirement: Clients can create manifests as identified by CIM_BlockStatisticsCapabilities.SynchronousMethodsSupported. This is an association between the StatisticsCollection and a client defined manifest collection.
CIM_BlockStatisticsManifest (Provider Support) (7.9.3)	Mandatory	An instance of this class defines the statistics properties supported by the profile implementation for one element type.
CIM_BlockStatisticsManifest (Client Defined) (7.9.4)	Conditional	Conditional requirement: Clients can modify manifests as identified by CIM_BlockStatisticsCapabilities.SynchronousMethodsSupported. An instance of this class defines the statistics properties of interest to the client for one element type.
CIM_BlockStatisticsManifestCollection (Provider Defined) (7.9.5)	Mandatory	An instance of this class defines the predefined collection of default block statistics manifests (one manifest for each element type).
CIM_BlockStatisticsManifestCollection (Client Defined) (7.9.6)	Conditional	Conditional requirement: Clients can create manifests as identified by CIM_BlockStatisticsCapabilities.SynchronousMethodsSupported. An instance of this class defines one client defined collection of block statistics manifests (one manifest for each element type).
CIM_BlockStorageStatisticalData (7.9.7)	Mandatory	This is a Subclass of CIM_StatisticalData for Block servers. It would be instantiated as specific block statistics for particular components.
CIM_BlockStatisticsCapabilities (7.9.8)	Mandatory	This defines the statistics capabilities supported by the implementation of the profile.
CIM_BlockStatisticsService (7.9.9)	Mandatory	This is a Service that provides (optional) services of bulk statistics retrieval and manifest set manipulation methods.
CIM_StatisticsCollection (7.9.10)	Mandatory	This would be a collection point for all Statistics that are kept for a Block Server.

Table 73: CIM Elements for Block Server Performance

Element Name	Requirement	Description
CIM_ElementCapabilities (7.9.11)	Mandatory	This associates the BlockStatisticsCapabilities to the BlockStatisticsService.
CIM_ElementStatisticalData (Top Level System Stats) (7.9.12)	Conditional	<p>Conditional requirement: Top level system statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "2".</p> <p>This associates a BlockStorageStatisticalData instance to the Top Level ComputerSystem for which the statistics are collected.</p>
CIM_ElementStatisticalData (Component System Stats) (7.9.13)	Conditional	<p>Conditional requirement: Component Systems statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "3", "4" or "5".</p> <p>This associates a BlockStorageStatisticalData instance to the component ComputerSystem for which the statistics are collected.</p>
CIM_ElementStatisticalData (Volume Stats) (7.9.14)	Conditional	<p>Conditional requirement: Volume statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "8", and the parent profile supports Storage Volumes.</p> <p>This associates a BlockStorageStatisticalData instance to the volume for which the statistics are collected.</p>
CIM_ElementStatisticalData (Logical Disk Stats) (7.9.15)	Conditional	<p>Conditional requirement: Volume statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "8", and the parent profile supports Logical Disks.</p> <p>This associates a BlockStorageStatisticalData instance to the volume for which the statistics are collected.</p>
CIM_ElementStatisticalData (Front end Port Stats) (7.9.16)	Conditional	<p>Conditional requirement: Front-end port statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "6".</p> <p>This associates a BlockStorageStatisticalData instance to the target port for which the statistics are collected.</p>

Table 73: CIM Elements for Block Server Performance

Element Name	Requirement	Description
CIM_ElementStatisticalData (Back end Port Stats) (7.9.17)	Conditional	<p>Conditional requirement: Back end port statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "7".</p> <p>This associates a BlockStorageStatisticalData instance to the back end port for which the statistics are collected.</p>
CIM_ElementStatisticalData (Remote Copy Stats) (7.9.18)	Conditional	<p>Conditional requirement: Remote Copy statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "12".</p> <p>This associates a BlockStorageStatisticalData instance to the remote copy service network for which the statistics are collected.</p>
CIM_ElementStatisticalData (Extent Stats) (7.9.19)	Conditional	<p>Conditional requirement: Extent statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "9".</p> <p>This associates a BlockStorageStatisticalData instance to the StorageExtent (composite extent) for which the statistics are collected.</p>
CIM_ElementStatisticalData (Disk Stats) (7.9.20)	Conditional	<p>Conditional requirement: Disk Drive statistics support. This is mandatory if CIM_BlockStatisticsCapabilities.ElementTypesSupported = "10".</p> <p>This associates a BlockStorageStatisticalData instance to the StorageExtent (Disk Drive) for which the statistics are collected.</p>
CIM_HostedService (7.9.21)	Mandatory	This associates the BlockStatisticsService to the ComputerSystem that hosts it.
CIM_HostedCollection (Provider Supplied) (7.9.22)	Mandatory	This would associate the StatisticsCollection to the top level system for the profile (e.g., array).
CIM_HostedCollection (Default) (7.9.23)	Mandatory	This would associate a default BlockStatisticsManifestCollection to the top level system for the profile (e.g., array).
CIM_HostedCollection (Client Defined) (7.9.24)	Optional	This would associate a client defined BlockStatisticsManifestCollection to the top level system for the profile (e.g., array).
CIM_MemberOfCollection (Member of statistics collection) (7.9.25)	Mandatory	This would associate all block statistics instances to the StatisticsCollection.
CIM_MemberOfCollection (Member of pre-defined collection) (7.9.26)	Mandatory	This would associate pre-defined Manifests to default manifest collection.

Table 73: CIM Elements for Block Server Performance

Element Name	Requirement	Description
CIM_MemberOfCollection (Member of client defined collection) (7.9.27)	Conditional	Conditional requirement: Clients can modify manifests as identified by CIM_BlockStatisticsCapabilities.SynchronousMethodsSupported. This would associate Manifests to client defined manifest collections.

7.9.1 CIM_AssociatedBlockStatisticsManifestCollection (Provider defined collection)

The CIM_AssociatedBlockStatisticsManifestCollection associates an instance of a CIM_BlockStatisticsManifestCollection to the instance of CIM_StatisticsCollection to which it applies. The default manifest collection defines the CIM_BlockStorageStatisticalData properties that are supported by the profile implementation.

CIM_AssociatedBlockStatisticsManifestCollection is not subclassed from anything.

One instance of the CIM_AssociatedBlockStatisticsManifestCollection shall exist for the default manifest collection if the Block Server Performance Subprofile is implemented.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 74 describes class CIM_AssociatedBlockStatisticsManifestCollection (Provider defined collection).

Table 74: SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Provider defined collection)

Properties	Flags	Requirement	Description & Notes
Statistics		Mandatory	The StatisticsCollection to which the manifest collection applies
ManifestCollection		Mandatory	The default manifest collection.

7.9.2 CIM_AssociatedBlockStatisticsManifestCollection (Client defined collection)

The CIM_AssociatedBlockStatisticsManifestCollection associates an instance of a CIM_BlockStatisticsManifestCollection to the instance of CIM_StatisticsCollection to which it applies. Client defined manifest collections identify the Manifests (properties) for retrieval of block statistics.

CIM_AssociatedBlockStatisticsManifestCollection is not subclassed from anything.

There will be one instance of the CIM_AssociatedBlockStatisticsManifestCollection class, for each client defined manifest collection that has been created.

Created By: Extrinsic: CreateManifestCollection

Modified By: Static

Deleted By: Static

Class Mandatory: ClientManSync

Table 75 describes class CIM_AssociatedBlockStatisticsManifestCollection (Client defined collection).

Table 75: SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Client defined collection)

Properties	Flags	Requirement	Description & Notes
Statistics		Mandatory	The StatisticsCollection to which the manifest collection applies
ManifestCollection		Mandatory	A client defined manifest collection.

7.9.3 CIM_BlockStatisticsManifest (Provider Support)

The CIM_BlockStatisticsManifest class is Concrete class that defines the CIM_BlockStorageStatisticalData properties that supported by the Provider. These Manifests are established by the Provider for the default manifest collection.

CIM_BlockStatisticsManifest is subclassed from CIM_ManagedElement.

At least one Provider supplied instance of the CIM_BlockStatisticsManifest class shall exist, if the Block Server Performance Subprofile is supported.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 76 describes class CIM_BlockStatisticsManifest (Provider Support).

Table 76: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Provider Support)

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A Provider defined string that identifies the manifest in the context of the Default Manifest Collection.
InstanceID		Mandatory	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.

Table 76: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Provider Support)

Properties	Flags	Requirement	Description & Notes
ElementType		Mandatory	This value is required AND the current version of SMI-S specifies the following values: ValueMap {"2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"} Values { "Computer System", "Front-end Computer System", "Peer Computer System", "Back-end Computer System", "Front-end Port", "Back-end Port", "Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group"}
IncludeStatisticTime		Mandatory	
IncludeTotalIOs		Mandatory	
IncludeKBytesTransferred		Mandatory	
IncludeIOTimeCounter		Mandatory	
IncludeReadIOs		Mandatory	
IncludeReadHitIOs		Mandatory	
IncludeReadIOTimeCounter		Mandatory	
IncludeReadHitIOTimeCounter		Mandatory	
IncludeKBytesRead		Mandatory	
IncludeWriteIOs		Mandatory	
IncludeWriteHitIOs		Mandatory	
IncludeWriteIOTimeCounter		Mandatory	
IncludeWriteHitIOTimeCounter		Mandatory	
IncludeKBytesWritten		Mandatory	
IncludeIdleTimeCounter		Mandatory	
IncludeMaintOp		Mandatory	
IncludeMaintTimeCounter		Mandatory	
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile

Table 76: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Provider Support)

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

7.9.4 CIM_BlockStatisticsManifest (Client Defined)

The CIM_BlockStatisticsManifest class is Concrete class that defines the CIM_BlockStorageStatisticalData properties that should be returned on a GetStatisticsCollection request.

CIM_BlockStatisticsManifest is subclassed from CIM_ManagedElement.

In order for a client defined instance of the CIM_BlockStatisticsManifest class to exist, the all the manifest collection manipulation functions shall be identified in the "SynchronousMethodsSupported" property of the CIM_BlockStatisticsCapabilities (BlockStatisticsCapabilities.SynchronousMethodsSupported = "6") instance, AND a client must have created at least ONE instance of CIM_BlockStatisticsManifestCollection.

Created By: Extrinsic: AddOrModifyManifest

Modified By: Extrinsic: AddOrModifyManifest

Deleted By: Extrinsic: RemoveManifest

Class Mandatory: SynchMM

Table 77 describes class CIM_BlockStatisticsManifest (Client Defined).

Table 77: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Client Defined)

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	A Client defined string that identifies the manifest.
InstanceID		Mandatory	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.
ElementType		Mandatory	This value is required AND the current version of SMI-S specifies the following values: ValueMap {"2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"} Values { "Computer System", "Front-end Computer System", "Peer Computer System", "Back-endComputer System", "Front-end Port", "Back-end Port", "Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group"}
IncludeStatisticTime		Mandatory	
IncludeTotalIOs		Mandatory	
IncludeKBytesTransferred		Mandatory	

Table 77: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Client Defined)

Properties	Flags	Requirement	Description & Notes
IncludeIOTimeCounter		Mandatory	
IncludeReadIOs		Mandatory	
IncludeReadHitIOs		Mandatory	
IncludeReadIOTimeCounter		Mandatory	
IncludeReadHitIOTimeCounter		Mandatory	
IncludeKBytesRead		Mandatory	
IncludeWriteIOs		Mandatory	
IncludeWriteHitIOs		Mandatory	
IncludeWriteIOTimeCounter		Mandatory	
IncludeWriteHitIOTimeCounter		Mandatory	
IncludeKBytesWritten		Mandatory	
IncludeIdleTimeCounter		Mandatory	
IncludeMaintOp		Mandatory	
IncludeMaintTimeCounter		Mandatory	
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
IncludeStartStatisticTime	N	Optional	Not Specified in this version of the Profile

7.9.5 CIM_BlockStatisticsManifestCollection (Provider Defined)

An instance of a default CIM_BlockStatisticsManifestCollection defines the set of Manifests that define the properties supported for each ElementType supported for the implementation. It can also be used by clients in retrieval of Block statistics by the GetStatisticsCollection method.

CIM_BlockStatisticsManifestCollection is subclassed from CIM_SystemSpecificCollection.

At least ONE CIM_BlockStatisticsManifestCollection shall exist if the Block Server Performance Subprofile is implemented. This would be the default manifest collection that defines the properties supported by the implementation.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 78 describes class CIM_BlockStatisticsManifestCollection (Provider Defined).

Table 78: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Provider Defined)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	For the default manifest collection, this should be set to "DEFAULT".
IsDefault		Mandatory	Denotes whether or not this manifest collection is a provider defined default manifest collection. For the default manifest collection this is set to "true".
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile

7.9.6 CIM_BlockStatisticsManifestCollection (Client Defined)

An instance of a client defined CIM_BlockStatisticsManifestCollection defines the set of Manifests to be used in retrieval of Block statistics by the GetStatisticsCollection method.

CIM_BlockStatisticsManifestCollection is subclassed from CIM_SystemSpecificCollection.

In order for a client defined instance of the CIM_BlockStatisticsManifestCollection class to exist, then all the manifest collection manipulation functions shall be identified in the "SynchronousMethodsSupported" property of the CIM_BlockStatisticsCapabilities instance and a client must have created a Manifest Collection..

Created By: Extrinsic: CreateManifestCollection

Modified By: Static

Deleted By: Static

Class Mandatory: ClientManSync

Table 79 describes class CIM_BlockStatisticsManifestCollection (Client Defined).

Table 79: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Client Defined)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	A client defined user-friendly name for the manifest collection. It is set during creation of the Manifest Collection through the ElementName parameter of the CreateManifestCollection method.

Table 79: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Client Defined)

Properties	Flags	Requirement	Description & Notes
IsDefault		Mandatory	Denotes whether or not this manifest collection is a provider defined default manifest collection. For the client defined manifest collections this is set to "false".
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile

7.9.7 CIM_BlockStorageStatisticalData

The CIM_BlockStorageStatisticalData class defines the block statistics properties that may be kept for an metered element of the block storage entity (such as a ComputerSystem, StorageVolume, Port or Disk Drive).

CIM_BlockStorageStatisticalData is subclassed from CIM_StatisticalData.

Instances of this class will exist for each of the metered elements if the 'ElementTypesSupported' property of the CIM_BlockStatisticsCapabilities indicates that the metered element is supported. For example, 'Computer System' is identified in the 'ElementTypesSupported' property, then this indicates support for metering of the Top level computer system or 'Component Computer System'.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 80 describes class CIM_BlockStorageStatisticalData.

Table 80: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
StatisticTime		Mandatory	Time statistics table by object was last updated (Time Stamp in CIM 2.2 specification format)
ElementType		Mandatory	This value is required AND current version of SMI-S specifies the following values: ValueMap {"2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"} Values { "Computer System", "Front-end Computer System", "Peer Computer System", "Back-end Computer System", "Front-end Port", "Back-end Port", "Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group"}
TotalIOs		Mandatory	The cumulative count of I/Os for the object.

Table 80: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Properties	Flags	Requirement	Description & Notes
KBytesTransferred		Conditional	<p>Conditional requirement: This property is required if the ElementType is 2, 6, 8, 9, 10, 11 or 12.. The cumulative count of data transferred in KBytes (1024bytes = 1KByte).</p> <p>Note: This is mandatory for the Top level computer system, Front-end Ports, Volumes, Extents, Disk Drives, ArbitraryLUs and Remote Replica Groups, but is optional for the component computer systems and Back-end Ports.</p>
IOTimeCounter		Optional	<p>The cumulative elapsed I/O time(number of Clock Tick Intervals) for all cumulative I/Os as defined in "Total I/Os" above. (I/O response time is added to this counter at the completion of each measured I/O using ClockTickInterval units. This value can be divided by number of IOs to obtain an average response time.</p> <p>Note: This is not SPECIFIED for CompositeExtents, ArbitraryLUs or Remote Replica Groups..</p>
ReadIOs		Conditional	<p>Conditional requirement: This property is required if the ElementType is 3, 4, 8 or 10.. The cumulative count of all reads.</p> <p>Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, Volumes and Disk Drives, but it is optional for the Top level computer system.</p> <p>Note: This is not specified for Ports, CompositeExtents, "Back-end" component computer systems, ArbitraryLUs or Remote Replica Groups..</p>
ReadHitIOs		Conditional	<p>Conditional requirement: This property is required if the ElementType is 3, 4 or 8.. The cumulative count of all read cache hits (Reads from Cache).</p> <p>Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, and Volumes, but it is optional for the Top level computer system.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, DiskDrives, ArbitraryLUs or Remote Replica Groups.</p>
ReadIOTimeCounter		Optional	<p>The cumulative elapsed time for all Read I/Os) for all cumulative Read I/Os.</p> <p>Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system, Volumes and Disk Drives.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, ArbitraryLUs or Remote Replica Groups.</p>

Table 80: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Properties	Flags	Requirement	Description & Notes
ReadHitIOTimeCounter		Optional	<p>The cumulative elapsed time for all Read I/Os read from cache for all cumulative Read I/Os.</p> <p>Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and Volumes.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, DiskDrives, ArbitraryLUs or Remote Replica Groups.</p>
KBytesRead		Optional	<p>The cumulative count of data read in KBytes (1024bytes = 1KByte).</p> <p>Note: This is optional for all ComputerSystems, Volumes, and Disk Drives.</p> <p>Note: This is not specified for Ports, CompositeExtents, ArbitraryLUs or Remote Replica Groups..</p>
WriteIos		Conditional	<p>Conditional requirement: This property is required if the ElementType is 3, 4 or 8.. The cumulative count of all writes.</p> <p>Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems and Volumes, but it is optional for the Top level computer system and Disk Drives.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, ArbitraryLUs or Remote Replica Groups.</p>
WriteHitIos		Conditional	<p>Conditional requirement: This property is required if the ElementType is 3, 4 or 8.. The cumulative count of Write Cache Hits (Writes that went directly to Cache without blocking).</p> <p>Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems and Volumes, but it is optional for the Top level computer system.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, DiskDrives, ArbitraryLUs or Remote Replica Groups.</p>
WriteIOTimeCounter		Optional	<p>The cumulative elapsed time for all Write I/Os for all cumulative Writes.</p> <p>Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and Volumes and Disks Drives.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, ArbitraryLUs or Remote Replica Groups.</p>

Table 80: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Properties	Flags	Requirement	Description & Notes
WriteHitIOTimeCounter		Optional	<p>The cumulative elapsed time for all Write I/Os written to cache for all cumulative Write I/Os.</p> <p>Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and Volumes.</p> <p>Note: This is not specified for "Back-end" component computer systems, Ports, CompositeExtents, DiskDrives, ArbitraryLUs or Remote Replica Groups.</p>
KBytesWritten		Optional	<p>The cumulative count of data written in KBytes (1024bytes = 1KByte).</p> <p>Note: This is optional for all ComputerSystems, Volumes and Disk Drives.</p> <p>Note: This is not specified for Ports, CompositeExtents, ArbitraryLUs or Remote Replica Groups.</p>
IdleTimeCounter		Optional	<p>The cumulative elapsed idle time using ClockTickInterval units (Cumulative Number of Time Units for all idle time in the array).</p> <p>Note: This is optional for "Back-end" component ComputerSystems, Front end Ports, Volumes, Extents and Disk Drives.</p> <p>Note: This is not specified for back-end Ports, Top level computer system, "Front-end" and "Peer" component computer systems, ArbitraryLUs or Remote Replica Groups.</p>
MaintOp		Optional	<p>The cumulative count of all disk maintenance operations (SCSI commands such as: Verify, skip-mask, XOR read, XOR write-read, etc.) This is needed to understand the load on the disks that may interfere with normal read and write operations.</p> <p>Note: This is optional for Extents and Disk Drives.</p> <p>Note: This is not specified for ComputerSystems, Ports, Volumes, ArbitraryLUs or Remote Replica Groups.</p>
MaintTimeCounter		Optional	<p>The cumulative elapsed disk maintenance time. maintenance response time is added to this counter at the completion of each measured maintenance operation using ClockTickInterval units.</p> <p>Note: This is optional for Extents and Disk Drives.</p> <p>Note: This is not specified for ComputerSystems, Ports, Volumes, ArbitraryLUs or Remote Replica Groups.</p>
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile

Table 80: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Properties	Flags	Requirement	Description & Notes
ElementName	N	Optional	Not Specified in this version of the Profile
StartStatisticTime	N	Optional	Not Specified in this version of the Profile
ResetSelectedStats()		Optional	Not Specified in this version of the Profile

7.9.8 CIM_BlockStatisticsCapabilities

An instance of the CIM_BlockStatisticsCapabilities class defines the specific support provided with the block statistics implementation. Note: There would be zero or one instance of this class in a profile. There would be none if the profile did not support the Block Server Performance Subprofile. There would be exactly one instance if the profile did support the Block Server Performance Subprofile.

CIM_BlockStatisticsCapabilities class is subclassed from CIM_Capabilities.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 81 describes class CIM_BlockStatisticsCapabilities.

Table 81: SMI Referenced Properties/Methods for CIM_BlockStatisticsCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
ElementTypesSupported		Mandatory	ValueMap { "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12" }, Values {"Computer System", "Front-end Computer System", "Peer Computer System", "Back-end Computer System", "Front-end Port", "Back-endPort", "Volume", "Extent", "Disk Drive", "Arbitrary LUs", "Remote Replica Group"}
SynchronousMethodsSupported		Mandatory	This property is mandatory, but the array may be empty. ValueMap { "2", "3", "4", "5", "6", "7", "8"}, Values {"Exec Query", "Indications", "QueryCollection", "GetStatisticsCollection", "Manifest Creation", "Manifest Modification", "Manifest Removal" }
AsynchronousMethodsSupported		Optional	Not supported in current version of SMI-S.

Table 81: SMI Referenced Properties/Methods for CIM_BlockStatisticsCapabilities

Properties	Flags	Requirement	Description & Notes
ClockTickInterval		Mandatory	An internal clocking interval for all timers in the subsystem, measured in microseconds (Unit of measure in the timers, measured in microseconds). Time counters are monotonically increasing counters that contain "ticks". Each tick represents one ClockTickInterval. If ClockTickInterval contained a value of 32 then each time counter tick would represent 32 microseconds.
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile
CreateGoalSettings()		Optional	Not Specified in this version of the Profile

7.9.9 CIM_BlockStatisticsService

The CIM_BlockStatisticsService class provides methods for statistics retrieval and Manifest Collection manipulation.

The CIM_BlockStatisticsService class is subclassed from CIM_Service.

There shall be an instance of the CIM_BlockStatisticsService, if the Block Server Performance Subprofile is implemented. It is not necessary to support any methods of the service, but the service shall be populated.

The methods that are supported can be determined from the SynchronousMethodsSupported and AsynchronousMethodsSupported properties of the CIM_BlockStatisticsCapabilities.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 82 describes class CIM_BlockStatisticsService.

Table 82: SMI Referenced Properties/Methods for CIM_BlockStatisticsService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
Name		Mandatory	
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

Table 82: SMI Referenced Properties/Methods for CIM_BlockStatisticsService

Properties	Flags	Requirement	Description & Notes
OperationalStatus	N	Optional	Not Specified in this version of the Profile
StatusDescriptions	N	Optional	Not Specified in this version of the Profile
InstallDate	N	Optional	Not Specified in this version of the Profile
HealthState	N	Optional	Not Specified in this version of the Profile
TimeOfLastStateChange	N	Optional	Not Specified in this version of the Profile
OtherEnabledState	N	Optional	Not Specified in this version of the Profile
EnabledDefault	N	Optional	Not Specified in this version of the Profile
RequestedState	N	Optional	Not Specified in this version of the Profile
EnabledState	N	Optional	Not Specified in this version of the Profile
Started	N	Optional	Not Specified in this version of the Profile
PrimaryOwnerName	N	Optional	Not Specified in this version of the Profile
PrimaryOwnerContact	N	Optional	Not Specified in this version of the Profile
GetStatisticsCollection()		Optional	Support for this method is optional. This method retrieves all statistics kept for the profile as directed by a manifest collection.
CreateManifestCollection()		Optional	Support for this method is optional. This method is used to create client defined manifest collections.
AddOrModifyManifest()		Optional	Support for this method is optional. This method is used to add or modify block statistics manifests in a client defined manifest collection.
RemoveManifests()		Optional	Support for this method is optional. This method is used to remove a block statistics manifest from a client defined manifest collection.
RequestStateChange()		Optional	Not Specified in this version of the Profile
StopService()		Optional	Not Specified in this version of the Profile
StartService()		Optional	Not Specified in this version of the Profile

7.9.10 CIM_StatisticsCollection

The CIM_StatisticsCollection collects all block statistics kept by the profile. There is one instance of the CIM_StatisticsCollection class and all individual element statistics can be accessed by using association traversal(using MemberOfCollection) from the StatisticsCollection.

CIM_StatisticsCollection is subclassed from CIM_SystemSpecificCollection.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 83 describes class CIM_StatisticsCollection.

Table 83: SMI Referenced Properties/Methods for CIM_StatisticsCollection

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
SampleInterval		Mandatory	Minimum recommended polling interval for an array, storage virtualizer system or volume manager. It is set by the provider and cannot be modified.
TimeLastSampled		Mandatory	Time statistics table by object was last updated (Time Stamp in SMI 2.2 specification format)
Caption	N	Optional	Not Specified in this version of the Profile
Description	N	Optional	Not Specified in this version of the Profile

7.9.11 CIM_ElementCapabilities

CIM_ElementCapabilities represents the association between ManagedElements (i.e., CIM_BlockStatisticsService) and their Capabilities (e.g., CIM_BlockStatisticsCapabilities). Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementCapabilities association for the referenced instance of Capabilities. ElementCapabilities describes the existence requirements and context for the referenced instance of ManagedElement. Specifically, the ManagedElement shall exist and provides the context for the Capabilities.

CIM_ElementCapabilities is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 84 describes class CIM_ElementCapabilities.

Table 84: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The managed element (BlockStatisticsService)
Capabilities		Mandatory	The Capabilities instance associated with the BlockStatisticsService.

7.9.12 CIM_ElementStatisticalData (Top Level System Stats)

CIM_ElementStatisticalData is an association that relates a top level ComputerSystem to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific ComputerSystem.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: SystemStats

Table 85 describes class CIM_ElementStatisticalData (Top Level System Stats).

Table 85: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Top Level System Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to the top level ComputerSystem for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the ComputerSystem.

7.9.13 CIM_ElementStatisticalData (Component System Stats)

CIM_ElementStatisticalData is an association that relates a component ComputerSystem to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific component ComputerSystem.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ComponentSystemStats

Table 86 describes class CIM_ElementStatisticalData (Component System Stats).

Table 86: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Component System Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a component ComputerSystem for which the Statistics apply.

Table 86: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Component System Stats)

Properties	Flags	Requirement	Description & Notes
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the ComputerSystem.

7.9.14 CIM_ElementStatisticalData (Volume Stats)

CIM_ElementStatisticalData is an association that relates a StorageVolume to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific volume.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: VolumeStats

Table 87 describes class CIM_ElementStatisticalData (Volume Stats).

Table 87: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Volume Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a StorageVolume for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the StorageVolume.

7.9.15 CIM_ElementStatisticalData (Logical Disk Stats)

CIM_ElementStatisticalData is an association that relates a LogicalDisk to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific logical disk.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: VolumeStats

Table 88 describes class CIM_ElementStatisticalData (Logical Disk Stats).

Table 88: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Logical Disk Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a LogicalDisk for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the LogicalDisk.

7.9.16 CIM_ElementStatisticalData (Front end Port Stats)

CIM_ElementStatisticalData is an association that relates a target port to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific target port.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: FEPortStats

Table 89 describes class CIM_ElementStatisticalData (Front end Port Stats).

Table 89: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Front end Port Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a target port for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the Port.

7.9.17 CIM_ElementStatisticalData (Back end Port Stats)

CIM_ElementStatisticalData is an association that relates a back end port to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific back end port.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: BEPortStats

Table 90 describes class CIM_ElementStatisticalData (Back end Port Stats).

Table 90: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Back end Port Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a back end port for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the Port.

7.9.18 CIM_ElementStatisticalData (Remote Copy Stats)

CIM_ElementStatisticalData is an association that relates a Network to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific Network.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: RemoteCopyStats

Table 91 describes class CIM_ElementStatisticalData (Remote Copy Stats).

Table 91: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Remote Copy Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a Network (remote replication group) for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the Network.

7.9.19 CIM_ElementStatisticalData (Extent Stats)

CIM_ElementStatisticalData is an association that relates a StorageExtent (CompositeExtent) to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific StorageExtent.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentStats

Table 92 describes class CIM_ElementStatisticalData (Extent Stats).

Table 92: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Extent Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a StorageExtent for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the StorageExtent.

7.9.20 CIM_ElementStatisticalData (Disk Stats)

CIM_ElementStatisticalData is an association that relates a StorageExtent (Disk Drive) to its statistics. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the CIM_ElementStatisticalData association for the referenced instance of BlockStatistics. ElementStatisticalData describes the existence requirements and context for the BlockStatistics, relative to a specific StorageExtent of a Disk Drive.

CIM_ElementStatisticalData is not subclassed from anything.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: DiskDriveStats

Table 93 describes class CIM_ElementStatisticalData (Disk Stats).

Table 93: SMI Referenced Properties/Methods for CIM_ElementStatisticalData (Disk Stats)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	A reference to a Disk Drive StorageExtent for which the Statistics apply.
Stats		Mandatory	A reference to the BlockStorageStatisticalData that hold the statistics for the Disk Drive.

7.9.21 CIM_HostedService

CIM_HostedService is an association between a Service (CIM_BlockStatisticsService) and the System (ComputerSystem) on which the functionality resides. Services are weak with respect to their hosting System. Heuristic: A Service is hosted on the System where the LogicalDevices or SoftwareFeatures that implement the Service are located.

CIM_HostedService is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 94 describes class CIM_HostedService.

Table 94: SMI Referenced Properties/Methods for CIM_HostedService

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	The Service hosted on the System.

7.9.22 CIM_HostedCollection (Provider Supplied)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Block Server Performance Subprofile, it is used to associate the StatisticsCollection to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 95 describes class CIM_HostedCollection (Provider Supplied).

Table 95: SMI Referenced Properties/Methods for CIM_HostedCollection (Provider Supplied)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The top level ComputerSystem of the profile.
Dependent		Mandatory	The StatisticsCollection.

7.9.23 CIM_HostedCollection (Default)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Block Server Performance Subprofile, it is used to associate the default BlockStatisticsManifestCollection to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 96 describes class CIM_HostedCollection (Default).

Table 96: SMI Referenced Properties/Methods for CIM_HostedCollection (Default)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The top level ComputerSystem of the profile.
Dependent		Mandatory	The provider defined BlockStatisticsManifestCollection.

7.9.24 CIM_HostedCollection (Client Defined)

CIM_HostedCollection defines a SystemSpecificCollection in the context of a scoping System. It represents a Collection that only has meaning in the context of a System, and/or whose elements are restricted by the definition of the System. In the Block Server Performance Subprofile, it is used to associate a client defined BlockStatisticsManifestCollections to the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 97 describes class CIM_HostedCollection (Client Defined).

Table 97: SMI Referenced Properties/Methods for CIM_HostedCollection (Client Defined)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The top level ComputerSystem of the profile.
Dependent		Mandatory	A client defined BlockStatisticsManifestCollection.

7.9.25 CIM_MemberOfCollection (Member of statistics collection)

This use of MemberOfCollection is to collect all BlockStorageStatisticalData instances (in the StatisticsCollection). Each association is created as a side effect of the metered object getting created.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 98 describes class CIM_MemberOfCollection (Member of statistics collection).

Table 98: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of statistics collection)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The default manifest collection
Member		Mandatory	The individual Manifest Instance that is part of the set.

7.9.26 CIM_MemberOfCollection (Member of pre-defined collection)

This use of MemberOfCollection is to Collect all Manifests instances in the default manifest collection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 99 describes class CIM_MemberOfCollection (Member of pre-defined collection).

Table 99: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of pre-defined collection)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The provider defined default manifest collection
Member		Mandatory	The individual Manifest Instance that is part of the set.

7.9.27 CIM_MemberOfCollection (Member of client defined collection)

This use of MemberOfCollection is to Collect all Manifests instances in a client defined manifest collection.

Created By: Extrinsic: AddOrModifyManifest

Modified By: Static

Deleted By: Extrinsic: RemoveManifest

Class Mandatory: SynchMM

Table 100 describes class CIM_MemberOfCollection (Member of client defined collection).

Table 100: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Member of client defined collection)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	A client defined manifest collection
Member		Mandatory	The individual Manifest Instance that is part of the set.

STABLE

EXPERIMENTAL

Clause 8: CKD Block Services Profile

8.1 Description

8.1.1 Synopsis

Profile Name: CKD Block Services

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.13

Central Class: CIM_StoragePool

Scoping Class: CIM_System

8.1.2 Overview

The CKD Block Services Profile models CKD (Count Key Data) storage of a block server storage system. CKD storage is storage that is formatted to support Count and Key fields to support mainframe access. CKD storage is at the StorageVolume level (which means the StorageVolume is access using single byte FC protocols) or at the StoragePool level (that is, a StoragePool may be dedicated to holding CKD StorageVolumes).

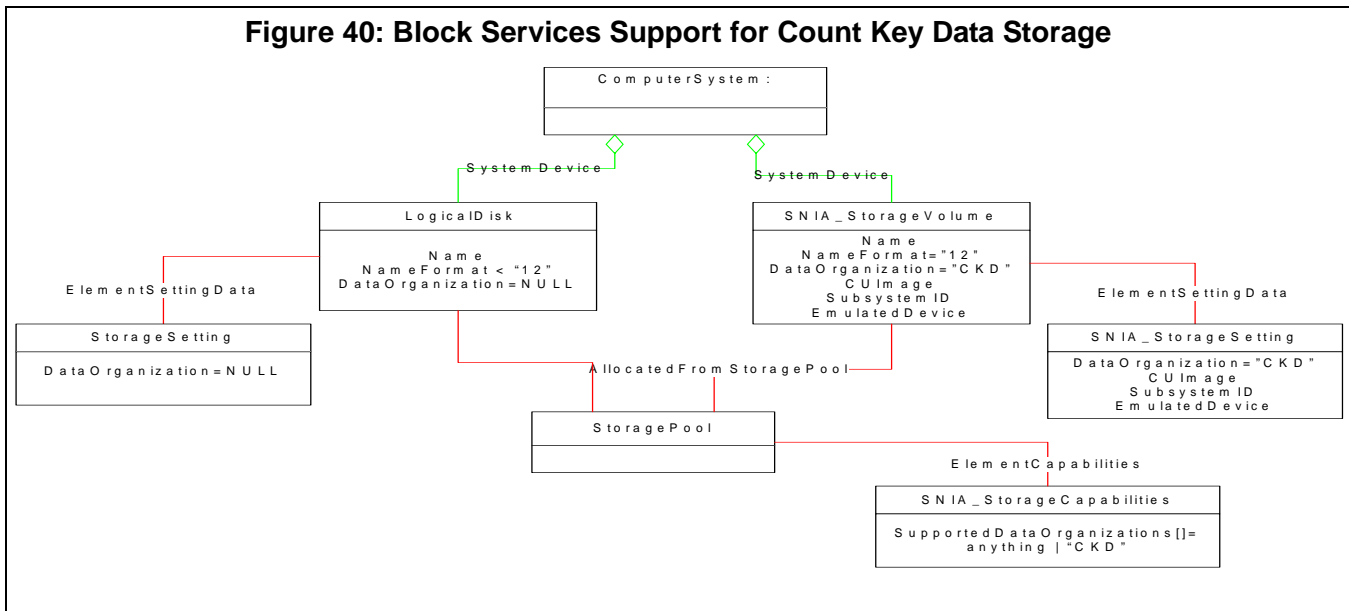
The CKD Block Services Profile is a component profile (subprofile) that provides a way for storage profiles to model mainframe storage. With this support a client will be able to distinguish non-CKD storage that is provided for non-CKD access from CKD storage that is provided for mainframe access. This is an important distinction for management, since storage that is available to one (e.g., SCSI access) is typically not usable by the other (e.g., mainframe access). Similarly, management functions for other functions of block servers (e.g., masking and mapping) are somewhat different for CKD storage than non-CKD storage. So, it is important for management applications to be aware of the distinctions.

The CKD Block Services requires and specializes the Block Services Package. That is, the functions of the Block Services Package apply for CKD storage as well as non-CKD storage. The CKD Block Services Profile extends the model for CKD storage.

8.1.3 Implementation

8.1.3.1 Block Services Support for CKD Storage

Some profile implementations may support Extended Count Key Data formatted storage. This support is provided using existing classes, but adds some new properties as illustrated in Figure 40.

Figure 40: Block Services Support for Count Key Data Storage

CKD storage may apply to StoragePools (via StorageCapabilities), StorageVolumes or LogicalDisks. CKD storage is indicated by the DataOrganization property in StorageVolume and LogicalDisk classes. For SMI-S the values of this property shall be "4" for CKD Volumes (or LogicalDisks). The capability of a StoragePool to support either (or both) non-CKD or CKD volumes is indicated by the SupportedDataOrganizations[] property of StorageCapabilities associated to the StoragePool.

DataOrganization can be specified on StorageSetting to indicate that an CKD Volume is desired on either of the Volume creation methods. If this property is left as null, it will be set according to the StoragePool that is being used. If the StoragePool supports both non-CKD and CKD storage, then the default will be to create a non-CKD volume (or LogicalDisk) for backward compatibility. This property exists in StorageVolume, LogicalDisk, and StorageSetting classes.

An additional difference between non-CKD and CKD Volumes are the NameFormats supported. For CKD Volumes, the volumes follow a Node Element Descriptor (NED) format. For non-CKD volumes there are a variety of formats that may be supported.

There is also a CUImage property on both the SNIA_StorageVolume and the SNIA_StorageSetting. In the SB architecture and CKD access the CKD Volume has a "home" ProtocolController (in a Masking and Mapping sense). This property is covered in more detail in (need a Masking and Mapping reference here). But an CKD Volume cannot exist without an associated CUImage (ProtocolController). This is accommodated by the CUImage property on StorageSetting. That is, on creation of an CKD Volume the CUImage parameter is passed as part of the StorageSetting for the Volume being created. The CUImage in the SNIA_StorageSetting is the CUImage requested and the CUImage in the SNIA_StorageVolume is the CUImage assigned. CUImage is not supported for LogicalDisks.

A host can see more than 16 CU images by changing the SSID associated with the image. For example, there can be two CU images with the same image number but with different SSIDs. Thus, the same CU image numbers can be in use multiple times within the array and the host as long as each image has a unique SubsystemID. The second CU image with the same number is known as a "split."

Mainframe systems use the SubsystemID to locate physical disk controllers, and all devices in the CU image shall have the same SubsystemID. If the CU image that is specified does not exist yet, the SubsystemID of the first device is used as the SubsystemID of the CU image. If the CU image already exists and contains other devices

(and thus a SubsystemID), the SubsystemIDs of the newly mapped devices are changed to match the existing SubsystemID of the CU image.

8.1.3.2 Use Cases for CKD Storage

8.1.3.2.1 Summarize Pools and Capacities by SupportedDataOrganizations

Primordial StoragePools may be capable of supporting non-CKD, CKD or both non-CKD and CKD storage. This can be determined by inspecting the SupportedDataOrganizations property of the StorageCapabilities of the primordial StoragePool. If the property is NULL or not "4", then the pool only supports non-CKD storage and all concrete StoragePools allocated from this Primordial StoragePool shall only support non-CKD storage. Similarly, if the property only identifies "4" (Count Key Data), then the pool only supports CKD storage and all concrete StoragePools allocated from this primordial StoragePool shall only support CKD storage.

If the StorageCapabilities.SupportedDataOrganizations property for primordial StoragePool identifies both "4" (Count Key Data) and something else (including NULL), then the storage allocated from the pool can be either non-CKD or CKD storage. It will be necessary to follow the AllocatedFromStoragePool association to the concrete StoragePools above the primordial StoragePool. As the client moves up the AllocatedFromStoragePool association, it would keep track of the SpaceConsumed value in the AllocatedFromStoragePool. If all concrete StoragePools are also capable of both non-CKD and CKD storage, then the primordial capacity of the storage is considered capable of supporting both non-CKD and CKD Volumes (or LogicalDisks).

If, however, the client reaches a concrete StoragePool that is only capable of supporting non-CKD or CKD storage, then the SpaceConsumed value by that StoragePool would be considered either non-CKD or CKD. It may be necessary to "pro-rate" the SpaceConsumed value to determine the actual primordial storage that has been allocated to non-CKD or CKD.

8.1.3.2.2 Find the Capacity of CKD Capable Storage

Building on the previous use case, a client would determine the capacity of primordial StoragePools that are only CKD capable (that is, StorageCapabilities.SupportedDataOrganization = "4" and only "4"). This capacity is dedicated to CKD storage.

Next the client would consider primordial StoragePools that are capable of both non-CKD and CKD storage. The client would inspect the concrete StoragePools that are allocated from those primordial StoragePools. If any are identified as CKD only, the SpaceConsumed property on the AllocatedFromStoragePool will indicate the primordial storage that is dedicated to CKD.

If the concrete StoragePool just above the primordial StoragePool is also capable of supporting non-CKD or CKD storage, divide the SpaceConsumed value by the TotalManagedSpace value of the concrete StoragePool and save this "multiplier".

The client would continue executing the previous step until it finds a concrete StoragePool that only supports non-CKD storage. At this point, the client would multiply all the multipliers it has saved away to derive the amount of primordial space that has been dedicated to non-CKD storage. This value would be subtracted from the TotalManagedSpace value of the primordial StoragePool to determine the primordial capacity available for CKD storage. The client would execute this logic on all upper level concrete StoragePools that are identified as non-CKD only to get the remaining primordial capacity available for CKD storage.

8.1.3.2.3 Create an CKD Volume

To create an CKD Volume (or LogicalDisk) a client would create a StorageSetting (or select a SettingAssociated to Capabilities) with DataOrganization set to "4" and the CUIImage set to a valid CUIImage value.

With the appropriate CKD Volume Setting the client would issue either CreateOrModifyElementFromStoragePool or CreateOrModifyElementFromElements.

8.2 Health and Fault Management Consideration

No change for CKD.

8.3 Cascading Considerations

No change for CKD.

8.4 Supported Profiles, Subprofiles, and Packages

Table 101: Supported Profiles for CKD Block Services

Registered Profile Names	Mandatory	Version
Job Control	No	1.2.0
Extent Composition	No	1.2.0
Block Services	Yes	1.2.0

8.5 Methods of the Profile

All methods of the Block Services Package should work for CKD storage (subject to restrictions of particular profile implementations).

8.6 Client Considerations and Recipes

No change for CKD.

8.7 Registered Name and Version

CKD Block Services version 1.2.0

Specialized SNIA Block Services version 1.2.0

8.8 CIM Elements

Table 102: CIM Elements for CKD Block Services

Element Name	Requirement	Description
CIM_StoragePool (Primordial) (8.8.1)	Mandatory	The primordial StoragePool. It is created by the provider and cannot be deleted or modified. It cannot be used to allocate any storage element other than concrete StoragePools.
CIM_StoragePool (Concrete) (8.8.2)	Mandatory	The concrete StoragePool. A concrete StoragePool shall be allocated from another StoragePool. It shall be used for allocating StorageVolumes and LogicalDisks as well as other concrete StoragePools.
CIM_HostedStoragePool (8.8.3)	Mandatory	
CIM_HostedService (8.8.4)	Optional	
CIM_StorageConfigurationCapabilities (8.8.5)	Optional	
CIM_StorageConfigurationService (8.8.6)	Optional	
CIM_OwningJobElement (8.8.7)	Conditional	Conditional requirement: Support for Job Control profile.
SNIA_StorageCapabilities (8.8.8)	Mandatory	
SNIA_StorageSetting (8.8.9)	Mandatory	
CIM_StorageSettingWithHints (8.8.10)	Optional	
CIM_ElementSettingData (8.8.11)	Mandatory	
CIM_ElementCapabilities (StorageCapabilities to StoragePool) (8.8.12)	Mandatory	Associates StorageCapabilities with StoragePool.
CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService) (8.8.13)	Mandatory	Associates StorageConfigurationCapabilities with StorageConfigurationService.
CIM_AllocatedFromStoragePool (Pool from Pool) (8.8.14)	Mandatory	AllocatedFromStoragePool
CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool) (8.8.15)	Conditional	AllocatedFromStoragePool
SNIA_StorageVolume (8.8.16)	Conditional	Representation of a virtual disk (for SCSI, a logical unit). A StorageVolume is allocated from a concrete StoragePool. See the "Standard Formats for Logical Unit Names" section in the Storage Management Technical Specification, Part 1 Common Architecture for details on how to set Name, NameFormat, and NameNamespace properties.
CIM_SystemDevice (System to StorageVolume or LogicalDisk) (8.8.17)	Mandatory	Associates top level system from Array, Virtualizer, ... to StorageVolume or LogicalDisk

Table 102: CIM Elements for CKD Block Services

Element Name	Requirement	Description
CIM_BasedOn (StorageVolume to extent from Extent Composition) (8.8.18)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_LogicalDisk (8.8.19)	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. A LogicalDisk is allocated from a concrete StoragePool.
CIM_BasedOn (LogicalDisk to extent from Extent Composition) (8.8.20)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_StorageSettingsAssociatedToCapabilities (8.8.21)	Optional	This class associates the StorageCapabilities with the preset setting. Any StorageSetting instance associated with this association shall work, unmodified, to create a storage element. The preset settings should not change overtime and represent possible settings for storage elements are set of design time rather than runtime. All StorageSetting instances linked with this association shall have a ChangeableType of "0" ("Fixed - Not Changeable").
CIM_StorageSettingsGeneratedFromCapabilities (8.8.22)	Optional	This class associates the StorageCapabilities with the StorageSetting generated from it via the CreateSetting method. StorageSettings instances generated in this manner, as identified with this association, may be removed from the model at any time by the implementation if the ChangeableType of the associated setting is set to "2" ("Changeable - Transient"). All StorageSettings associated with this class shall be changeable, ChangeableType is "2" or "3". Some implementations may permit the modification of the ChangeableType property itself on StorageSetting instances associated via this class. Provided this is allowed, an client may change the ChangeableType to "3" ("Changeable - Persistent") to have this setting retained either after generation of the instance or after its modification by the client. The DefaultSetting property of the StorageSetting instances linked with this association is meaningless.
CIM_ConcreteComponent (storage element to extent from Extent Composition) (8.8.23)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_AssociatedComponentExtent (StoragePool to extent from Composition) (8.8.24)	Conditional	Conditional requirement: Support for Extent Composition profile.

Table 102: CIM Elements for CKD Block Services

Element Name	Requirement	Description
CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition) (8.8.25)	Conditional	Conditional requirement: Support for Extent Composition profile.
CIM_EnabledLogicalElementCapabilities (8.8.26)	Optional	This class is used to express the naming and possible requested state change possibilities for storage elements.
CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool) (8.8.27)	Optional	Associates EnabledLogicalElementCapabilities with StorageConfigurationService.
CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk) (8.8.28)	Optional	Associates EnabledLogicalElementCapabilities with StorageConfigurationService.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool) (8.8.29)	Optional	Expressed the ability for the element to be named or have its state changed.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk) (8.8.30)	Optional	Expressed the ability for the element to be named or have its state changed.
CIM_ElementCapabilities (8.8.31)	Mandatory	The CIM_ElementCapabilities associates a StoragePool or StorageConfigurationService to its Capabilities (StorageCapabilities and StorageConfigurationCapabilities). There are no enhancements for CKD.
CIM_AllocatedFromStoragePool (8.8.32)	Mandatory	The CIM_AllocatedFromStoragePool associates a Storage Element (Volume, LogicalDisk or StoragePool) to its parent StoragePool. There are no enhancements for CKD.
CIM_StoragePool (8.8.33)	Mandatory	Primordial and Concrete Pools. These are unchanged for CKD storage.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Creation/Deletion of StoragePool
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Deletion of StoragePool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Creation of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Deletion of StorageVolume, if the StorageVolume storage element is implemented.

Table 102: CIM Elements for CKD Block Services

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Deprecated WQL - Change of status of a Storage Volume, if Storage Volume is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.CIM_StorageVolume::OperationalStatus <> PreviousInstance.CIM_StorageVolume::OperationalStatus	Optional	Experimental CQL - Change of status of a Storage Volume, if Storage Volume is implemented.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Creation of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deletion of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deprecated WQL - Change of status of LogicalDisk, if LogicalDisk is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Experimental CQL - Change of status of LogicalDisk, if LogicalDisk is implemented.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Creation/Deletion of StoragePool
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deletion of StoragePool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Creation of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Deletion of StorageVolume, if the StorageVolume storage element is implemented.

Table 102: CIM Elements for CKD Block Services

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Deprecated WQL - Change of status of a Storage Volume, if Storage Volume is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.CIM_StorageVolume::OperationalStatus <> PreviousInstance.CIM_StorageVolume::OperationalStatus	Optional	Experimental CQL - Change of status of a Storage Volume, if Storage Volume is implemented.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Creation of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deletion of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Deprecated WQL - Change of status of LogicalDisk, if LogicalDisk is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus <> PreviousInstance.CIM_LogicalDisk::OperationalStatus	Optional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Experimental CQL - Change of status of LogicalDisk, if LogicalDisk is implemented.

8.8.1 CIM_StoragePool (Primordial)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 103 describes class CIM_StoragePool (Primordial).

Table 103: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	Shall be true.
InstanceID		Mandatory	
ElementName		Optional	
PoolID		Mandatory	A unique name in the context of this system that identifies this Pool.
TotalManagedSpace		Mandatory	
RemainingManagedSpace		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
GetSupportedSizes()		Mandatory	List the discrete storage element sizes that can be created or expanded from this Pool.
GetSupportedSizeRange()		Mandatory	List the size ranges for storage element that can be created or expanded from this Pool.
GetAvailableExtents()		Optional	List the StorageExtents from this Pool that may be used to create or expand a storage element. The StorageExtents may not already be in use as supporting capacity for existing storage element.

8.8.2 CIM_StoragePool (Concrete)

Created By: Extrinsic: StorageConfigurationService.CreateOrModifyStoragePool

Modified By: Extrinsic: StorageConfigurationService.CreateOrModifyStoragePool

Deleted By: Extrinsic: StorageConfigurationService.DeleteStoragePool

Class Mandatory: Mandatory

Table 104 describes class CIM_StoragePool (Concrete).

Table 104: SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	Shall be false.
InstanceID		Mandatory	

Table 104: SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)

Properties	Flags	Requirement	Description & Notes
ElementName		Optional	
PoolID		Mandatory	A unique name in the context of this system that identifies this Pool.
TotalManagedSpace		Mandatory	
RemainingManaged Space		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
GetSupportedSizes()		Mandatory	List the discrete storage element sizes that can be created or expanded from this Pool.
GetSupportedSizeRange()		Mandatory	List the size ranges for storage element that can be created or expanded from this Pool.
GetAvailableExtents()		Optional	List the StorageExtents from this Pool that may be used to create or expand a storage element. The StorageExtents may not already be in use as supporting capacity for existing storage element.

8.8.3 CIM_HostedStoragePool

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 105 describes class CIM_HostedStoragePool.

Table 105: SMI Referenced Properties/Methods for CIM_HostedStoragePool

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

8.8.4 CIM_HostedService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 106 describes class CIM_HostedService.

Table 106: SMI Referenced Properties/Methods for CIM_HostedService

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	The Service hosted on the System.

8.8.5 CIM_StorageConfigurationCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 107 describes class CIM_StorageConfigurationCapabilities.

Table 107: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
SupportedStoragePoolFeatures		Mandatory	Lists what StorageConfigurationService functionalities are implemented. Matches 2 3 5 6 7 (InExtents or Single InPool or Storage Pool QoS Change or Storage Pool Capacity Expansion or Storage Pool Capacity Reduction).
SupportedSynchronousActions		Optional	Lists what actions, invoked through StorageConfigurationService methods, shall not produce Concrete jobs.
SupportedStorageElementTypes		Mandatory	Lists the type of storage elements that are supported by this implementation.
SupportedAsynchronousActions		Optional	Lists what actions, invoked through StorageConfigurationService methods, may produce Concrete jobs.

Table 107: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedStorageElementFeatures		Mandatory	Lists actions supported through the invocation of StorageService.CreateOrModifyElementFromStoragePool(). Matches 3 5 8 9 11 12 13 (StorageVolume Creation or StorageVolume Modification or LogicalDisk Creation or LogicalDisk Modification or Storage Element QoS Change or Storage Element Capacity Expansion or Storage Element Capacity Reduction).
SupportedStorageElementUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on supported storage elements.
ClientSettableElementUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on the usage of client-settable elements.
SupportedStoragePoolUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on storage pools.
ClientSettablePoolUsage		Optional	Indicates the intended usage or any restrictions that may have been imposed on the usage of a client-settable storage pool.

8.8.6 CIM_StorageConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 108 describes class CIM_StorageConfigurationService.

Table 108: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
CreationClassName		Mandatory	
SystemName		Mandatory	
Name		Mandatory	
CreateOrModifyStoragePool()		Mandatory	Create (or modify) a StoragePool. A job may be created as well.
DeleteStoragePool()		Mandatory	Start a job to delete a StoragePool.

Table 108: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Properties	Flags	Requirement	Description & Notes
CreateOrModifyElementFromStoragePool()		Mandatory	Create or modify a storage element. A job may be created as well.
CreateOrModifyElementFromElements()		Optional	Create or modify a storage element using component StorageExtents of the Pool. A job may be created as well.
ReturnToStoragePool()		Mandatory	Release the capacity represented by this storage element back to the Pool.
RequestUsageChange()		Optional	Allows a client to change the Usage for the element.
GetElementsBasedOnUsage()		Optional	Allows a client to retrieve elements for a specialized Usage.

8.8.7 CIM_OwningJobElement

Conditional on support for Job Control profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: JobControl

Table 109 describes class CIM_OwningJobElement.

Table 109: SMI Referenced Properties/Methods for CIM_OwningJobElement

Properties	Flags	Requirement	Description & Notes
OwnedElement		Mandatory	
OwningElement		Mandatory	

8.8.8 SNIA_StorageCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 110 describes class SNIA_StorageCapabilities.

Table 110: SMI Referenced Properties/Methods for SNIA_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
ElementType		Mandatory	
NoSinglePointOfFailure		Mandatory	
NoSinglePointOfFailureDefault		Mandatory	
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
DataRedundancyDefault		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	
PackageRedundancyDefault		Mandatory	
ExtentStripeLengthDefault		Optional	
ParityLayoutDefault		Optional	
UserDataStripeDepthDefault		Optional	
SupportedDataOrganizations		Mandatory	Supported values for SMI-S are "4" (Count Key Data) and anything else (including NULL) for non-CKD volumes. CKD Volumes use "4".
CreateSetting()		Mandatory	Generate a setting to use as a goal for creating or modifying storage elements.
GetSupportedStripeLengths()		Optional	List the possible discrete stripe lengths supported at this time of this method's execution.
GetSupportedStripeLengthRange()		Optional	List the possible stripe length ranges supported at the time of this method's execution
GetSupportedParityLayouts()		Optional	List the possible parity layouts supported at the time of this method's execution.
GetSupportedStripeDepths()		Optional	List the possible stripe depths supported at the time of this method's execution.

Table 110: SMI Referenced Properties/Methods for SNIA_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
GetSupportedStripeDepthRange()		Optional	List the possible stripe depth ranges supported at the time of this method's execution.

8.8.9 SNIA_StorageSetting

Created By: Extrinsic: StorageCapabilities.CreateSetting

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 111 describes class SNIA_StorageSetting.

Table 111: SMI Referenced Properties/Methods for SNIA_StorageSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
DataRedundancyGoal		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	
PackageRedundancyGoal		Mandatory	
ExtentStripeLength		Optional	
ExtentStripeLengthMin		Optional	
ExtentStripeLengthMax		Optional	
ParityLayout		Optional	
UserDataStripeDepth		Optional	

Table 111: SMI Referenced Properties/Methods for SNIA_StorageSetting

Properties	Flags	Requirement	Description & Notes
UserDataStripeDepthMin		Optional	
UserDataStripeDepthMax		Optional	
ChangeableType		Mandatory	
StorageExtentInitialUsage		Optional	The Usage value to be used when creating a new storage element.
StoragePoolInitialUsage		Optional	The Usage value to be used when creating a new storage pool.
DataOrganization		Mandatory	Supported value for CKD Volumes in SMI-S is "4" (Count Key Data). For non-CKD Volumes the property is either NULL or any value other than "4".
CUIImage		Optional	This property is the Node Element Descriptor of the Control Unit Image (this property is required for CKD StorageVolumes). It is not required for LogicalDisks.
SubsystemID		Optional	This property is the Subsystem ID if the array or virtualizer supports Subsystem IDs. If they are supported they would be required on volume creation.
EmulatedDevice		Optional	This string property specifies the specific device (e.g., 3380 or 3390) that is emulated by the volume.

8.8.10 CIM_StorageSettingWithHints

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 112 describes class CIM_StorageSettingWithHints.

Table 112: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	The user-friendly name for this instance of SettingData. In addition, the user friendly name can be used as a index property for a search of query. (Note: Name does not have to be unique within a namespace.)

Table 112: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Properties	Flags	Requirement	Description & Notes
NoSinglePointOfFailure		Mandatory	
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	
DataAvailabilityHint		Mandatory	This hint is an indication from a client of the importance placed on data availability. Values are 0=Don't Care to 10=Very Important.
AccessRandomnessHint		Mandatory	This hint is an indication from a client of the randomness of accesses. Values are 0=Entirely Sequential to 10=Entirely Random.
AccessDirectionHint		Mandatory	This hint is an indication from a client of the direction of accesses. Values are 0=Entirely Read to 10=Entirely Write
AccessSizeHint		Mandatory	This hint is an indication from a client of the optimal access sizes. Several sizes can be specified. Units("Megabytes")
AccessLatencyHint		Mandatory	This hint is an indication from a client how important access latency is. Values are 0=Don't Care to 10=Very Important.
AccessBandwidthWeight		Mandatory	This hint is an indication from a client of bandwidth prioritization. Values are 0=Don't Care to 10=Very Important.
StorageCostHint		Mandatory	This hint is an indication of the importance the client places on the cost of storage. Values are 0=Don't Care to 10=Very Important. A StorageVolume provider might choose to place data on low cost or high cost drives based on this parameter.
StorageEfficiencyHint		Mandatory	This hint is an indication of the importance placed on storage efficiency by the client. Values are 0=Don't Care to 10=Very Important. A StorageVolume provider might choose different RAID levels based on this hint.
ChangeableType		Mandatory	

8.8.11 CIM_ElementSettingData

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 113 describes class CIM_ElementSettingData.

Table 113: SMI Referenced Properties/Methods for CIM_ElementSettingData

Properties	Flags	Requirement	Description & Notes
IsDefault		Mandatory	An enumerated integer indicating that the referenced setting is a default setting for the element, or that this information is unknown. Value shall be 0,1 or 2 (Unknown or Is Default or Is Not Default).
IsCurrent		Mandatory	An enumerated integer indicating that the referenced setting is currently being used in the operation of the element, or that this information is unknown. Value shall be 0,1 or 2 (Unknown or Is Default or Is Not Default).
ManagedElement		Mandatory	StorageVolume or LogicalDisk
SettingData		Mandatory	The Setting Data object associated with the ManagedElement.

8.8.12 CIM_ElementCapabilities (StorageCapabilities to StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 114 describes class CIM_ElementCapabilities (StorageCapabilities to StoragePool).

Table 114: SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageCapabilities to StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element.

8.8.13 CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 115 describes class CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService).

Table 115: SMI Referenced Properties/Methods for CIM_ElementCapabilities (StorageConfigurationCapabilities to StorageConfigurationService)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

8.8.14 CIM_AllocatedFromStoragePool (Pool from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 116 describes class CIM_AllocatedFromStoragePool (Pool from Pool).

Table 116: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	Antecedent references the parent pool from which the dependent pool is allocated.
Dependent		Mandatory	

8.8.15 CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Array|Virt|HHRC

Table 117 describes class CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool).

Table 117: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Volume or LogicalDisk from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

8.8.16 SNIA_StorageVolume

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: Array|Virt

Table 118 describes class SNIA_StorageVolume.

Table 118: SMI Referenced Properties/Methods for SNIA_StorageVolume

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
ElementName		Optional	
Name	CD	Mandatory	An Identifier for this volume.
OtherIdentifyingInfo	CD	Optional	
IdentifyingDescriptions		Optional	
NameFormat		Mandatory	Format for Name property. For CKD Volumes, this shall be set to "12" (NED)
NameNamespace		Mandatory	The namespace that defines uniqueness for the NameFormat.
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

Table 118: SMI Referenced Properties/Methods for SNIA_StorageVolume

Properties	Flags	Requirement	Description & Notes
BlockSize		Mandatory	The BlockSize would report the number of bytes in a cylinder.
NumberOfBlocks		Mandatory	The number of blocks would be the number of cylinders.
ConsumableBlocks		Mandatory	The number of usable cylinders.
IsBasedOnUnderlyingRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
DataOrganization		Mandatory	Supported value for CKD Storage Volumes in SMI-S is "4" (Count Key Data). For non-CKD volumes the property is either NULL or any value other than "4".
CUIImage		Mandatory	This property is the Node Element Descriptor of the Control Unit Image (this property is required for CKD Volumes)
SubsystemID		Optional	This property is the Subsystem ID if the array or virtualizer supports Subsystem IDs. If they are supported they would be required on volume creation.
EmulatedDevice		Optional	This string property specifies the specific device (e.g., 3380 or 3390) that is emulated by the volume.

8.8.17 CIM_SystemDevice (System to StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: Mandatory

Table 119 describes class CIM_SystemDevice (System to StorageVolume or LogicalDisk).

Table 119: SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

8.8.18 CIM_BasedOn (StorageVolume to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 120 describes class CIM_BasedOn (StorageVolume to extent from Extent Composition).

Table 120: SMI Referenced Properties/Methods for CIM_BasedOn (StorageVolume to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	StorageExtent instance from Extent Composition
Dependent		Mandatory	

8.8.19 CIM_LogicalDisk

Created By: Static

Modified By: Static

Deleted By: Extrinsic: StorageConfigurationService.ReturnToStoragePool

Class Mandatory: LVM

Table 121 describes class CIM_LogicalDisk.

Table 121: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	

Table 121: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Optional	User-friendly name
Name		Mandatory	OS Device Name
NameFormat		Mandatory	Format for name
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	Value shall be 2 3 6 8 15 (OK or Degraded or Error or Starting or Dormant).
BlockSize		Mandatory	The BlockSize would report the number of bytes in a cylinder.
NumberOfBlocks		Mandatory	The number of blocks would be the number of cylinders.
ConsumableBlocks		Mandatory	The number of usable cylinders.
IsBasedOnUnderlyingRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	
Usage		Optional	The specialized usage intended for this element.
OtherUsageDescription		Optional	Set when Usage value is "Other".
ClientSettableUsage		Optional	Lists Usage values that can be set by a client for this element.
DataOrganization		Mandatory	Supported value for SMI-S is "4" (Count Key Data). Values that are not "4" are for non-CKD LogicalDisks. CKD LogicalDisks use "4".

8.8.20 CIM_BasedOn (LogicalDisk to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 122 describes class CIM_BasedOn (LogicalDisk to extent from Extent Composition).

Table 122: SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	StorageExtent instance from Extent Composition
Dependent		Mandatory	

8.8.21 CIM_StorageSettingsAssociatedToCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 123 describes class CIM_StorageSettingsAssociatedToCapabilities.

Table 123: SMI Referenced Properties/Methods for CIM_StorageSettingsAssociatedToCapabilities

Properties	Flags	Requirement	Description & Notes
DefaultSetting		Mandatory	This boolean designates the setting that will be used if the CreateSetting() method is called with providing the NewSetting parameter. However, some implementations may require that the NewSetting parameter be non null. There may be only one default setting per the combination of StorageCapabilities and associated StoragePool as associated through ElementCapabilities.
Dependent		Mandatory	The StorageSetting reference.
Antecedent		Mandatory	The StorageCapabilities reference.

8.8.22 CIM_StorageSettingsGeneratedFromCapabilities

Created By: Extrinsic: CreateSetting

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 124 describes class CIM_StorageSettingsGeneratedFromCapabilities.

Table 124: SMI Referenced Properties/Methods for CIM_StorageSettingsGeneratedFromCapabilities

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

8.8.23 CIM_ConcreteComponent (storage element to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 125 describes class CIM_ConcreteComponent (storage element to extent from Extent Composition).

Table 125: SMI Referenced Properties/Methods for CIM_ConcreteComponent (storage element to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	StoragePool instance from Extent Composition

8.8.24 CIM_AssociatedComponentExtent (StoragePool to extent from Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 126 describes class CIM_AssociatedComponentExtent (StoragePool to extent from Composition).

Table 126: SMI Referenced Properties/Methods for CIM_AssociatedComponentExtent (Storage-Pool to extent from Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	Parent StoragePool

Table 126: SMI Referenced Properties/Methods for CIM_AssociatedComponentExtent (Storage-Pool to extent from Composition)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	The referenced StorageExtent represents capacity has not been allocated, is allocated in part, or is allocated in its entirety.

8.8.25 CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition)

Conditional on support for Extent Composition Profile

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ExtentComposition

Table 127 describes class CIM_AssociatedRemainingExtent (StoragePool to extent from Extent Composition).

Table 127: SMI Referenced Properties/Methods for CIM_AssociatedRemainingExtent (Storage-Pool to extent from Extent Composition)

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	Parent StoragePool
PartComponent		Mandatory	The referenced StorageExtent represents the capacity of the StorageExtent on which it is based that was not used in resource allocation.

8.8.26 CIM_EnabledLogicalElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 128 describes class CIM_EnabledLogicalElementCapabilities.

Table 128: SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	The moniker for the instance.
ElementNameEditSupported		Mandatory	Denotes whether a storage element can be named

Table 128: SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities

Properties	Flags	Requirement	Description & Notes
MaxElementNameLength		Mandatory	Specifies the maximum length in glyphs (letters) for the name. See MOF for details.
ElementNameMask		Mandatory	The regular expression that specifies the possible content and format for the element name. See MOF for details
RequestedStatesSupported		Optional	Expresses the states to which this element may be changed using the RequestStateChange method. If this property, it may be assumed that the state may not be changed.

8.8.27 CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 129 describes class CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool).

Table 129: SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

8.8.28 CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 130 describes class CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk).

Table 130: SMI Referenced Properties/Methods for CIM_ElementCapabilities (Used to declare the naming capabilities of the StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

8.8.29 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 131 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool).

Table 131: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StoragePool)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	

8.8.30 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 132 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk).

Table 132: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.

Table 132: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageVolume or LogicalDisk)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	

8.8.31 CIM_ElementCapabilities

This class is unchanged from the Block Services Package.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

8.8.32 CIM_AllocatedFromStoragePool

This class is unchanged from the Block Services Package.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

8.8.33 CIM_StoragePool

This class is unchanged from the Block Services Package. Changes for StoragePools are in the StorageCapabilities associated to the StoragePools.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

EXPERIMENTAL

STABLE**Clause 9: Copy Services Subprofile****9.1 Description****9.1.1 Synopsis**

Profile Name: Copy Services

Version: 1.2.0

Organization: SNIA

CIM schema version: 2.11

Central Class: N/A

Scoping Class: ComputerSystem

9.1.2 Overview

The Copy Services Subprofile is an optional subprofile for the Array, Virtualization and Volume Manager profiles.

The subprofile defines a management interface for local and remote mirror management, local snapshot management and clone management. A provider may allow local snapshot management to use a remote mirror as a source element. This capability indirectly provides remote snapshot management.

The subprofile specification uses terminology consistent with the SNIA dictionary of storage networking except for the term clone. A clone is a fully copied replica the same size as the source element created with the intent of becoming an independent element.

Two types of synchronization views are supported. A replica may be synchronized to the current view of the source element or may be synchronized to a point-in-time view. Snapshots and clones always represent a point-in-time view of the source element. A mirror can represent either a current view or a point-in-time view as indicated by the synchronization state property of the association. A provider maintains a stateful view of a source element as long as the source and replica association is maintained. The synchronization view is modeled with a StorageSynchronized association. A client can determine the type and state of the synchronized view by inspecting properties of the association instance.

The subprofile supports two types of storage elements. Replicas can be instances of StorageVolume or LogicalDisk. The source and replica elements shall be the same element type. All of the instance diagrams that follow show StorageVolume replicas but apply equally to LogicalDisk replicas.

A copy service for storage elements deploys some type of copy engine. Copy techniques for storage elements include full background copy, copy-on-write and copy-on-read. Most aspects of copy engines are opaque to clients. A provider may allow the client to manage the copy engine for background copy operations. This optional capability is discussed in 9.6.8, "Managing Background Copy".

EXPERIMENTAL

The subprofile includes special considerations for remote replication. Local replication assumes that an associated source element and replica element are hosted in a single managed system such as an array platform. Remote replication assumes that source and replica elements are hosted in separate systems. The client shall discover both system elements whether controlled by a single SMI-S server/CIMOM or separate SMI-S server/CIMOMs. The client uses interfaces to both system instances but only invokes remote replication methods to a single

instance of `StorageConfigurationService`. The subprofile requires that any stitching is handled by a cascading provider when two SMI-S servers/CIMOMs are involved.

The subprofile includes a variable space consumption model that a provider may use for delta replica elements. Most storage elements receive a fixed allocation of space when the element is created and the consumed space is a contiguous block set. Delta replicas may not receive any space allocation when created and, subsequently, consume space one block at a time as the associated source element is updated. The resulting block set for a delta replica is typically scattered throughout a container element such as a storage pool.

EXPERIMENTAL

The Copy Services Subprofile enables a provider to deploy all of the modeled replication capabilities in a single service instance. For example, one service instance may support local mirrors, remote mirrors and delta snapshots. A client discovers and analyzes each of these capabilities as shown in Figure 41.

A provider exposes an instance of `StorageReplicationCapabilities` for each replication capability supported. The `CopyType` property as defined in `CIM_StorageSynchronized` describes the replication policies supported by the subprofile.

Async: Create and maintain an asynchronous mirror copy of the source. May be used to maintain a remote copy when the latency of a synchronous copy is unacceptable. Separate instances of `StorageReplicationCapabilities` may be defined for local and remote replication policies corresponding to this `CopyType` value.

Sync: Create and maintain a synchronous mirror copy of the source. Writes done to the source element are reflected to the mirror before signalling the host that the write is complete. Used to maintain a copy requiring guaranteed consistency during a recovery operation. Separate instances of `StorageReplicationCapabilities` may be defined for local and remote replication policies corresponding to this `CopyType` value.

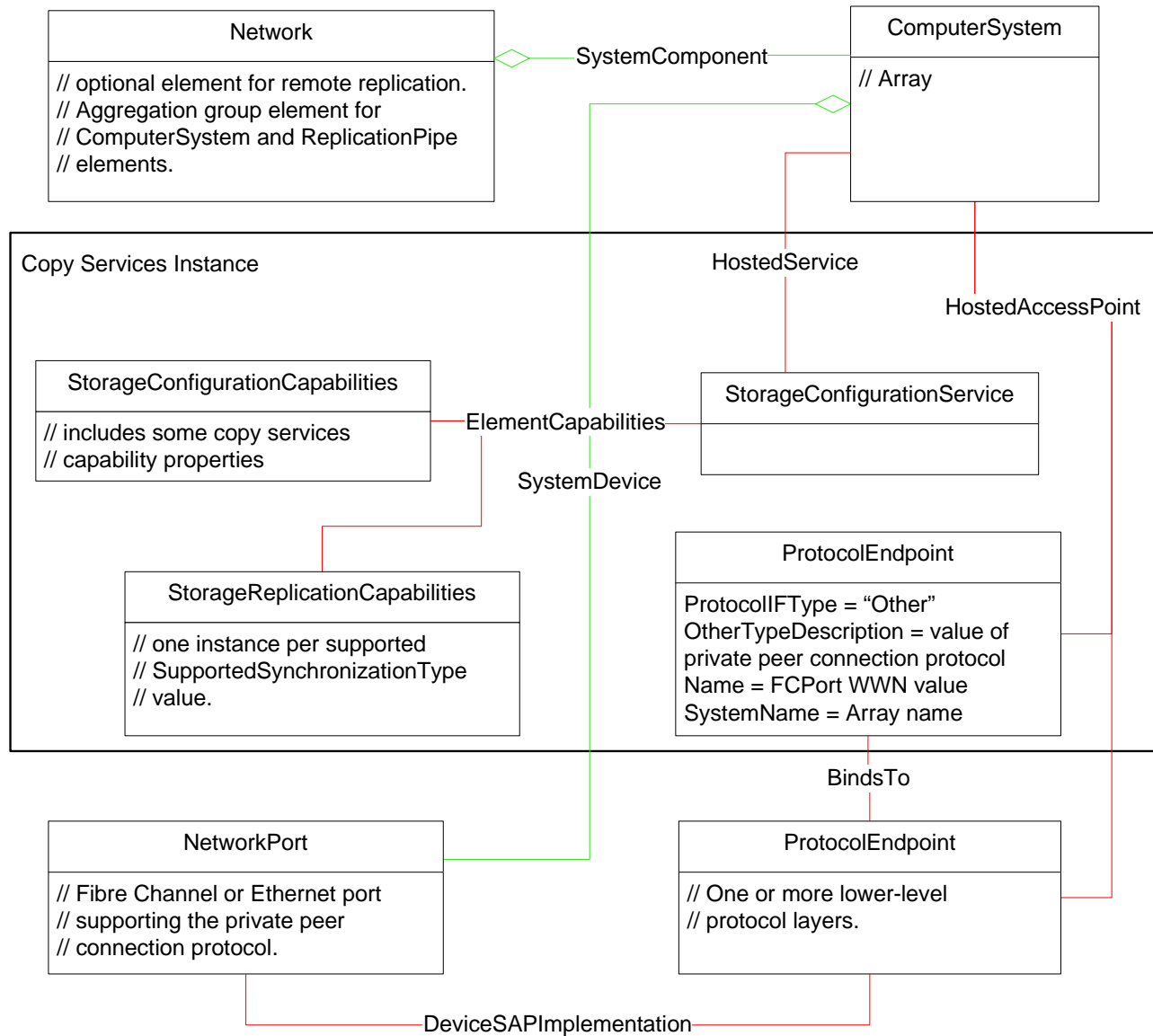
UnSyncAssoc: Creates an unsynchronized copy associated to the source element. This type of copy is called a “snapshot” and represents a point-in-time image of the source element. Separate instances of `StorageReplicationCapabilities` may be defined for full size snapshots and delta snapshots corresponding to this `CopyType` value.

UnSyncUnAssoc: Creates an unsynchronized clone of the source element and does not maintain the source association after completing the copy operation.

The `StorageReplicationCapabilities` class defines informational properties with un-modifiable values that guide a client using the various capabilities of the service. For example:

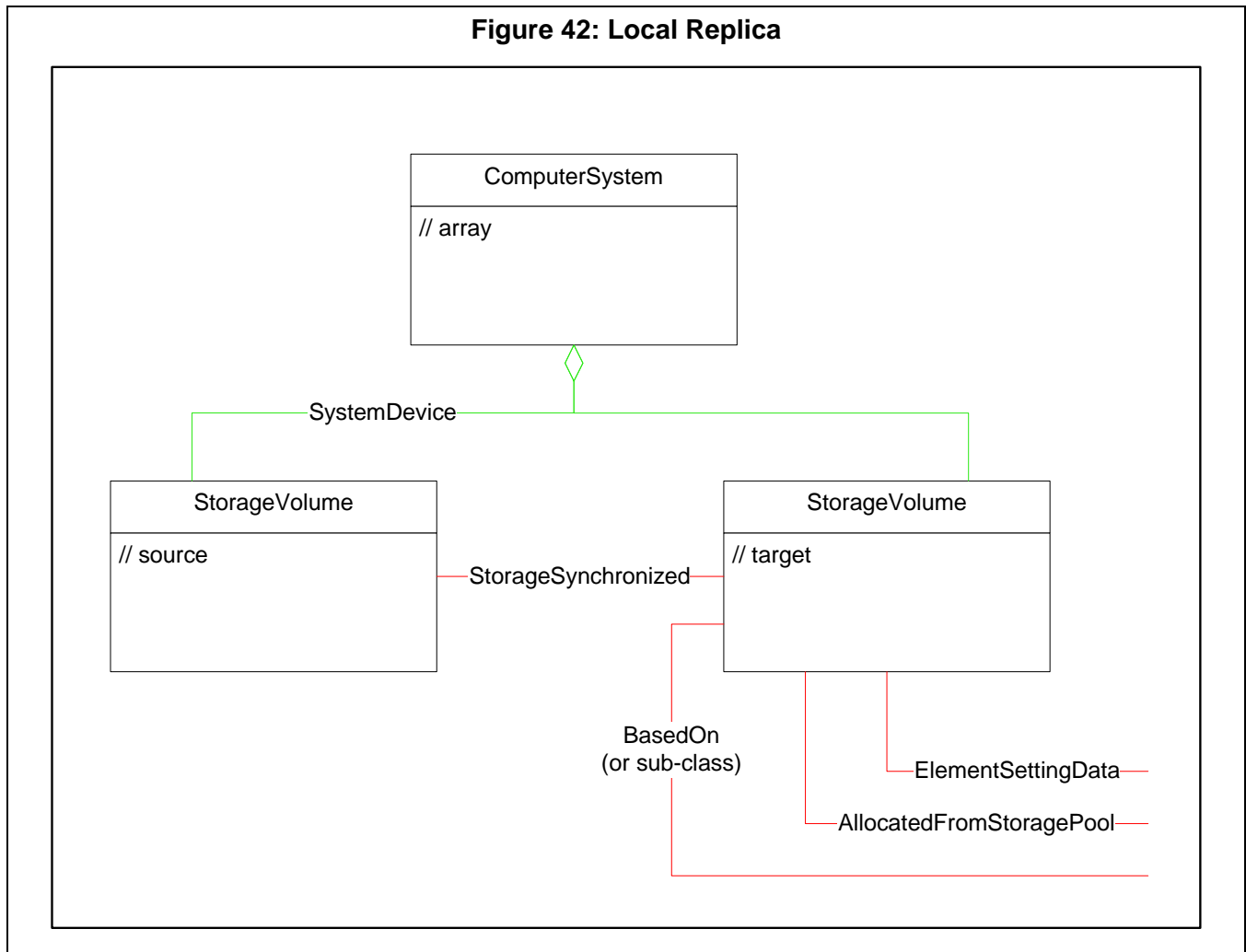
- Instance 1 defines the capability to create local mirrors. `SupportedSynchronizationType` is set to a value of “Sync” and the `AttachOrModifyReplica` method is the only method supported for mirror creation. The `InitialReplicaState` is “Synchronized”.
- Instance 2 defines the capability to create snapshots. `SupportedSynchronizationType` is set to a value of “UnSyncAssocDelta” and the `CreateReplica` method is the only method supported for snapshot creation. The `InitialReplicaState` is “Idle”.

Further details concerning discovery and the use of capability properties are included in the client considerations section. The extrinsic methods invoked to create and manage replicas are defined in the `StorageConfigurationService` class shown in the discovery instance diagram.

Figure 41: Copy Services Discovery

Discovered array provider supporting the copy services subprofile including remote replication capability. The Network, NetworkPort and ProtocolEndpoint elements are optional. These elements are exposed by a provider supporting managed peer-to-peer connections.

Figure 42 shows the basic model of a local replica.



A local replica is created by invoking either the `CreateReplica` or the `AttachOrModifyReplica` extrinsic methods. `CreateReplica` creates a new storage element in a storage pool. `AttachOrModifyReplica` transforms an existing, independent storage element into a replica. The new replica is the same element type as the source element. Several associations are implicitly created for all replica elements. A `StorageSynchronized` association shall be created if the new replica remains associated with its source element. A `SystemDevice` association shall be created or shall already exist. An `AllocatedFromStoragePool` association shall be created or shall already exist. An `ElementSettingData` association with an instance of `StorageSetting` is created or shall already exist for the replica element. An optional `BasedOn` association may exist if `AttachOrModifyReplica` is invoked to transform an existing element into an associated replica.

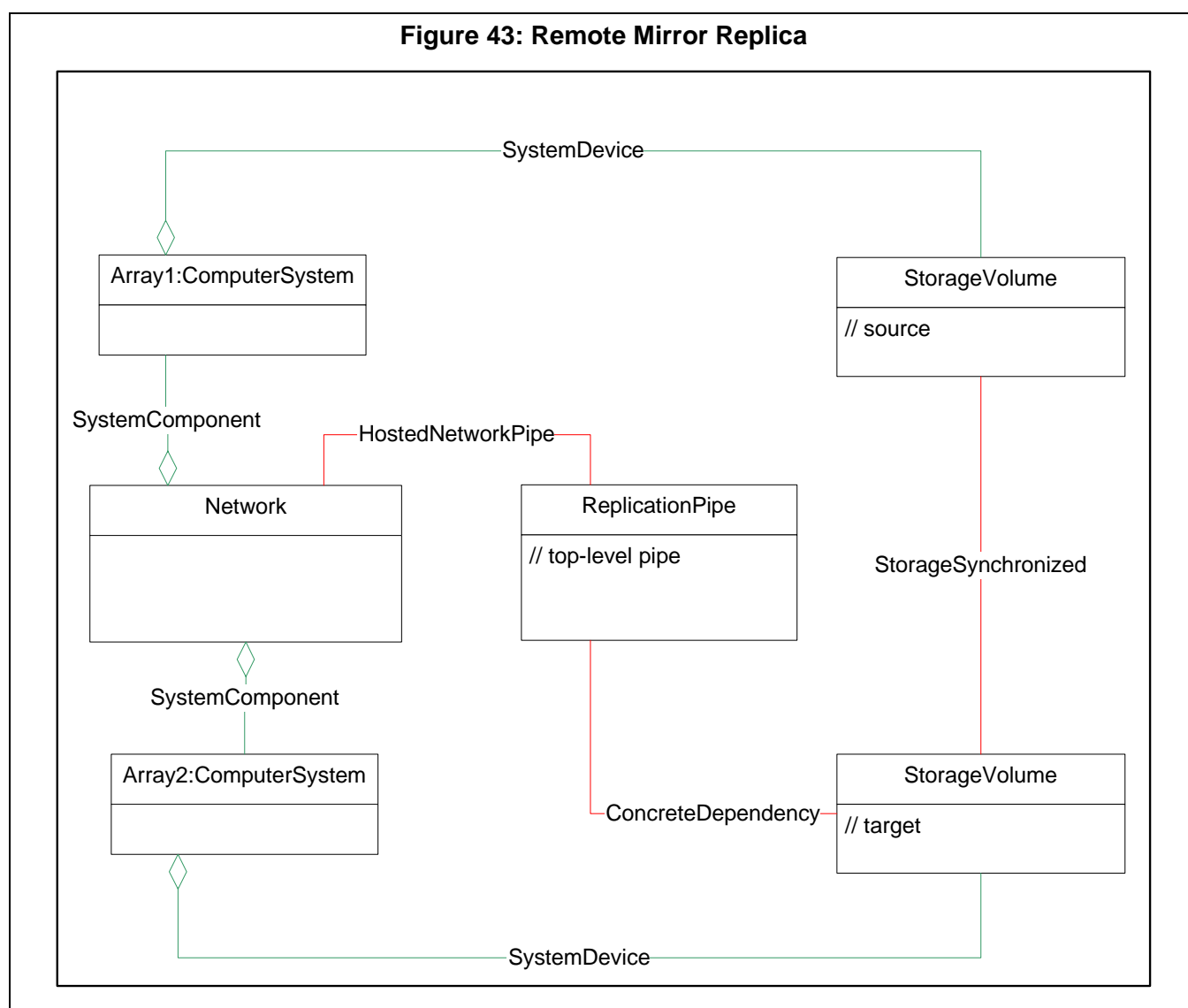
EXPERIMENTAL

A remote replica is created by invoking either the `CreateReplica` or the `AttachOrModifyReplica` extrinsic method. A peer-to-peer connection may be required by a provider before creating a remote replica. Peer-to-peer connections are explained later in this section. The basic remote replication model associates two existing storage elements with a new instance of `StorageSynchronized`. If the remote replica pair is managed within the context of a peer-to-peer connection, the target storage element is also associated to a `ReplicationPipe` element as shown in Figure 43.

The CreateReplica method allows a client to delegate the selection of a target element location and settings to the invoked provider. The client selects a source element for the replication operation and may optionally choose to supply a storage pool location and storage settings or to let the provider make the choices. The AttachOrModifyReplica method allows a client to completely manage the source/target replication pairing. The client creates a new target element or selects an existing element to be used as the target. Once the target element is prepared, the client invokes the AttachOrModifyReplica method and the provider pairs the source and target elements selected by the client. All providers shall support at least one of these two methods.

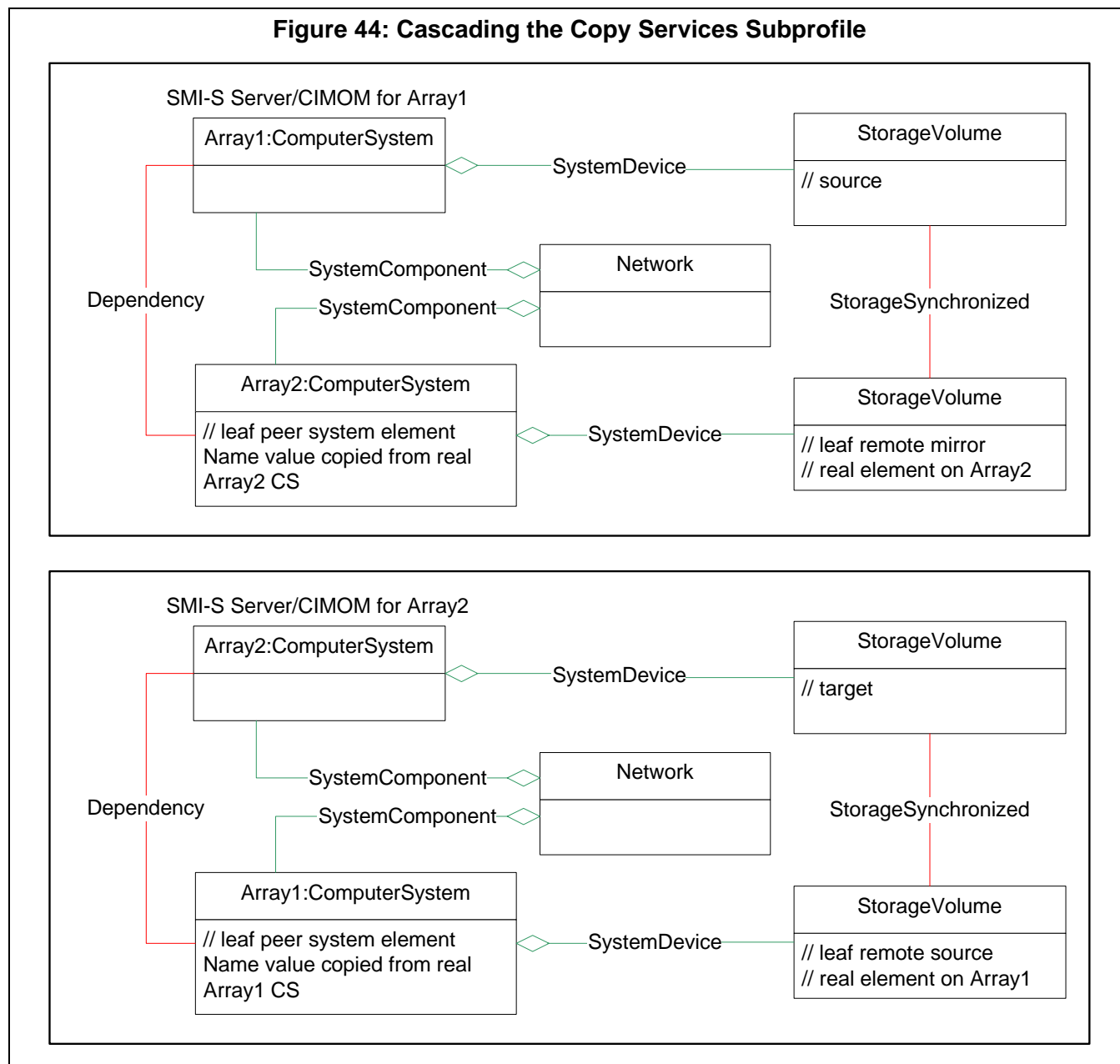
Instances of StorageConfigurationService shall exist for both the source system element and the target system element. The RemoteReplicationServicePointAccess property in StorageReplicationCapabilities indicates the service instance that is used for invocation of remote replication methods. The provider may indicate the service instance hosted on the source system or on the target system.

Figure 43 shows the basic model of a remote mirror replica when both source and target elements are controlled by one SMI-S server:



Remote replication may involve separate SMI-S servers/CIMOMs for each peer system element. A cascading provider shall provide stitching between the two SMI-S servers that enables a client to manage replica pairs and peer-to-peer connections through either server. This may be deployed using leaf elements as shown in Figure 44.

Figure 44: Cascading the Copy Services Subprofile

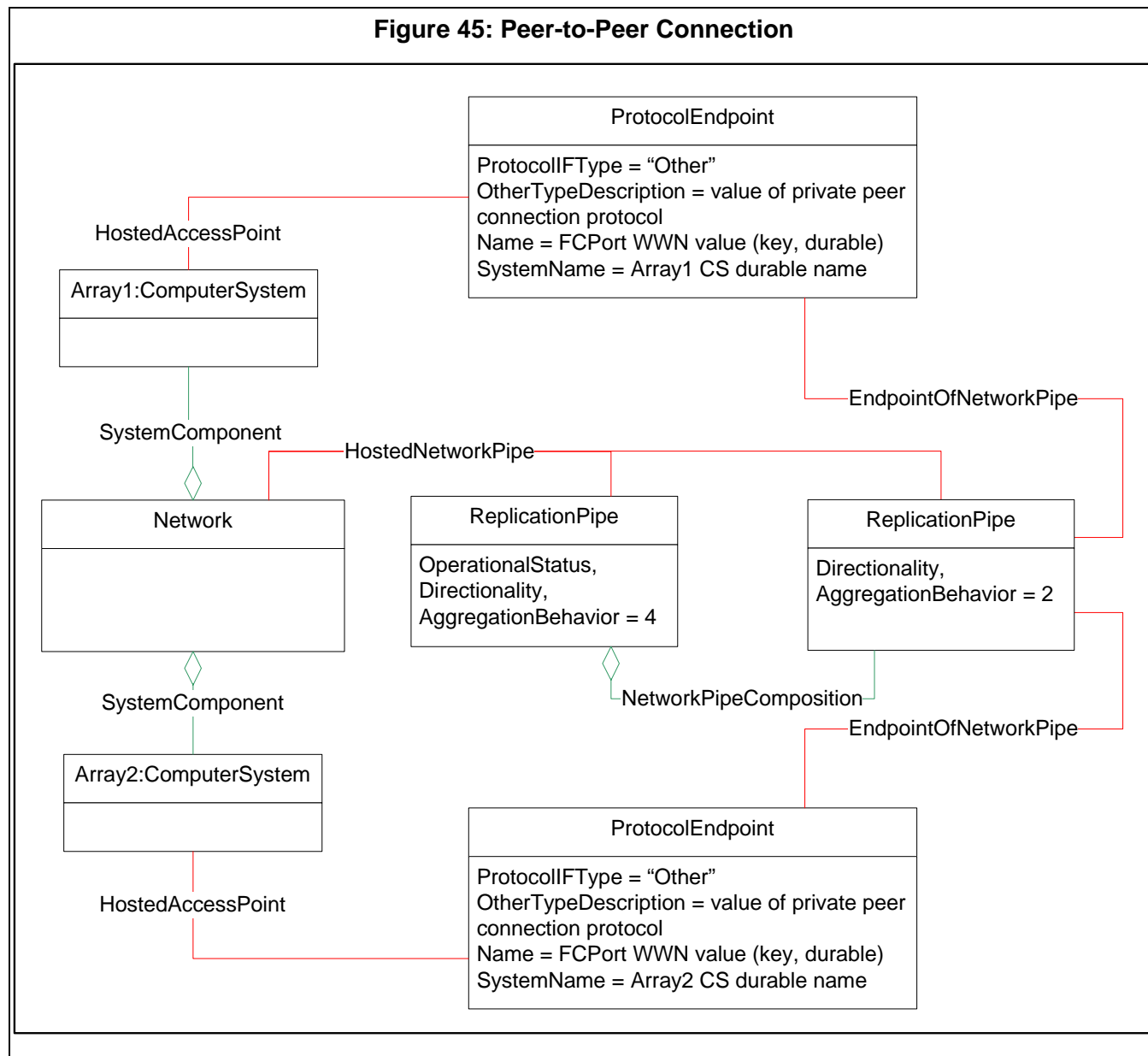


The provider ensures that leaf elements representing real instances of ComputerSystem, ProtocolEndpoint and an element such as StorageVolume are created in each CIMOM. Correlatable name properties and other key properties are copied from the real elements to the leaf elements. The provider shall ensure that state and status properties such as OperationalStatus and SyncState have consistent values between the leaf elements and real elements for all properties required by the subprofile.

Peer systems can be arrays, volume managers or any element type supporting the subprofile. Both peers shall be the same element type. Two peer systems are correlated using durable name properties when a connection is established. Real and leaf elements are correlated using durable name properties when leaf elements are created. The Name value of the real element is stored as the Name value of the leaf element. If a provider allows

connections to be monitored and managed, a special instance of `Network` is exposed to clients as an aggregation point for `ReplicationPipe` elements that identify connections. `ReplicationPipe` instances can be static or can be created by extrinsic method invocation. All connections (`ReplicationPipe` instances) and `ComputerSystem` instances supporting managed connections are associated to this `Network` element.

Figure 45 shows the complete model of a peer-to-peer replication connection:



Remote replication requires a transport connection between two peer systems before remote replicas can be created. These connections are called “peer-to-peer connections”. The Copy Services Subprofile does not provide for managing the topology of these connections. Any managed routing for switched connections shall be completed by an external action before establishing a connection. The underlying network and lower-level protocols are transparent to the peer-to-peer connection model. Any network protocols supported by SMI-S can bind to peer-to-peer protocol endpoints.

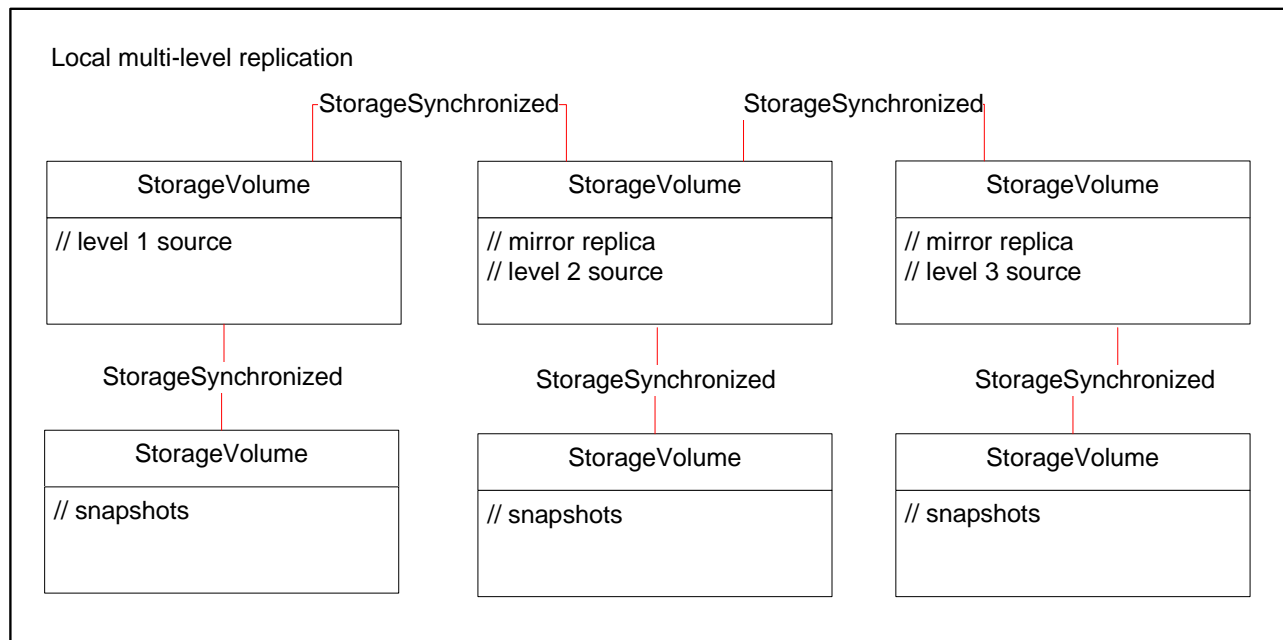
The subprofile allows use of either point-to-point or switched topologies for connections. Point-to-point connections are static connections that may be discovered and monitored by a client. Switched connections are dynamic connections and a client can manage the performance and availability characteristics of the connection.

Peer-to-peer connections may be uni-directional or bi-directional connections between two peer systems. One peer is the host of the source storage element and the other peer is the host of the replica target element. Peer systems can be either the top-level ComputerSystem in the array or a tiered ComputerSystem located by traversing a ComponentCS association from the top-level element. Managed connections use a special instance of Network to aggregate all of the system elements supporting remote replication. An established connection is modeled as a two-level ReplicationPipe composition. The top-level ReplicationPipe provides an operational status property and a directionality property. The peer-to-peer ProtocolEndpoint instances correlate the source and target peer systems using the SystemName key property. This correlation is necessary because a peer can establish connections with many other peers. A provider can support remote replication without using connections. AttachOrModifyReplica method providers receive a REF to a top-level ReplicationPipe if connections are supported. The peer-to-peer ProtocolEndpoint instances may associate to port elements or may bind to lower-level ProtocolEndpoint instances. The provider may dedicate these ports for peer-to-peer connections or may allow the ports to be shared with host connections. This behavior is opaque to clients.

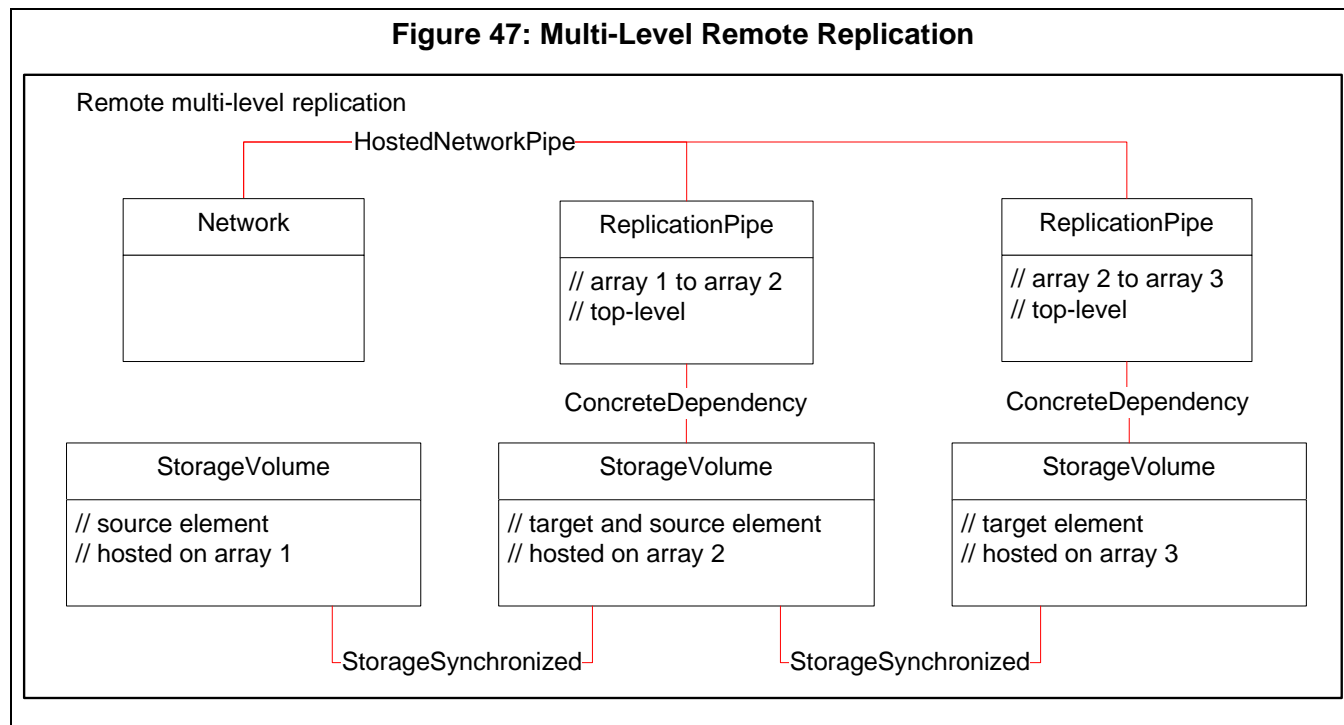
EXPERIMENTAL

The subprofile supports both multiple replicas per associated source element and multi-level replication. Properties in StorageReplicationCapabilities allow the provider to indicate the maximum number of replicas for one source element and the maximum depth for multi-level replication. Figure 46 and Figure 47 show the basic models for local multi-level replication and remote multi-level replication.

Figure 46: Multi-Level Local Replication

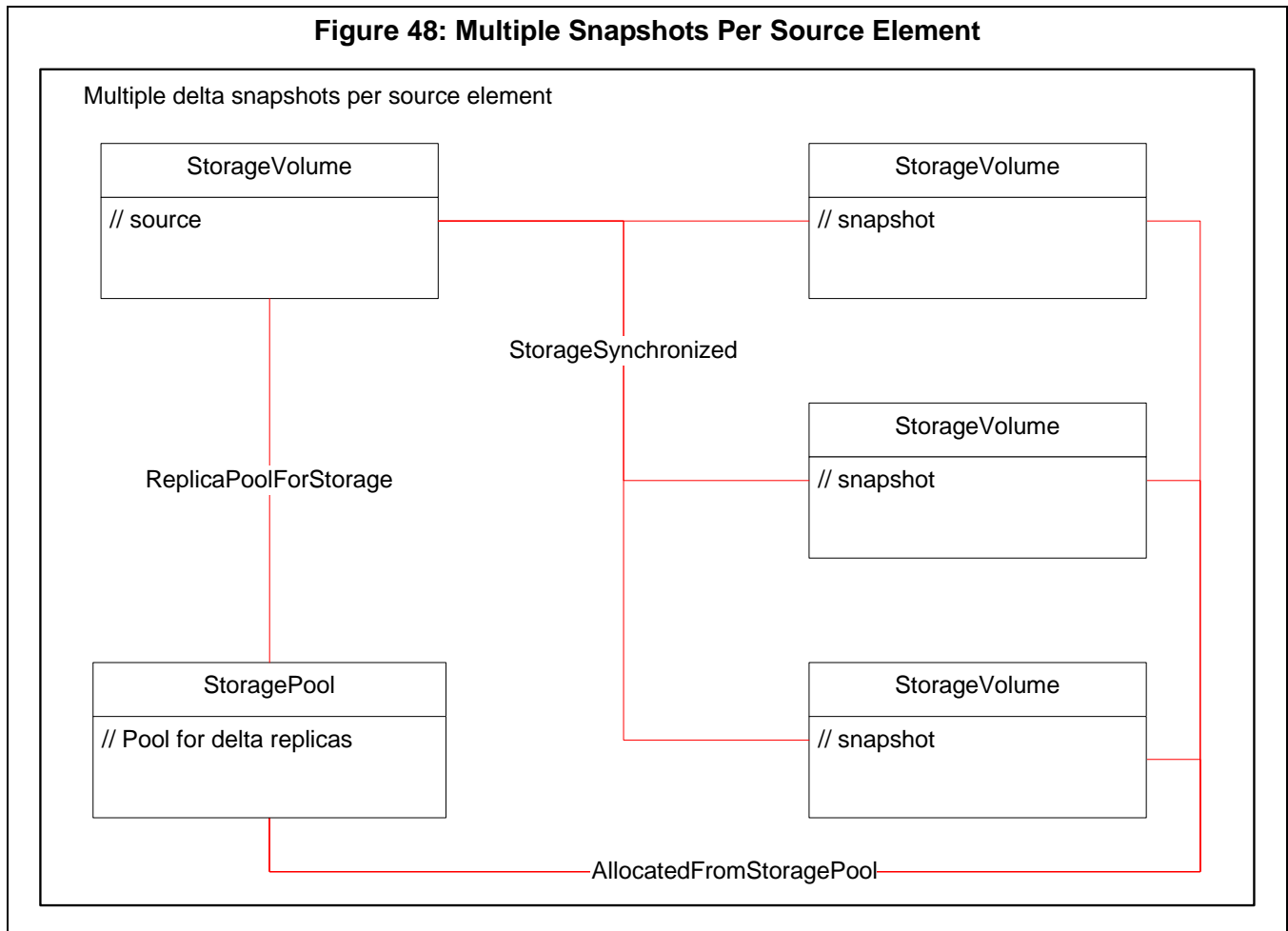


If remote replication depth exceeds two levels, the mid-level peer systems contain both remote source and remote target elements as shown in Figure 47:



Snapshots are created using CopyType “UnSyncAssoc” when either the CreateReplica or AttachOrModifyReplica extrinsic method is invoked. Snapshots may be created as full replicas or delta replicas. A provider supporting delta replicas may enable several optional capabilities used with the variable space consumption model described in the client considerations section. A client uses these capabilities to ensure sufficient but not excessive availability of space for groups of delta replicas. Action can be taken by a client to prevent failure of delta replica elements caused by lack of consumable space.

Figure 48 shows the basic model of snapshots created as delta replicas.



9.1.3 Durable Names and Correlatable IDs of the Profile

Durable names and Correlatable IDs are used to manage remote replication.

The Name property of the ComputerSystem instance representing a peer system is used to correlate two peers when a peer-to-peer connection is established. When a leaf instance of ComputerSystem is created by a cascaded provider, the value of the Name property is copied from the instance of the real ComputerSystem element.

ProtocolEndpoint elements eligible for use in peer-to-peer connections should have a Name value equivalent to the PermanentAddress value of the associated port. This is typically a port WWN for a FC port and is considered durable. These elements have a SystemName value equal to the Name value of the hosting system element.

The Name property of the storage element is used to correlate a leaf element on one peer with a local, realized element on the other peer if each peer has a separate SMI-S server/CIMOM. The Name value from a local, realized element on one peer is assigned as the Name value of the leaf element on the other peer.

9.1.4 Accessibility to Created Elements

The subprofile recommends that method providers for replica creation methods make all replica elements and associations accessible when the method response is returned to the client. This includes the case when the provider returns “job started” to the client. This allows the client to immediately monitor and manage the replica, new associations to the replica and new associated elements.

If the provider returns “job completed”, all new elements and associations shall be accessible. If “job started” is returned, new elements may not be immediately accessible. There are two cases the provider should consider:

Case 1: a new element and new associations are created (CreateReplica, CreateReplicationBuffer).

If the provider returns a reference to the new element as a method output parameter, all new associations shall also be accessible and AffectedJobElement shall now reference the new element for the returned job reference. No instance creation indications need to be generated. If the provider does not return a reference to the new element, an instance creation indication shall be generated when the new element is accessible. When the job completes successfully, AffectedJobElement shall reference the new element. The new element and all new associations shall be accessible when the instance creation indication is generated or the job completes successfully, whichever occurs first. Instance creation indications are not generated for new associations.

Case 2: a new association is created for an existing element (AttachOrModifyReplica, AttachReplica).

If the provider returns “job started”, AffectedJobElement already references the existing element and the client may attempt to access the new StorageSynchronized association. If the new association is not accessible, an instance creation indication for StorageSynchronized shall be generated when the association is accessible. The new association shall be accessible when the instance creation indication is generated or the job completes successfully, whichever occurs first.

For both cases, at the time an element or association is accessible to the client, all manageable element and association properties have valid values.

9.1.5 Completion of Long Operations

The subprofile supports three ways of indicating the completion of long running operations when a replica element is created or modified. This does not apply to a detach operation.

Option 1:

- 1) Provider returns “job completed” status.
- 2) SyncState value set to “... In Progress”.
- 3) Instance modification or instance deletion indication when SyncState value changes to final, steady state.

Option 2:

- 1) Provider returns “job started” status and REF to replica element.
- 2) SyncState value set to “... In Progress”.
- 3) Instance modification or instance deletion indication when SyncState value changes to final, steady state.
- 4) Instance modification when ConcreteJob ends.

Option 3:

- 1) Provider returns “job started” status but no REF to replica element.
- 2) Instance creation indication for StorageSynchronized when element is available. May indicate “... In Progress” state or final state.
- 3) Instance modification or instance deletion indication when SyncState value changes to final, steady state.
- 4) Instance modification when ConcreteJob ends.

Options 2 and 3 based on job control allow a provider to indicate “percent complete” for long operations and report job failure information with an instance of Error.

Any option may be selected for un-associated replicas if the provider creates a temporary instance of StorageSynchronized that is implicitly deleted when the replica is finished. If a temporary instance is not created, then only options 2 and 3 may be selected and steps 2 and 3 are bypassed.

The ModifySynchronization detach operation and the ReturnToStoragePool method cause element and association deletion. There are two ways to indicate completion of long delete operations.

Option 1:

Provider returns “job completed”. All affected elements and associations are no longer accessible. No instance deletion indications should be generated.

Option 2:

- 1) Provider returns “job started” status. Client assumes elements and associations are no longer accessible.
- 2) An instance deletion indication is generated for StorageSynchronized for a detach operation or for a replica element for a ReturnToStoragePool invocation. The element is successfully deleted when either job completion occurs or the instance deletion indication is generated, whichever occurs first.

9.1.6 State Management For Associated Replicas

Both mirror and snapshot replicas maintain stateful associations with source elements. The SyncState property of a StorageSynchronized association identifies the state. All providers shall support the ModifySynchronization extrinsic method that allows a client to manage the synchronization state of an associated replica unless a provider only allows unassociated replicas. All of the modify operations supported by the subprofile are classified as mandatory, optional or not supported by type of replica. Mirror replicas are the only type of replica created for CopyType values “Sync” and “Async”. Snapshot replicas are the only type of replica created for CopyType value “UnSyncAssoc”. Table 133 shows the classification.

Table 133: Synchronization Operation Support Requirements

ModifySynchronization Operation	Mirror Replicas	Snapshot Replicas
Detach	Mandatory	Optional
Resync	Mandatory	Mandatory
Fracture	Mandatory	Not supported
Quiesce	Optional	Optional
Unquiesce	Optional	Not supported
Prepare	Optional	Optional
Unprepare	Optional	Optional
Restore	Optional	Optional
Start Copy	Not supported	Optional
Stop Copy	Not Supported	Optional
Reset To Sync	Optional	Not supported
Reset To Async	Optional	Not supported

All instances of `StorageReplicationCapabilities` shall indicate all mandatory operations plus all supported optional operations in the value list assigned to the `SupportedModifyOperations[]` property. Undeployed optional operations should be implemented as a stubbed “no operation” to ensure backward compatibility with earlier versions of the subprofile. Modify operations perform the following actions:

Resync: Causes a fractured mirror replica to change from a point-in-time (PIT) view to a synchronized mirror replica representing the current view of the source element. The provider can execute a full or incremental copy as needed to realize a synchronized state. Causes a snapshot to be restarted as a new PIT image with a new value assigned to `WhenSynced`. May release all space previously consumed by the snapshot.

Fracture: Splits a synchronized mirror replica from its source element, changing the replica from a current view of the source element to a PIT view.

Restore: Copies a fractured mirror or a snapshot to the source element. At the completion of the restore operation, the source and replica represent the same PIT view. The Restore operation for each supported `CopyType` can be implemented as an incremental restore or a full restore based on the capabilities of the provider.

Detach: Removes the association between the source and replica elements. The `StorageSynchronized` association is deleted. If the replica is still a valid PIT image, the provider sets `OperationalStatus` to “OK”. If not a valid image but the storage element can be reused, the provider sets `OperationalStatus` to “Error”. A Detach operation does not delete the replica element. A client should invoke `ReturnToStoragePool` if the element is to be deleted following the Detach operation.

Start Copy: Starts a background copy operation for a snapshot replica. At the completion of the copy operation, the snapshot enters “Frozen” state.

Stop Copy: Stops a background copy operation for a snapshot replica. The snapshot state changes from “Copy In Progress” to “Idle”.

Quiesce/Unquiesce: This operation has optional, vendor-specific behavior for mirror replicas that is opaque to clients. The Quiesce operation stops the copy engine for snapshots and the snapshot no longer consumes space. A snapshot is no longer a valid PIT image if the source element is updated after the snapshot enters “Quiesced” state.

Prepare/Unprepare: This operation has optional, vendor-specific behavior for all replica types that may also depend on the entry state. A prepare operation typically starts a copy engine if entered from “Initialized” state.

Reset To Sync: Changes the `CopyType` value of a mirror replica from “Async” to “Sync”.

Reset To Async: Changes the `CopyType` value of a mirror replica from “Sync” to “Async”.

This information is summarized in Table 134, “SyncState Values”.

Table 134: SyncState Values

Synchronization State (<code>SyncState</code> value)	Mirror Replicas	Snapshot Replicas	Required ModifySynchronization Operations For Optional States
Initialized	Optional	Optional	Prepare
Prepare In Progress	Optional	Optional	
Prepared	Optional	Optional	Unprepare
Resync In Progress	Mandatory	Mandatory	
Synchronized	Mandatory	Not specified	
Idle	Not specified	Mandatory	

Table 134: SyncState Values (Continued)

Quiesce In Progress	Optional	Optional	Quiesce
Quiesced	Optional	Optional	Quiesce
Fracture In Progress	Mandatory	Not specified	
Fractured	Mandatory	Not specified	
Copy In Progress	Not specified	Optional	Start Copy
Frozen	Not specified	Mandatory	
Restore In Progress	Optional	Optional	Restore
Broken	Optional	Optional	

All providers shall have access to a time service that allows the provider to assign a date/time value to the WhenSynced property of StorageSynchronized at the time a replica becomes a valid PIT view of its source element. The WhenSynced value for mirror replicas shall be non-null for the “Fractured” and “Restore In Progress” synchronization states. The WhenSynced value for snapshot replicas shall be non-null for any synchronization state allowing host access to the replica.

A provider shall enforce state transition rules for associated replicas. If a client initiates a ModifySynchronization operation that causes a state transition violation, the provider returns an error response of “Invalid State Transition”. The provider shall allow a client to bypass certain transitions related to operations not supported by the provider. For example, a snapshot transition from “Idle” to “Resync In Progress” is allowed if the provider does not support Quiesce and Prepare operations.

Synchronization states have the following behavior:

Initialized: A source element and replica element are associated and all implicitly created associations are accessible. The copy engine has not started.

Synchronized: A mirror replica is fully copied and represents the current view of the source element.

Idle: A snapshot is accessible but not copied and represents a PIT view of the source element. A copy engine is actively executing copy-on-write operations.

Fractured: A mirror element is split from its source element and is now a PIT view.

Frozen: A snapshot is accessible and fully copied and represents a PIT view of the source element. The copy engine is stopped.

Broken: A replica is not a valid view of the source element and OperationalStatus of the replica element may have a value of “Error” if a repair action is necessary. The provider may allow access to a replica in this state if indicated in HostAccessibleState[] of StorageReplicationCapabilities. The subprofile currently does not specify how to recover from “Broken” state. A ModifySynchronization Detach operation may be invoked to a replica in this state.

Values of the SyncMaintained and WhenSynced properties in a StorageSynchronized association are maintained as shown in the following table. The table does not apply to CopyType “UnSyncUnAssoc”.

Table 135: SyncMaintained and WhenSynced Properties

Synchronization State	SyncMaintained		WhenSynced	
	Sync/Async	UnSyncAssoc	Sync/Async	UnSyncAssoc
Initialized	True or False	True or False	Null	Date/Time frozen

Table 135: SyncMaintained and WhenSynced Properties

Prepare In Progress	True or False	True or False	Null	Date/Time frozen
Prepared	True or False	True or False	Null	Date/Time frozen
Resync In Progress	True or False	True or False	Null	Date/Time frozen
Synchronized	True	Not specified	Null or D/T copy done	Null
Idle	Not specified	True or False	Null	Date/Time frozen
Quiesce In Progress	True or False	False	Null or D/T copy done	Null
Quiesced	True or False	False	Null or D/T copy done	Null
Fracture In Progress	True or False	Not specified	Null or D/T copy done	Null
Fractured	False	Not specified	Date/Time frozen	Null
Copy In Progress	Not specified	True or False	Null	Date/Time frozen
Frozen	Not specified	False	Null	Date/Time frozen
Restore In Progress	False	False	Date/Time frozen	Date/Time frozen
Broken	False	False	Null	Null

SyncMaintained “True” means that a copy engine is actively copying updated blocks from the source element to the target element. “False” means either the copy engine is stopped or copying the target to the source during “Restore In Progress” state. WhenSynced can contain two forms of a Date/Time value. A non-null value indicates either the date/time a frozen image is created or the date/time that the source element is completely copied to the target mirror element. The Fracture, Resync and Restore operations for ModifySynchronization may cause the WhenSynced value to change.

Figure 49 shows state transitions for mirrors and clones.

Figure 49: State Transitions for Mirrors and Clones

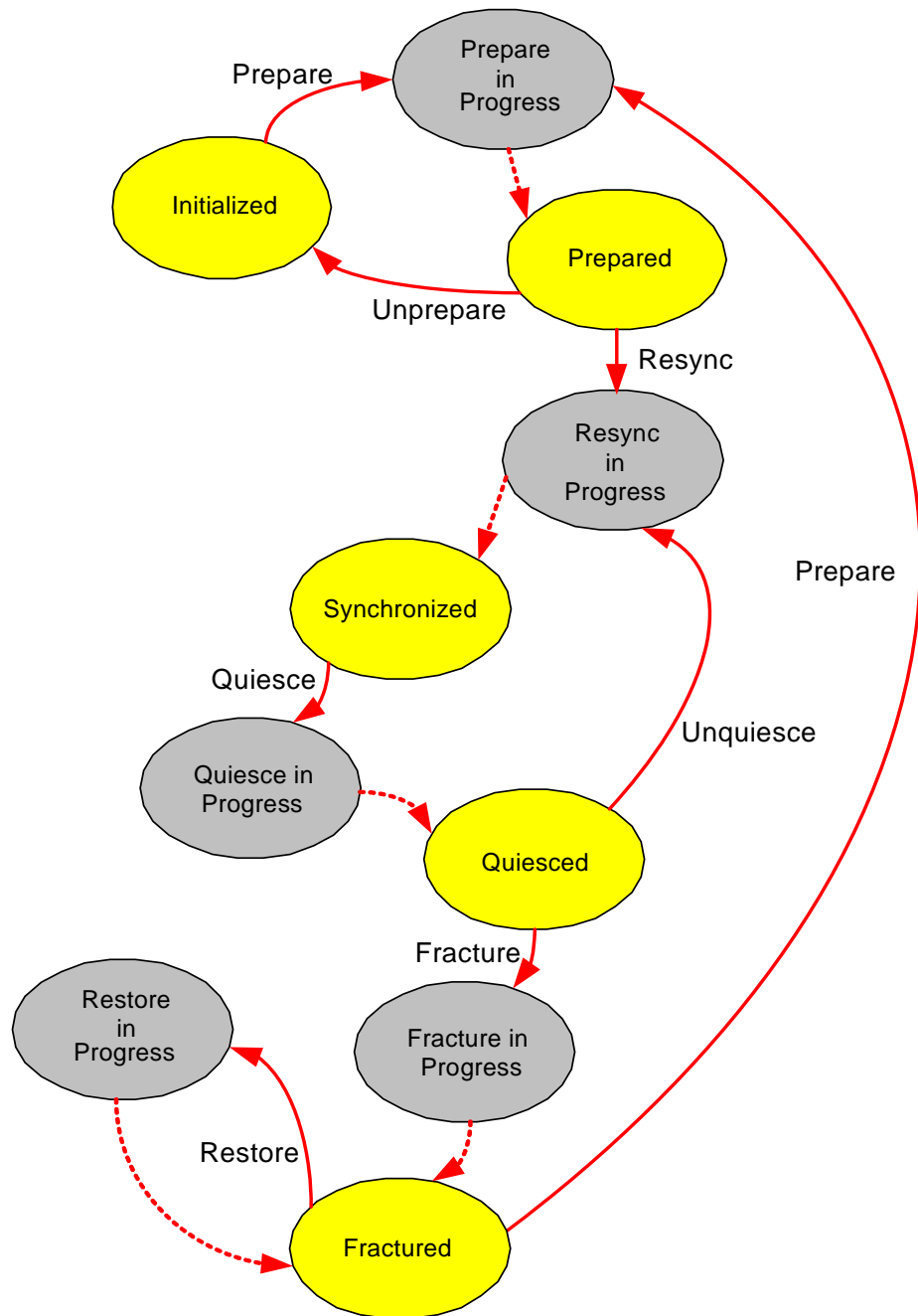
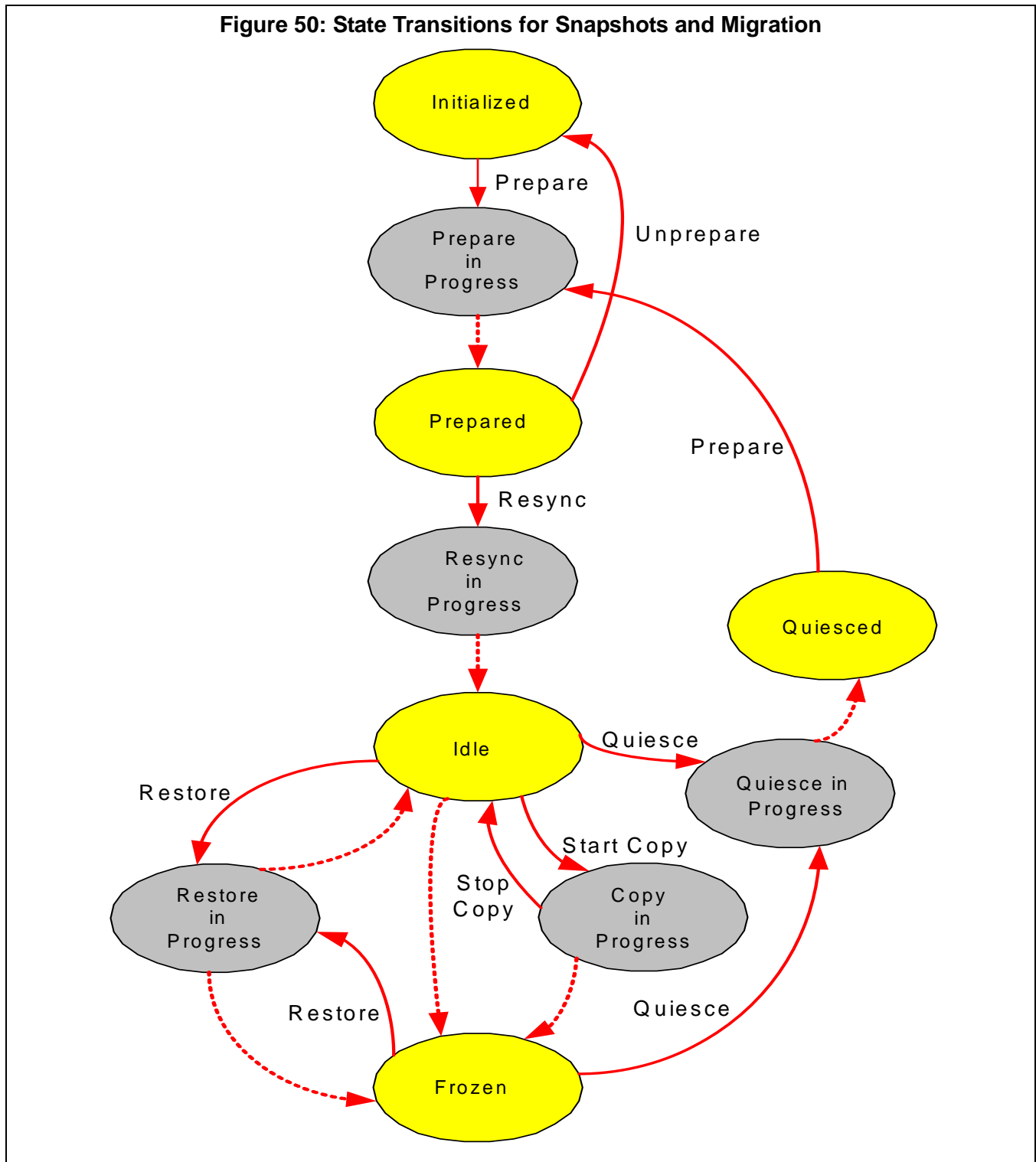


Figure 50 shows state transitions for snapshots and migration.



The preceding state diagrams for mirrors and snapshots use the following conventions:

- The state diagram is entered when any of the three replica creation methods is invoked. Exit occurs when a ModifySynchronization Detach operation is invoked.

- A transition from a steady state to an in progress state is shown by a solid arrow line and is initiated by a ModifySynchronization operation other than Detach.
- An automatic transition from an in progress state to a steady state is shown by a dashed arrow line.
- Automatic exit occurs from an in progress state when cloning and migration operations have completed.

9.1.7 Host Access Restrictions

The Copy Services Subprofile does not provide any services for managing access to replicas. However, replication services often restrict access to replicas for the following reasons:

- 1) Replicas have the same volume signature as their source element. Exposing both the source and replica to the same host may cause problems with a duplicate volume signature.
- 2) Delta replicas created by embedded software elements such as a volume manager may be unavailable for export to a secondary host.

The subprofile uses two properties in StorageReplicationCapabilities to indicate host access restrictions:

- 1) ReplicaHostAccessibility
- 2) HostAccessibleState[]

A provider may set values for these two properties indicating any host access restrictions imposed on replicas. These restrictions apply to all replicas created with the same CopyType value. Access control for a specific replica by a specific host is normally managed using services described in the Masking and Mapping subprofile.

EXPERIMENTAL

9.1.8 Settings, Specialized Elements and Pools for Replicas

A copy services provider shall support StorageSetting with the additional properties defined to manage replica elements and replication operations. These properties are listed in the definition of StorageSetting in this subprofile. This definition extends the basic list of required StorageSetting properties listed in the Block Services Package. The CreateSetting method should return a REF to a StorageSetting instance with all of the replication properties initialized to values consistent with the capabilities indicated in StorageReplicationCapabilities. Many replication properties allow an initial value of “not applicable” if the provider does not use the property. The provider sets the value lists for the SupportedStorageElementUsage[] and SupportedStoragePoolUsage[] properties in StorageConfigurationCapabilities to indicate which values of StorageSetting.StorageExtentInitialUsage and StorageSetting.StoragePoolInitialUsage are supported by the provider.

A provider may require specialized pools to contain delta replicas, specialized elements as replica targets and specialized elements as concrete components for delta replica pools. The provider may require the client to manage creation of these specialized elements – this is explained in detail in the client considerations section. Alternatively, the provider may automatically create specialized pools and elements and make them available for discovery by clients. In either case, the StorageExtentInitialUsage and StoragePoolInitialUsage properties in StorageSetting shall be supported by the provider as part of the goal parameter for pool/element creation methods.

Elements and pools specialized for Copy Services are located using the GetElementsBasedOnUsage method described in the Block Services Package.

When StorageExtentInitialUsage or StoragePoolInitialUsage is set in the goal parameter for an element or pool creation method, the value acts as an additional parameter indicating a specialized element. The provider ensures that the required element type is created and the Usage property value is set in the new replica element or pool. Certain types of specialized replica elements can be provided by changing existing elements using the RequestUsageChange method. The ClientSettableElementUsage[] value list indicates the allowable modifications

for a storage element and the `ClientSettablePoolUsage[]` value list indicates the allowable modifications for a storage pool.

9.1.9 Provider Configurations for Remote Replication

Remote replication always involves two peer system instances that may be managed through an established peer-to-peer connection. Provider developers can implement either one provider or two provider configurations for controlling remote replication service access points. The remote replication model allows connections that are bi-directional or uni-directional.

Configuration 1: one provider instance controls both peers. A client interfaces to one SMI-S server and CIMOM. The only stitching required between arrays uses a `StorageSynchronized` association between storage elements in separate arrays.

Configuration 2: A separate provider instance controls each peer system. Each provider has its own SMI-S server/CIMOM instance. The provider shall deploy stitching and cascading mechanisms based on the use of leaf elements.

The two-provider configuration can be used with either proxy or embedded providers. The two-provider configuration may provide higher availability in a cluster environment supporting failover and fallback.

The client managing a remote replication service shall discover both the source system and target system instances. The instance of `StorageReplicationCapabilities` for `SupportedSynchronizationType` values "Sync-Remote" and "Async-Remote" are probed to determine the service access point for remote replication. These are SMI-S server access points for method invocation of `StorageConfigurationService` methods. Two properties indicate the service access point(s):

RemoteReplicationServicePointAccess indicates the primary access point:

- Source: client interfaces to provider hosting the source storage elements.
- Target: client interfaces to provider hosting the target storage elements.
- Proxy: client interfaces to a cascading proxy provider for all operations [reserved for future use].

AlternateReplicationServicePointAccess indicates an alternate access point that may be used if the primary server fails or is not responding:

- None: no alternate access point is available.
- Source, Target, Proxy: same definition as above

Each provider instance exposes all peer-to-peer connection elements and all replica pair elements needed for client management through multiple service access points.

EXPERIMENTAL

9.1.10 Backward Compatibility

A 1.2 copy services provider can maintain backward compatibility with a 1.0 copy services client. The following conditions are necessary for backward compatibility:

- 1) The instance of `StorageConfigurationCapabilities` should set replication capability property values in the same way indicated for a 1.0 copy services provider. A 1.2 copy services client should ignore these properties and use `StorageReplicationCapabilities` instead.
- 2) The provider should treat `AttachReplica` as an alias for `AttachOrModifyReplica`.

- 3) The provider should treat `StorageSynchronized.SyncState` values “Synchronized” and “Idle” as equivalent for `CopyType` “UnSyncAssoc”.

9.1.11 Mutually Exclusive Capabilities

Both `StorageReplicationCapabilities` and `StorageConfigurationCapabilities` contain the `SupportedSynchronousActions[]` and `SupportedAsynchronousActions[]` properties. The provider shall not include the value corresponding to an action in both properties. An action can run synchronously or asynchronously but not both. An action indicated in one of the `StorageConfigurationCapabilities` properties shall also be indicated in a corresponding instance of `StorageReplicationCapabilities`.

9.2 Health and Fault Management Considerations

Certain capabilities of the subprofile use alert, instance modification and instance deletion indications for health and fault management. In general, instance modification indications when the `OperationalStatus` values of a replica element or a peer connection element change may indicate a fault. Instance modification indications when `StorageSynchronized.SyncState` automatically changes from any other value to “Broken” indicates a fault. If delta replica pools are supported with warning thresholds, alert indications may be generated by the provider when remaining space in a pool falls below a warning threshold or is completely consumed. The information in the alert indications is described in Table 137, “Copy Services Alert Indications”.

If remote replication is supported using managed connections, the provider shall generate instance modifications when `OperationalStatus` values change for a top-level `ReplicationPipe` identifying a peer-to-peer connection. A client may locate all replica pairs associated with the faulty connection by traversing `ConcreteDependency` associations from the pipe to the target elements of the pairs. The values listed in `OperationalStatus` Values for `ReplicationPipe` (Table 136:) are supported for `ReplicationPipe.OperationalStatus[]`:

Table 136: OperationalStatus Values for ReplicationPipe

OperationalStatus	Description
2: OK	Peer-to-peer connection is fully operational
3: Degraded	One or more connection paths is unavailable
6: Error	Connection has failed and copy operations cannot be completed
10: Stopped	Connection suspended by the provider
16: Supporting Entity in Error	One or more elements associated with the connection has a fault

When `OperationalStatus` indicates “Supporting Entity in Error”, a client should search for instances of `RelatedElementCausingError` associations to a `ReplicationPipe` as the dependent element. These associations identify endpoints for connections paths with a fault.

EXPERIMENTAL

The Copy Services Subprofile generates alert indications that allow monitoring of dynamic space consumption by delta replica elements. All of the alert indications indicate an `AlertType` value of “Device Alert” and an `OwnerEntity` value of “SNIA”. Alerts are generated for `CIM_StoragePool` elements to indicate that remaining consumable space is below a warning threshold percentage of total space or that all space in the pool has been consumed. The

LowSpaceWarningThreshold, TotalManagedSpace and RemainingManagedSpace properties can be analyzed to determine an appropriate response.

Table 137: Copy Services Alert Indications

AlertingManagedElement	PerceivedSeverity	ProbableCause	ProbableCauseDescription
Storage pool	Minor (4)	Threshold Crossed (52)	Pool at low space warning threshold: RemainingManagedSpace/ TotalManagedSpace
Storage pool	Major (5)	Out of Memory (33)	No remaining space in storage pool

The Copy Services Subprofile returns the error responses listed in Table 138, “Copy Services Error Responses” for the extrinsic methods supported by the subprofile. The subprofile uses MessageID values defined in the common error registry and the storage error registry.

Table 138: Copy Services Error Responses

MessageID	Message Name
MP2	Operation Not Supported
MP3	Property Not Found
MP5	Parameter Error
MP11	Too Busy To Respond
MP17	Invalid Property Combination During Instance Modification
DRM20	Invalid Extent Passed
DRM24	Invalid State Transition
DRM25	Invalid SAP For Method
DRM26	Resource Not Available
DRM27	Resource Limit Exceeded

9.3 Cascading Considerations

The Copy Services Subprofile can be deployed as a cascading subprofile and a leaf subprofile when a provider supports remote replication. Any remote replication provider that supports the cascading role shall also support the leaf role.

EXPERIMENTAL

9.4 Supported Subprofiles and Packages

The Block Services Subprofile is a mandatory prerequisite for the Copy Services Subprofile. Clients require methods and recipes from block services for the following purposes:

- Identify replica target candidates

- Identify extents and pools to be used as replica containers
- Create and delete replica container elements
- Create and delete replica target elements
- Create generated setting objects with additional properties required by the copy services subprofile.

Many classes and methods defined in Block Services are used in Copy Services without extensions or additional properties. In this case, the classes and methods are not redefined in Copy Services.

The Job Control Subprofile is required if any of the copy services extrinsic methods run asynchronously with created job elements.

The Cascading subprofile is required when remote replication supports cascading.

Copy services defines instance indications and alert indications using required and optional properties described in the Indications Subprofile.

9.5 Methods of the Profile

The Copy Services Subprofile is dependent on many of the extrinsic methods provided by block services. The subprofile also requires the provider to support the CreateInstance, GetInstance, ModifyInstance and DeleteInstance intrinsic methods for certain optional capabilities of the subprofile. The ReturnToStoragePool extrinsic method defined by block services is used to delete a replica element. ReturnToStoragePool may receive an MP3 (property not found) error response for replica elements that are implicitly deleted by a ModifySynchronization Detach operation.

All of the subprofile methods return one of three status codes or return an error response. The supported status codes are:

- 0: Job completed with no error
- 1: Method not supported
- 0x1000: Job started

Table 139 summarizes the extrinsic methods for replica creation and management.

Table 139: Extrinsic Methods of ReplicationServices Subprofile

Method	Described in
ModifySynchronization()	Table 140, "ModifySynchronization"
CreateReplica()	Table 141, "CreateReplica Method"
AttachOrModifyReplica()	Table 143, "AttachOrModifyReplica Method"
CreateReplicationBuffer()	Table 144, "CreateReplicationBuffer Method"
CreateOrModifyReplicationPipe()	Table 145, "CreateOrModifyReplicationPipe Method"

Table 140: ModifySynchronization

Method: ModifySynchronization			
Errors: DRM24, MP2, DRM25			
Parameters:			
Qualifiers	Name	Type	Description/Values

Table 140: ModifySynchronization

IN, REQ	Operation	uint16	Type of operation to modify the replica: 2: Detach 3: Fracture 4: Resync 5: Restore 6: Prepare 7: Unprepare 8: Quiesce 9: Unquiesce 10: Reset to Sync 11: Reset to Async 12: Start Copy 13: Stop Copy
OUT	Job	ConcreteJob REF	Returned if job started.
IN, REQ	Synchronization	StorageSynchronized REF	Association to replica that is modified

“Detach” operation deletes the StorageSynchronized association. An instance deletion indication is generated for this operation.

All ModifySynchronization operations are described in 9.1.4 Accessibility to Created Elements. If “job completed” is returned and the replica association indicates an “... in progress” SyncState value, an instance modification indication should follow when the replica enters its final, expected state. If “job started” is returned, the replica association indicates an “... in progress” SyncState value. In this case, two instance modification indications may follow. One should indicate the final SyncState value of the replica association when the job completes with no error. The other should indicate job completion for the instance of ConcreteJob.

StorageReplicationCapabilities.SupportedModifyOperations[] allows a client to verify that a specific operation is supported by a provider.

Table 141: CreateReplica Method

Method: CreateReplica			
Errors: DRM26, DRM27, DRM25, MP5			
Parameters:			
Qualifiers	Name	Type	Description/Values
IN	ElementName	string	Client-assigned, friendly name
OUT	Job	ConcreteJob REF	
IN, REQ	SourceElement	LogicalElement REF	
OUT	TargetElement	LogicalElement REF	
IN	TargetSettingGoal	StorageSetting REF	
IN	TargetPool	StoragePool REF	

Table 141: CreateReplica Method (Continued)

IN, REQ	CopyType	uint16	Copy type created: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc 6: Migrate
---------	----------	--------	---

Method notes:

- Creates a storage element of the same type as the source element.
- Creates a StorageSynchronized association”.
- Creates a SystemDevice association.
- Creates an AllocatedFromStoragePool association.
- Creates a StorageSetting instance with an ElementSettingData association.
- May create a BasedOn association.
- May create a ReplicaPoolForStorage association.
- All CopyType values may be supported.

If TargetPool is not supplied by the client, the provider response is implementation specific. For all operations not using specialized delta replica pools, the behavior of the client follows these rules:

- 1) Provider may return MP5 message indicating that TargetPool is an invalid parameter. In this case, the client should select a pool and retry the operation.
- 2) The provider will select a pool and proceed with the operation.

If the TargetPool is supplied, the provider uses the requested pool except for the next special case. For CopyType “UnSyncAssoc” creating a delta replica and DeltaReplicaPoolAccess values of “Shared” or “Exclusive” are indicated by the provider, TargetPool should be managed by the client as shown in TargetPool Parameter for Delta Replicas (Table 142:)

Table 142: TargetPool Parameter for Delta Replicas

DeltaReplicaPoolAccessvalue	TargetPool supplied	TargetPool not supplied
Shared	Error with an MP5 message. The specialized pool pre-exists and is always supplied by the provider.	Always the correct client action. The provider locates the specialized pool.

Table 142: TargetPool Parameter for Delta Replicas

Exclusive	If the method invocation is creating the first delta replica for the specified source element, TargetPool is supplied by the client. The pool is used by the provider and a ReplicaPoolForStorage association is created as a side effect. If delta replicas already exist for the source element, an error with an MP5 message will be returned.	If the specified source element has a ReplicaPoolForStorage association, the provider uses this pool as the container for a new delta replica. If this association does not exist, an error with an MP5 message is returned.
-----------	---	--

If TargetSettingGoal is not supplied by the client, the provider generates a default StorageSetting element for the replica. If TargetSettingGoal is supplied by the client, the provider will return an MP5 error message if the goal is incompatible with the corresponding target pool. If "job started" is returned, a Target Element reference may or may not be returned by the provider. "Accessibility to Created Elements" explains when a reference to the new replica element is available to the client.

EXPERIMENTAL**Table 143: AttachOrModifyReplica Method**

Method: AttachOrModifyReplica			
Errors: DRM25, DRM26, DRM27, MP5, MP17			
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	ConcreteJob REF	
IN, REQ	SourceElement	ManagedElement REF	
IN, REQ	TargetElement	ManagedElement REF	
IN, REQ	CopyType	uint16	Copy type created: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc 6: Migrate
IN, EmlInst	Goal	string	Setting element as an embedded instance.
IN	ReplicationPipe	ReplicationPipe REF	

Method: AttachReplica			
Errors: DRM25, DRM26, DRM27, MP5, MP17			
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	ConcreteJob REF	
IN, REQ	SourceElement	ManagedElement REF	
IN, REQ	TargetElement	ManagedElement REF	
IN, REQ	CopyType	uint16	Copy type created: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc

Method notes:

Uses an existing, independent storage element selected as a local replica target.

Creates a StorageSynchronized association.

May create a leaf storage element corresponding to a real element on a remote peer.

May create a ConcreteDependency association for the target if managed connections are supported.

Only CopyType values “Sync” and “Async” may be indicated for remote replicas.

Either CreateReplica or AttachOrModifyReplica shall be provided if local replicas are supported. Both may be provided if required by the provider. AttachOrModifyReplica shall be provided if remote replicas are supported. Replica target elements are deleted using the ReturnToStoragePool method in block services. All associations and associated setting elements are automatically deleted at the same time the element is deleted.

TargetElement candidates cannot have an existing SyncedElement role to a StorageSynchronized association. The provider returns a DRM26 error message if the candidate is already in use as a replica target element. Source elements may generally be associated with multiple replica targets. The provider may return a DRM26 error in some cases if an element cannot serve as a replica source. The provider may return a DRM27 error if the client attempts to create replication targets exceeding the provider specified limits.

If the method returns “job completed”, the new StorageSynchronized association is accessible to the client. If the method returns “job started”, the association may not be accessible. In this case, an instance creation indication should be generated by the provider when the association is accessible.

If the provider supports replica modification, a Goal parameter may be passed by the client to change the value of modifiable setting properties. The provider may ignore properties not relevant to replication operations. The properties that may be supplied by the client include UseReplicationBuffer, InitialSynchronization and ReplicationPriority.

If the provider supports managed peer-to-peer connections for remote replication, the client shall supply the ReplicationPipe parameter to scope a remote replica pair within a connection.

AttachReplica is supported for backward compatibility with earlier versions of the copy services subprofile. Refer to the description of AttachOrModifyReplica that follows. AttachReplica is identical to AttachOrModifyReplica with the omission of the Goal and ReplicationPipe parameters.

The subprofile uses the following optional methods for managing peer-to-peer connections for remote replication:

Table 144: CreateReplicationBuffer Method

Method: CreateReplicationBuffer			
Errors: DRM20, DRM26, MP5			
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	ConcreteJob REF	
IN, REQ	Host	ManagedElement REF	Reference to either a host ComputerSystem or a top-level ReplicationPipe.
IN	TargetElement	StorageExtent REF	
IN	TargetPool	StoragePool REF	
OUT	ReplicaBuffer	Memory REF	

Method notes:

Creates an instance of Memory as the private buffer element.

Creates a AssociatedMemory association to either a hosting ComputerSystem or a replication ReplicationPipe. Hosting system may be top-level or tiered.

Creates a SystemDevice association.

Creates an AllocatedFromStoragePool association.

May create a BasedOn association.

The client may pass either a TargetElement parameter or TargetPool parameter but not both. If TargetElement is passed, the replication buffer is created and consumes the entire element. The provider either hides the element from the client or transforms the element to a StorageExtent with a BasedOn association to the replication buffer. If a TargetPool is passed, the buffer element is created in the pool and the size is determined by the provider. If neither parameter is passed, the provider determines the size and location of the buffer.

Error response DRM20 is returned if the client passes a pool or extent that is not eligible for use as replication buffer space. Error response DRM26 is returned if the client passes an extent that is too small. The DRM26 response data indicates the minimum size required for the passed extent.

The rules for “job completed” and “job started” follow the pattern described above for CreateReplica.

Table 145: CreateOrModifyReplicationPipe Method

Method: CreateOrModifyReplicationPipe			
Errors: DRM25, DRM27, MP5, MP11			
Parameters:			
Qualifiers	Name	Type	Description/Values
IN	PipeElementName	string	
IN, REQ	SourceSystem	ComputerSystem REF	
IN, REQ	TargetSystem	ComputerSystem REF	
IN	SourceEndpoint[]	ProtocolEndpoint REF	
IN	TargetEndpoint[]	ProtocolEndpoint REF	
IN	Modifier	uint16	Modifier operation: 2: Suspend 3: Resume 4: Enable Long Busy 5: Disable Long Busy 6: Force Creation 7: Force Removal
IN, OUT	ReplicationPipe	ReplicationPipe REF	

Method notes:

- Creates a ReplicationPipe composition with one top-level pipe and lower-level pipes for each supplied ProtocolEndpoint pair.
- Creates HostedNetworkPipe associations for all pipes and NetworkPipeComposition associations for all lower-level pipes.
- Creates EndpointOfNetworkPipe associations for all supplied ProtocolEndpoint elements.

- May create a leaf `ComputerSystem` instance with `SystemComponent` and `Dependency` associations. `SourceSystem` and `TargetSystem` may be top-level or tiered elements located by traversing a `ComponentCS` association.
- May create one or more leaf `ProtocolEndpoint` instances with same associations as real endpoint elements.

If the provider supports client assignment of selected ports to peer-to-peer connections, the client may pass `SourceEndpoint[]` and `TargetEndpoint[]` parameters with the same number of endpoint references in each parameter. If a new connection is created, a reference to the new top-level replication `ReplicationPipe` is returned to the client. If an existing connection is modified, `ReplicationPipe` is passed as an input parameter and the new endpoint lists replace the previous endpoint lists.

The `Modifier` property may be supplied along with an IN reference to a replication pipe. This causes the requested modify operation to be executed. `Force Creation` is the only modifier value allowed when an IN reference is not supplied.

EXPERIMENTAL

9.6 Client Considerations and Recipes

A single instance of a Copy Services provider may support mirrors, snapshots and clones. A client follows these steps to fully discover and understand all capabilities of the provider:

- Locate the hosted instance of `StorageConfigurationService`.
- Enumerate and get all of the informational capability objects associated with `StorageConfigurationService`

Block services shall be supported by the provider. The Copy Services Subprofile shall be registered by the provider. The provider shall host one instance of `StorageConfigurationService`.

The properties of `StorageConfigurationCapabilities` and `StorageReplicationCapabilities` indicate precisely how the provider supports each copy service feature. The client should find one instance of `StorageReplicationCapabilities` for each `SupportedSynchronizationType` value supported by the provider. `StorageReplicationCapabilities` can be specialized as shown in `Replica Specialization by CopyType` (Table 146):

Table 146: Replica Specialization by CopyType

SupportedSynchronizationType value	CopyType value	Specialization
Async-Local (2)	Async (2)	Asynchronous local mirror replication
Async-Remote (3)	Async (2)	Asynchronous remote mirror replication
Sync-Local (4)	Sync (3)	Synchronous local mirror replication
Sync-Remote (5)	Sync (3)	Synchronous remote mirror replication
UnSyncAssoc-Full (6)	UnSyncAssoc (4)	Full snapshots
UnSyncAssoc-Delta (7)	UnSyncAssoc (4)	Delta snapshots
UnSyncUnAssoc (8)	UnSyncUnAssoc (5)	Clone replication
Migration (9)	Migration (6)	Local volume migration

Each instance shows the client:

- Replica type supported (full or delta)
- Methods supported and ModifySynchronization operations supported
- Any restrictions on host access to replicas
- Upper limits such as maximum replicas for one source element
- Specialized features by CopyType

Most of the properties in `StorageReplicationCapabilities` are optional. The client first analyzes `SupportedSynchronousActions[]`, `SupportedAsynchronousActions[]`, `SupportedModifyOperations[]` and `SupportedSpecializedElements[]`. Support for the remaining optional properties is conditional on the values indicated for these properties.

If the provider supports remote replication, the client shall determine if peer-to-peer connections are used. If `PeerConnectionProtocol` has a non-null value, locate a `Network` element associated to the top-level `ComputerSystem` element. There shall be a `Network` element with a name of the form:

- `NameFormat` = "Other"
- `Name` = "RemoteReplicationNetwork.<value of PeerConnectionProtocol>"

This `Network` element is private and does not have a durable, correlatable name value. If a correctly named `Network` element is discovered, the provider supports peer-to-peer connections between any pair of arrays associated to the element. Two such elements with the same name in different SMI-S servers are assumed to be compatible if the provider supports cascading. If peer-to-peer connections are supported and `SupportedSynchronousActions[]` includes the value "Network Pipe Creation", the provider supports dynamic, managed connections.

EXPERIMENTAL

9.6.1 Managing Peer-to-peer Connections

Remote replication is supported if an instance of `StorageReplicationCapabilities` is discovered with a `SupportedSynchronizationType` value of either "Async-Remote" or "Sync-Remote". A client may need to establish a connection between two peer systems before remote replicas can be created and managed. The client selects two peer systems as source and target hosts. The two peers may be controlled by one SMI-S server or each may be controlled by a different SMI-S server.

Step 1: The client may verify the compatibility of two peer systems. Both peers shall be the same element type, e.g. both are arrays. Both peers shall represent the same vendor (`Product.Vendor` or `SoftwareIdentity.Manufacturer` – a requirement for SMI-S 1.1).

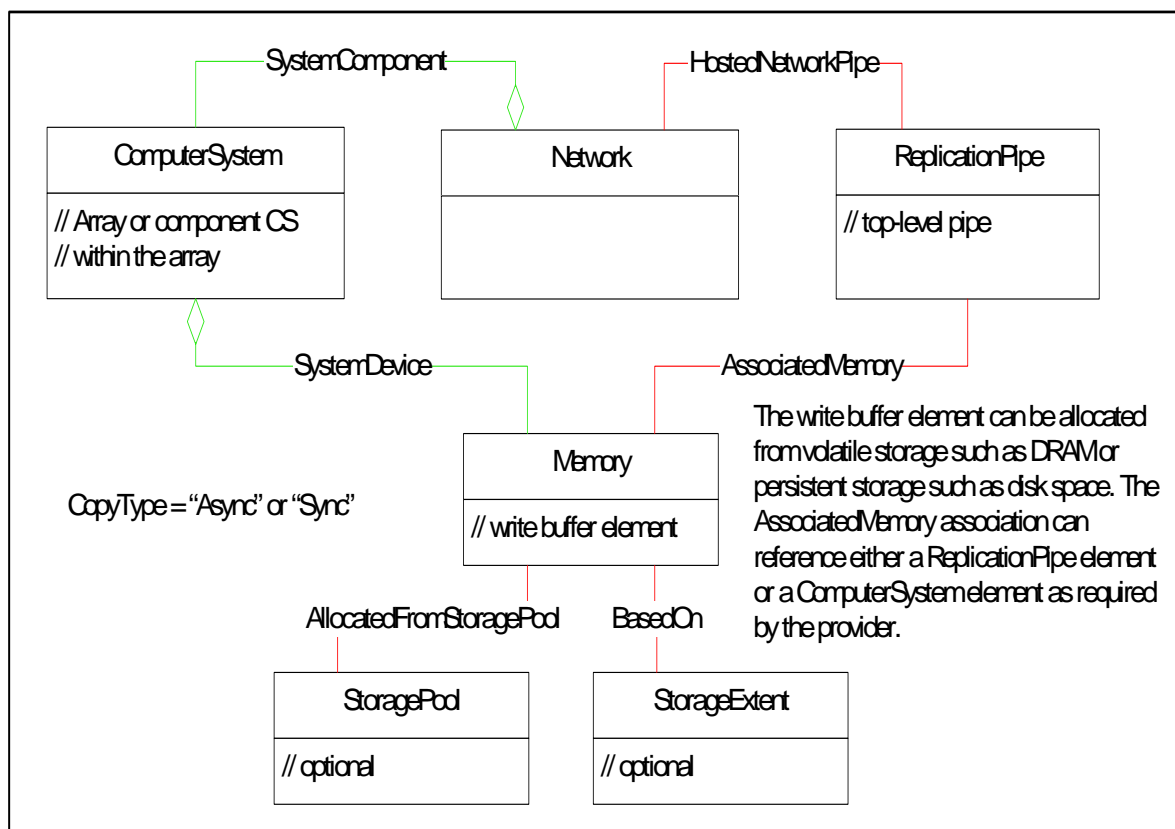
- There shall be a `SupportedSynchronizationType` value match: both support "Sync-Remote" or both support "Async-Remote".
- There shall be a `SupportedSynchronousActions[]` and `SupportedAsynchronousActions[]` value match. Both shall support the same set of replication operations. Use of Job Control shall match for all operations.
- If peer-to-peer connections are supported, both peers shall either be associated to the same `Network` instance or, if managed from different SMI-S servers, both shall be associated to identically named `Network` instances.
- If managed peer-to-peer connections are supported, there shall be a value match for the `BidirectionalConnectionsSupported` and `RemoteReplicationServicePointAccess` properties.

Step 2: If peer-to-peer connections are supported, search for an existing connection between two peer systems. Begin the search from the top-level system element providing the service access point as indicated by `RemoteReplicationServicePointAccess`. If a connection is not found, continue the search from any lower-level system element located with a `ComponentCS` association. The search traverses four levels of association: `ComputerSystem` --> `ProtocolEndpoint` --> `ReplicationPipe` --> `ProtocolEndpoint` --> `ComputerSystem`. If a match is found, follow the `NetworkPipeComposition` association from the lower-level pipe to the top-level pipe identifying the connection. A reference to this top-level pipe is subsequently used to create remote replicas within the context of this connection.

Step 3: If an existing connection is not found, a new connection is created by invoking the `CreateOrModifyReplicationPipe` method. Select a set of endpoint pairs as required. The provider may limit the maximum number of endpoint pairs per connection, maximum connections per peer system and the maximum number of connections handled by one endpoint. Limits are indicated by properties in an instance of `StorageReplicationCapabilities`. Select the same number of endpoints from each peer to form endpoint pairs. All of the endpoints eligible for assignment to peer connections have a `ProtocolIFType` value of "Other" and an `OtherTypeDescription` value equal to the value of `PeerConnectionProtocol`. Invoke the `CreateOrModifyReplicationPipe` method. A provider supports uni-directional connections if `BidirectionalConnectionsSupported` is "false". The `SourceSystem` and `TargetSystem` parameters shall reference the system elements that should host all of the source elements and all of the target elements respectively if uni-directional connections are supported. Otherwise, either parameter can reference either system element. The client may assign an element name value to the new `ReplicationPipe` instance. The selected endpoint pairs are passed as `SourceEndpoint[]` and `TargetEndpoint[]` corresponding to `SourceSystem` and `TargetSystem`. The client invokes the method to the service access point identified by `RemoteReplicationServicePointAccess`. The provider creates a `ReplicationPipe` composition with directionality properties set. There is one top-level pipe plus lower-level pipes corresponding to each `ProtocolEndpoint` pair assigned to the connection. The provider returns a reference to the top-level pipe. The model construction for the peer-to-peer connection is described in Figure 45.

Step 4: A write ahead buffer may be required by one or both peers as indicated by the `RemoteBufferSupported` property in `StorageReplicationCapabilities`. Four properties in `StorageReplicationCapabilities` indicate how to manage remote buffers:

- `RemoteBufferSupported` indicates if buffers are not supported, required or optional
- `RemoteBufferLocation` indicates if the buffer is hosted on the source system, target system or both.
- `RemoteBufferHost` indicates if a buffer is required for each system element, each component system element or each `ReplicationPipe` element.
- `RemoteBufferElementType` indicates if the client supplies a reference to a component element such as a storage volume, a storage pool or neither as the container element for the buffer.

Figure 51: Remote Replication Buffer

If the client supplies a component element, the client determines the necessary size of the buffer element. If the client supplies a target pool, the provider determines the size. If the client does not specify the target, the provider determines both the size and location. When the provider determines the location, the buffer can be realized in volatile DRAM or persistent disk space. The client invokes `CreateReplicationBuffer` to create the buffer element.

Buffer elements are deleted by invoking `DeleteInstance`

Once the connection is established, endpoint pairs may be attached to or detached from the connection as necessary using the `CreateOrModifyReplicationPipe` method. This allows a client to manage the amount of transport bandwidth assigned to each connection. A connection is removed by invoking `DeleteInstance` for a top-level `ReplicationPipe`. The entire `ReplicationPipe` composition is deleted along with corresponding associations. Deletion fails if any replica pairs remain associated with the connection.

EXPERIMENTAL

9.6.2 Using StorageSetting for Replicas

The `StorageSetting` class has several properties used to create and manage replicas. Instances of this class are used as goal parameters for many of the methods used by the subprofile. These instances are serially reusable for a short sequence of operations ending with creation of a pool or an element. The client should follow these steps:

- 1) Invoke `CreateSetting` with `SettingType` value "Goal" for a selected storage pool.

- 2) Set values for all of the properties used to create and manage replicas. These properties are listed in the definition of `StorageSetting` in this subprofile. Property values can be changed by the `ModifyInstance` intrinsic method. The `SupportedStorageElementUsage[]` and `SupportedStoragePoolUsage[]` properties in `StorageConfigurationCapabilities` indicates which values of `StorageExtentInitialUsage` and `StoragePoolInitialUsage` are supported. Other replication properties may have been returned to the client with an initial value of “not applicable”. The client should not modify the value of any property with a value of “not applicable”.
- 3) The generated setting may initially be used one or more times as a goal parameter for the `GetSupportedSizes` and `GetSupportedSizeRange` methods. The setting may then be used once as a goal parameter for a pool or element creation method.
- 4) When the client no longer needs the generated setting instance, invoke the `DeleteInstance` intrinsic method.

9.6.3 Finding and Creating Target Elements

If a provider supports the `AttachReplica` and/or the `AttachOrModifyReplica` methods, the client finds or creates target elements eligible to become replicas. A provider may restrict replica targets to a specialized set of elements if element usage restrictions are supported as indicated in `StorageConfigurationCapabilities`. The client should follow these steps:

- 1) Determine the required size of the target element. Use the size of the source element unless a delta replica is created. If a delta replica is created, the size may be smaller than the associated source element.
- 2) Create a goal setting instance. Set `StorageExtentInitialUsage` to the correct value for the type of specialized element needed by the client. Set other replication setting property values as desired. Refer to the “Creating and Managing Snapshots” section for guidelines on using delta reservation properties. Use this goal instance in all the remaining steps.
- 3) Search for existing `StorageVolume` instances that can be used as replica targets. A client can invoke the `GetElementsBasedOnUsage` method to locate available targets from existing elements. The client is responsible for screening the candidates for the required size and settings values. The search is always initiated on the system that will host the target element.
- 4) If no candidates exist, follow block services client considerations and recipes to create a new element as the replica target. Target elements may be created in pools or from element types that a provider supports as a component. As in step 2, set `StorageExtentInitialUsage` and all of the other replication setting properties to the required values before creating a new element. If a virtual element is created in a special delta replica pool (described in subsequent sections), the `Size` parameter value should be omitted when the element is created.

EXPERIMENTAL

9.6.4 Creating and Managing Pools for Delta Replicas

A provider may require specialized pools as containers for delta replicas. Such a pool only contains delta replicas based on the variable space consumption model explained below. The client should inspect the values of `StorageReplicationCapabilities.DeltaReplicaPoolAccess`. Values are:

- “Any” – Specialized pools not required for delta replicas. The provider creates delta replicas based on the fixed space consumption model and the client can select any pool as a container.
- “Shared” – a single shared pool is the container for all delta replicas. This type of pool is always preexisting and may be located with the `GetElementBasedOnUsage` method. The client may need to add space to this type of pool.
- “Exclusive” – each source element requires an exclusive, special pool for associated delta replicas. If the pool already exists, it is associated to the source element with a `ReplicaPoolForStorage` association. If the pool does not exist, the client creates the pool.

Delta replica pools are commonly created from or extended with component elements supplied by the `InExtents[]` parameter of the `CreateOrModifyStoragePool` method. The provider consumes all of the space in the supplied elements for this type of pool. All of the supplied elements should come from a single pool. Preexisting component elements may be located using the `GetElementsBasedOnUsage` method with the `Usage` parameter set to "Element Component". New component elements may be created using a goal parameter with `StorageExtentInitialUsage` set to "Element Component". The component element type shall be a type supported by the provider as indicated in `SupportedStorageElementTypes[]`.

A client may increase the size of a preexisting shared pool by adding component elements. A common practice would be to use multiple small elements of equal size. Selected component elements are passed to the `CreateOrModifyStoragePool` method using the `InExtents[]` parameter. The new elements are combined with any existing elements to increase the pool size.

A client may create new exclusive pools or increase the size of an existing exclusive pool. A new exclusive pool is commonly created by supplying one component element that supplies the required pool size. Later, the exclusive pool size is increased by supplying a `Size` parameter value indicating the required new size of the pool. The provider determines how to increase the size. An exclusive delta replica pool is automatically associated to a source element by the provider. A `ReplicaPoolForStorage` association to the source element is created during the first `CreateReplica` operation that refers to the pool.

If warning threshold alerts are supported, the client may invoke `ModifyInstance` to modify the value of `StoragePool.LowSpaceWarningThreshold`. The pool size can be increased following a low space alert indication.

If the provider requires a shared pool and only supports "Replica Attachment" as the method for creating delta snapshots, then the shared pool shall be provisioned with virtual devices to be used as target elements. The client should ensure that enough virtual devices exist to create the expected maximum number of delta replicas. Some number of virtual devices may preexist. If the client creates virtual devices, create a goal element for each virtual device with `StorageExtentInitialUsage` set to "Delta Replica Target" and omit the `Size` parameter when invoking the element creation method. This type of virtual device always has an initial `SpaceConsumed` value of zero and does not have a `StorageSynchronized` association until `AttachOrModifyReplica` is subsequently invoked by the client.

Capacity management for a delta replica pool adheres to the capacity relationship formula specified in Block Services, Extent Mapping and Extent Conservation. The standard capacity relationship is:

$$\text{TotalManagedSpace} = \text{RemainingManagedSpace} + \text{SUM}(\text{SpaceConsumed})$$

where `SpaceConsumed` is a sum for all elements created in the pool. `RemainingManagedSpace` and `SpaceConsumed` properties may have volatile values for a delta replica pool and the elements in the pool. The provider shall maintain values for these properties that satisfy the formula. However, a client may receive stale values when instance properties are retrieved in multiple operations. The stale values may result in an unequal comparison when the capacity management relationship is checked. A client should not expect to determine exactly how much space is consumed by a delta replica in a shared or exclusive pool. If a snapshot service provider allows multiple snapshots to share a consumed block, only one snapshot will count the block in its `SpaceConsumed` value. The most important capacity management role for the client is to correctly size the delta replica pool. The sizing should be based on the maximum number of snapshots retained in the pool and the expected space consumption per snapshot.

If the provider supports low space warning threshold alerts, the client should subscribe to these alert indications. The client should maintain adequate pool capacity by either increasing the pool size or deleting the oldest snapshots when an alert is received.

Extent mapping and extent conservation are not supported for elements created in a specialized delta replica pool.

EXPERIMENTAL

9.6.5 Creating and Managing Mirrors

A mirror replica is the same size as the associated source element and is fully copied from the source element. A provider may allow the mirror element to be a larger size than the source element. A full background copy is normally initiated by the provider when a mirror replica is created. If the provider defers the background copy, the client may need to initiate the copy at a later time.

A provider normally runs a copy engine that maintains a mirror as the current image of the associated source element. The copy engine may operate in either synchronous or asynchronous mode. If the client requests CopyType “Sync” when the replica is created, the copy engine runs in synchronous mode and any write I/O operation to the source does not receive ending status until the write operation is also completed for the mirror. If the client requests CopyType “Async”, the copy engine runs in asynchronous mode and write I/O operations receive ending status when the operation completes for the source element.

A mirror may be changed from a current image of the source element to a point-in-time image using a fracture operation. A mirror in the “Fractured” state is called a split mirror. A mirror can also be converted to an independent storage element by a “Detach” operation following a fracture operation. The detached mirror is equivalent to a clone element created with a CopyType “UnSyncUnAssoc” request (discussed below).

The subprofile supports both local mirrors and remote mirrors. A local mirror target element is hosted on the same system as the source element. A remote mirror target element is hosted on a different system than the source element and a remote mirror may require a managed connection between two peer systems. An operation to create a mirror includes the following steps:

Step 1: search the target host using the GetElementsBasedOnUsage method with the Usage parameter value set to “Local Replica Target” or “Remote Replica Target”. The client can search the entire host or selected pools on the host. The client interfaces to the host system for the source element if a local mirror is created. Otherwise, the client interfaces to the host system for the remote target element. The client shall provide a replica size value for the screening operation. Normally, this is the same size value as the source element. Select a candidate volume based on best fit or some other appropriate filter. Proceed to step 3 if a candidate is selected from existing elements.

Step 2: select a pool for creation of a new target element. For the pool being screened, access the associated StorageCapabilities instance and invoke CreateSetting to generate a modifiable setting object that is used as a goal parameter for one or more method invocations. Set StorageExtentInitialUsage to either “Local Replica Target” or “Remote Replica Target”. Invoke GetSupportedSizes or GetSupportedSizeRange and screen the pool based on the target element size. If the pool does not support the required size, proceed to the next candidate pool. If a candidate pool is found and CreateReplica will be used to create the new mirror, proceed to step 3. Otherwise, the client may follow operations described in the Block Services Package to create a new replica target candidate. Note: a client may elect to bypass screening and require a user to manually select a candidate pool or target element.

Step 3: invoke AttachOrModifyReplica or CreateReplica to create a new mirror replica. If the provider returns “job completed” status, the client can immediately access the StorageSynchronized association instance for the new replica. If the provider returns “job started” status, the client may need to wait for accessibility to the StorageSynchronized association as described in State Management For Associated Replicas (9.1.6). The client may need to initiate additional operations to bring the new replica to the required synchronization state. If the provider supports an InitialReplicationState of “Initialized”, the copy engine has not started a background copy operation and the client may invoke ModifySynchronization requesting a “Prepare” or “Resync” operation as needed.

Creation of remote mirrors requires special client consideration. The client inspects the StorageReplicationCapabilities.RemoteReplicationServicePointAccess to locate the service access point for invoking AttachOrModifyReplica. Refer to 9.1.9 Provider Configurations for Remote Replication for more information.

The `ModifySynchronization` method can be invoked to manage existing mirrors. The subprofile supports the following operations:

- 1) Mirrors can be split from their associated source element using a “Fracture” operation. A split mirror is a point-in-time image of the source element. The split mirror can be used as a source for a backup operation or can be treated as a temporary clone. A split mirror can be changed back to a current image of the source element using a “Resync” operation.
- 2) Mirrors can be converted to independent storage elements by a sequence of operations including “Fracture” and “Detach”.
- 3) The source element can be restored from a mirror by invoking a “Restore” operation. This should normally follow a client action that blocks host I/O to both the source element and all associated replica elements until the restore operation is completed.
- 4) A provider may support “ResetToSync” and “ResetToAsync” operations if availability and performance QoS policies change over time. Invoke “ResetToSync” when availability QoS changes to a higher priority than performance QoS. Invoke “ResetToAsync” when the reverse relationship occurs.

If `ModifySynchronization` is invoked for a remote replica association, follow the same rules described above to determine the service access point.

9.6.6 Creating a Clone and Redirected Restore Operations

A clone is a full size, fully copied local replica that becomes an independent storage element as soon as the background copy operation is completed. A clone is usually created by invoking the `AttachOrModifyReplica` or `CreateReplica` methods with the `CopyType` parameter set to a value of “UnSyncUnAssoc”. Alternatively, a clone may be created by detaching a split mirror or a frozen snapshot.

The provider shall automatically initiate a background copy operation when `CopyType` “UnSyncUnAssoc” is requested by a client. If the provider deploys the method as an asynchronous operation, then the provider may elect to create a temporary `StorageSynchronized` association that allows the client to manage copy priority for the background copy operation. This temporary association should only indicate a `SyncState` value of “Resync in progress” and the provider shall automatically delete the association when the background copy operation is completed. The client can modify the value of `CopyPriority` while the copy operation is in progress. The temporary association cannot be used for any other purpose and the client shall never invoke `ModifySynchronization` against this type of association.

A provider may allow a frozen snapshot to be treated as a clone. The client observes that a replica previously created with `CopyType` “UnSyncAssoc” has a `SyncState` value of “Frozen”. If the provider supports the `ModifySynchronization Start Copy` operation, this operation may be invoked to bring the replica from idle state to frozen state. The provider may allow copy priority to be managed as described in the next section.

The clone is a point-in-time image of the source element. The client shall supply any needed date/time value for the point-in-time because a guaranteed `WhenSynced` property value is not available for a clone created by a `CopyType` “UnSyncUnAssoc” operation. A provider may create a clone as either a synchronous or asynchronous operation. When the operation is completed, the client assumes the clone is ready to manage as an independent element if the `OperationalStatus` property indicates a value of “OK”.

The Restore operation for the `ModifySynchronization` method only allows restoration to the source element associated with a replica. If a provider supports multi-level replication, a variation of clone creation may be used to restore a replica to a redirected location. Invoke a replica creation method supported by the provider passing a replica element as the source parameter and also indicate `CopyType` “UnSyncUnAssoc”. The target may be a new element or an existing independent element.

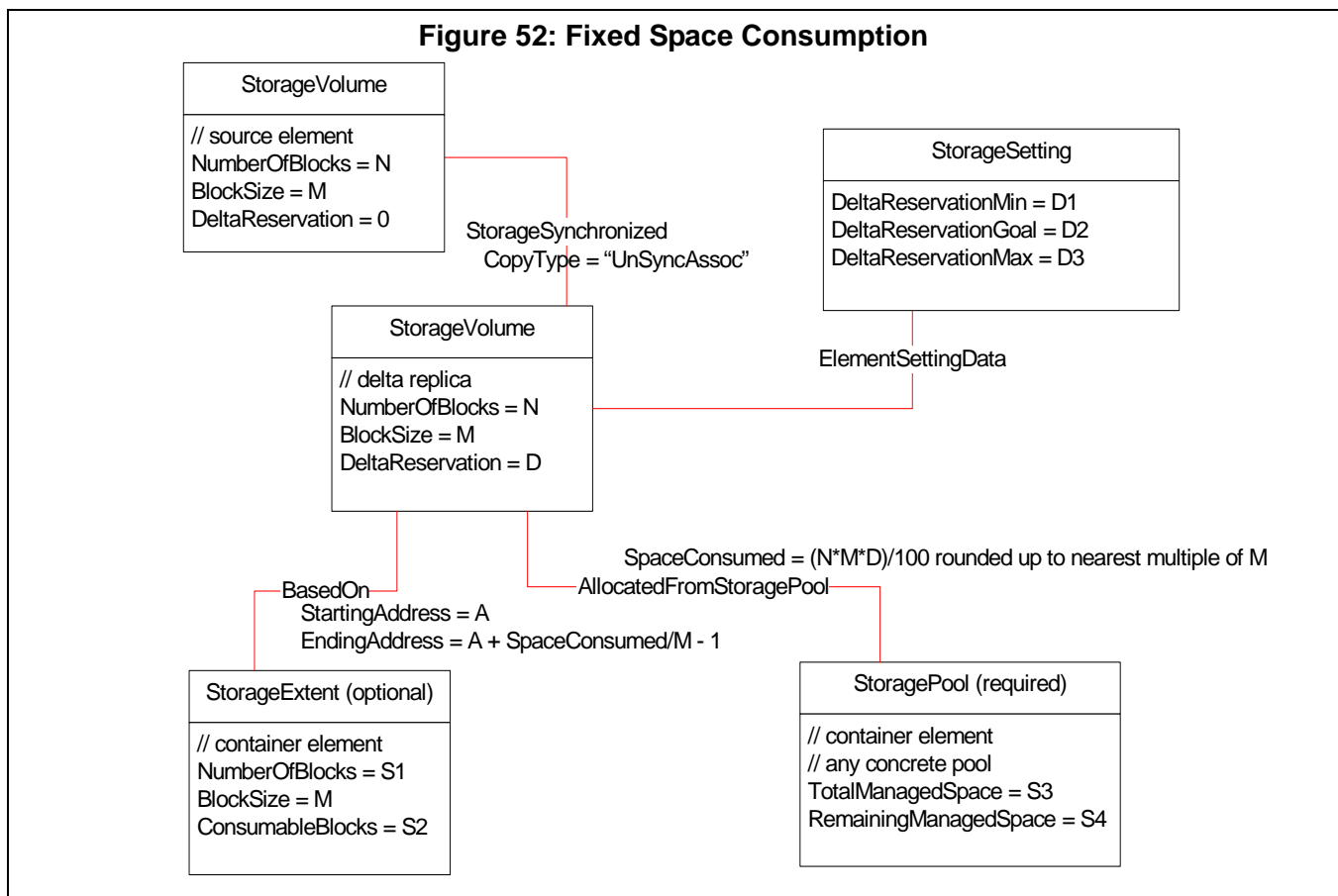
EXPERIMENTAL

9.6.7 Creating and Managing Snapshots

Snapshot replicas are point-in-time images created with CopyType value “UnSyncAssoc”. Snapshots can be created as full size replicas of a source element or as delta replicas of a source element. Snapshots usually have lower space consumption and lower copy engine overhead than either split mirrors or clones used as point-in-time images. Snapshots are only supported as local replicas hosted on the same storage system as the associated source element. A provider defines only one instance of StorageReplicationCapabilities for managing snapshots. This instance indicates one of two values for SupportedSynchronizationType:

- Full size: SupportedSynchronizationType = “UnSyncAssoc-Full”
- Delta: SupportedSynchronizationType = “UnSyncAssoc-Delta”

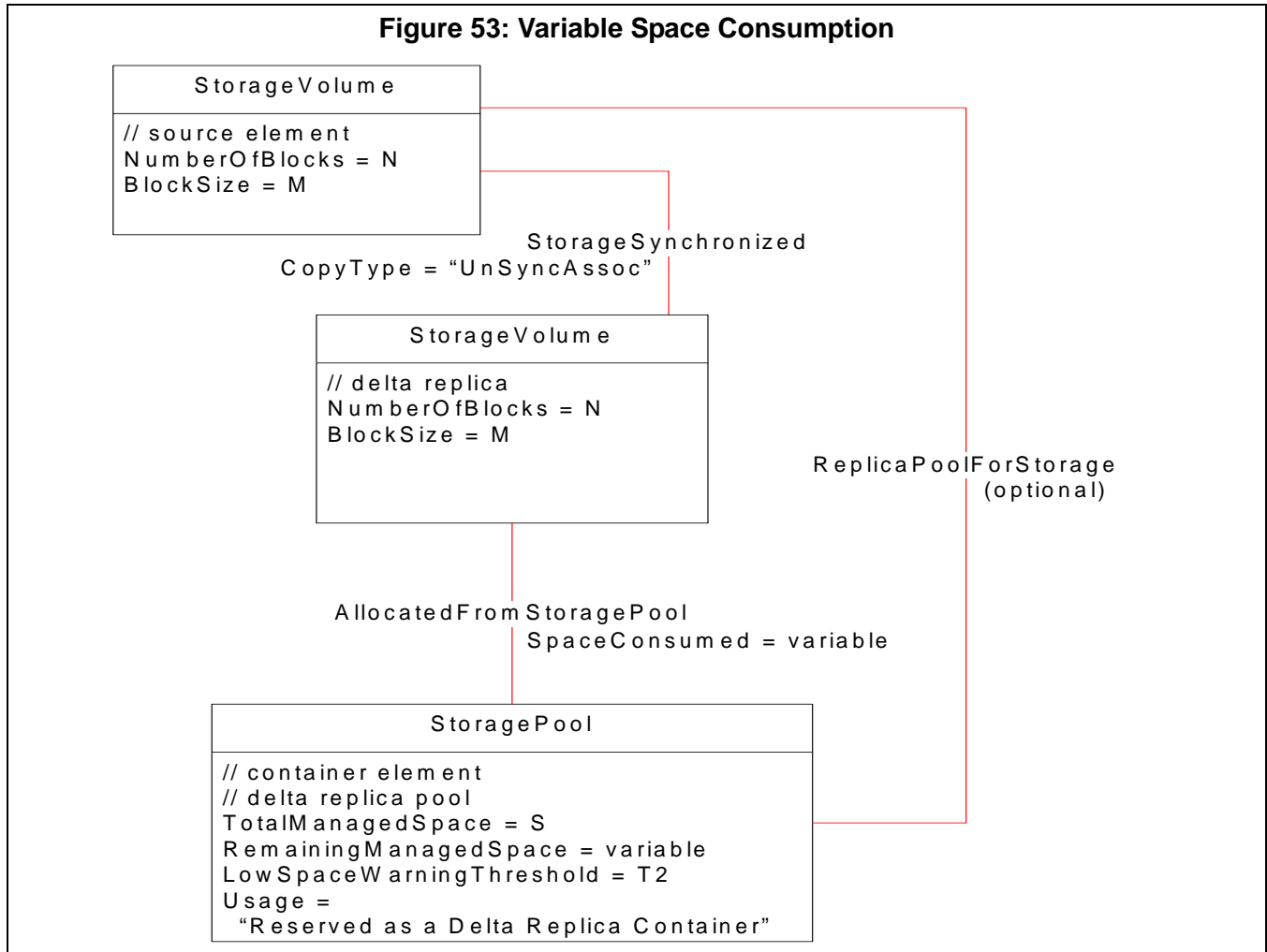
Snapshot providers may deploy either a fixed space consumption model or a variable space consumption model for snapshot replicas. A full size replica always uses a fixed space consumption model. A delta replica may use either a fixed or a variable model. Replica elements based on the variable model shall be created in special pools for delta replicas. A provider indicates support for special pools by including the value “Reserved as a Delta Replica Container” in StorageConfigurationCapabilities.SupportedStoragePoolUsage[]. The replica AllocatedFromStoragePool.SpaceConsumed property has a constant value for the fixed model and a volatile, increasing value for the variable model. The RemainingManagedSpace property for the corresponding pool has a volatile, decreasing value if the pool contains replicas based on the variable model. Figure 52 and Figure 53 show the fixed and variable space consumption models for delta snapshots:



For full size snapshots, NumberOfBlocks and BlockSize indicate the actual size of the target element which is as large or larger than the source element. For delta snapshots, NumberOfBlocks and BlockSize have the same

values as the associated source element. Delta reservation properties are only used for snapshots created by the CreateReplica method using fixed space consumption.

Figure 53: Variable Space Consumption



The instances of StorageReplicationCapabilities for "UnSyncAssoc-Delta" and "UnSyncAssoc-Full" may use the patterns detailed in Table 147, "Patterns Supported for StorageReplicationCapabilities".

Table 147: Patterns Supported for StorageReplicationCapabilities

SupportedSynchroniza tionType	Supported...Actions[n]	DeltaReplicaPoolAcce ss	Space Consumption
UnSyncAssoc-Delta	"Replica Attachment"	Any pool or extent	Fixed
UnSyncAssoc-Delta	"Replica Creation"	Any pool or extent	Fixed
UnSyncAssoc-Delta	"Replica Attachment"	Shared or Exclusive	Variable
UnSyncAssoc-Delta	"Replica Creation"	Shared or Exclusive	Variable
UnSyncAssoc-Full	"Replica Attachment"	n/a	Fixed
UnSyncAssoc-Full	"Replica Creation"	n/a	Fixed

The steps required to create a snapshot vary for each pattern. There are a number of common steps.

Step 1 the provider may limit the maximum number of replicas per source element. Verify that the limit is not exceeded when a new replica is created. The provider may restrict snapshots to independent source elements. If the source element is a replica, verify that the provider allows snapshots of local or remote replicas.

Step 2: locate a candidate pool eligible to contain a new snapshot. This is a special pool if the `DeltaReplicaPoolAccess` value is “Shared” or “Exclusive”. A shared, special pool is a preexisting element supplied by the provider. The special pool may be populated with virtual devices that do not consume space until the `AttachOrModifyReplica` method is invoked at a later time. An exclusive, special pool is created the first time a new delta replica is created for a source element that currently has no associated delta replicas. The operation for locating or creating a special pool for delta replicas is described in 9.6.4, “Creating and Managing Pools for Delta Replicas”. If snapshots can be created in any pool, enumerate all existing pool instances and begin screening the pools for eligibility. If snapshots are created by the `AttachOrModifyReplica` method, all existing storage elements in each candidate pool should be screened for eligibility in a subsequent step.

Step 3: For the special pool or for the pool being screened, access the associated `StorageCapabilities` instance and invoke `CreateSetting` to generate a modifiable setting object to be used as a goal parameter for one or more method invocations. Set `StorageExtentInitialUsage` to either “Local Replica Target” for a full snapshot or “Delta Replica Target” for a delta snapshot.

If the operation will use `CreateReplica` to create a delta snapshot using fixed space consumption, the `DeltaReservationMin`, `DeltaReservationGoal` and `DeltaReservationMax` properties are set by the client to appropriate values for a new delta replica. The values are set in the unassociated `StorageSetting` element to be passed as a goal parameter to an extrinsic method. The client cannot modify the values of delta reservation properties in a `StorageSetting` element associated to an existing storage element. The values set by the client satisfy the relationship:

$$\text{DeltaReservationMin} \leq \text{DeltaReservationGoal} \leq \text{DeltaReservationMax}$$

as constrained by the provider. The client cannot decrease the value of `DeltaReservationMin` and cannot increase the value of `DeltaReservationMax` returned by the provider. If the provider supports a fixed space consumption model, the client estimates the fixed size of the delta replica as a percentage of the source element size and the provider determines the actual size when the element is created.

Step 4: Skip this step if `CreateReplica` is used to create a delta replica with variable space consumption. For all other cases, screen the candidate pool or the storage elements contained in the pool. If `AttachOrModifyReplica` is used to create a delta replica with variable space consumption, search the special delta replica pool for a virtual storage element not in use as a replica target. For all fixed space consumption cases, the client calculates a replica size value for the screening operation. Use the source element size if a full snapshot replica is created. Use the `DeltaReplicaMax` percentage times the source element size if a delta snapshot replica is created. The generated setting created in step 3 is used as the goal parameter for the screening methods. Search existing volumes for replica target candidates as described in “Finding and Creating Target Elements” if `AttachOrModifyReplica` is used as the method to create the replica. Select a returned volume based on best fit or some other appropriate filter. Invoke `GetSupportedSizes` or `GetSupportedSizeRange` and verify that the replica size is supported by the candidate pool if `CreateReplica` is used. Proceed to step 5 if an eligible candidate element is found. Otherwise, proceed to the next candidate pool. If no candidates are located from existing pools, the client may follow recipes in block services to create a new candidate pool or element. Omit the `Size` parameter whenever a virtual replica element is created. Note: a client may elect to bypass screening and require a user to manually select a candidate pool or target element.

Step 5: invoke `AttachOrModifyReplica` or `CreateReplica` to create a new snapshot. The setting property values from the goal parameter apply to the new replica. The provider determines which setting property values from the goal parameter are copied to an existing setting instance when `AttachOrModifyReplica` is invoked. If a delta replica is created, the `NumberOfBlocks` and `BlockSize` values of the source element are assigned to the target.

The properties listed in Table 148, “Space Consumption Properties” are used to monitor and manage space consumption for delta replicas using a variable space consumption pattern.

Table 148: Space Consumption Properties

Delta Replica Property – Variable Space Consumption	Value	Modifiable
StorageExtent.NumberOfBlocks: valid for all elements. Same value as associated source element.	constant	no
StorageExtent.BlockSize: valid for all elements. Same value as associated source element.	constant	no
StoragePool.RemainingManagedSpace: valid for all pools. Value decreases by BlockSize each time replica consumes a block in the pool.	volatile	no
StoragePool.TotalManagedSpace: valid for all pools.	constant	no
StoragePool.LowSpaceWarningThreshold: valid for special delta replica pools if provider supports pool warning thresholds. Value 0 to 100.	constant	yes
AllocatedFromStoragePool.SpaceConsumed: valid for all elements. Value increases by BlockSize each time replica consumes a block in the pool.	volatile	no

The properties listed in Table 149, “Space Consumption Properties, Fixed Pattern” are used to monitor and manage space consumption for delta replicas using a fixed space consumption pattern.

Table 149: Space Consumption Properties, Fixed Pattern

Delta Replica Property – FixedSpace Consumption	Value	Modifiable
StorageExtent.NumberOfBlocks: valid for all elements. Same value as associated source element.	constant	no
StorageExtent.BlockSize: valid for all elements. Same value as associated source element.	constant	no
StorageExtent.DeltaReservation: valid for target elements. Value set by CreateReplica method providers for delta replicas.	constant	no
StoragePool.RemainingManagedSpace: valid for all pools. Value decreases by fixed element size when element is created.	constant	no
StoragePool.TotalManagedSpace: valid for all pools.	constant	no
AllocatedFromStoragePool.SpaceConsumed: valid for all elements. Value set to fixed element size when element is created.	constant	no
StorageSetting.DeltaReservationMin: Value is % of source element size that is minimum fixed size. Used only with CreateReplica method for delta replicas.	constant	yes (goal)
StorageSetting.DeltaReservationMax: Value is % of source element size that is maximum fixed size. Used only with CreateReplica method for delta replicas.	constant	yes (goal)
StorageSetting.DeltaReservationGoal: Value is % of source element size that is the client goal for the fixed size. Used only with CreateReplica method for delta replicas.	constant	yes (goal)

Two of the above properties have volatile values automatically changed by the provider when a delta replica uses a variable space consumption model. SpaceConsumed increases and RemainingManagedSpace decreases as the associated source element is updated. When a delta replica consumes an additional block, SpaceConsumed increases by the value of BlockSize and RemainingManagedSpace decreases by the value of BlockSize. If the replica uses a fixed space consumption model, the values of these two properties are constant and change only when an extrinsic method is invoked to create or modify the replica element. The value of SpaceConsumed at the instant the delta replica is created is zero if no space is reserved or greater than zero if space is reserved. The value of RemainingManagedSpace is decreased by the value of SpaceConsumed at the instant the replica is created.

The ModifySynchronization method can be invoked to manage existing snapshots. The subprofile supports the following operations:

- 1) A snapshot can be reused by invoking a “Resync” operation. This releases all of the space consumed by a snapshot using the variable space consumption model. The WhenSynced property in StorageSynchronized is reset to a new date/time value.
- 2) A “Detach” operation releases all of the space consumed by a snapshot using the variable space consumption model. The detached target element can be reused for another purpose or deleted by invoking the ReturnToStoragePool method. If the snapshot was not previously detached, invocation of ReturnToStoragePool deletes the StorageSynchronized association.
- 3) Snapshot space consumption can be stopped by invoking a “Quiesce” operation. If the associated source element is updated while the snapshot is in “Quiesced” state it is no longer a valid point-in-time image.
- 4) The source element can be restored from a snapshot by invoking a “Restore” operation. This may follow a client action that blocks host I/O to both the source element and all associated snapshot elements until the restore operation is completed.

9.6.8 Managing Background Copy

Background copy is a full copy operation that copies all blocks from a source element to a replica element. An initial background copy is normally started by a provider when a mirror or a clone is created. Initial background copy is not normally started when a snapshot is created. A provider may allow a client to initiate a deferred background copy. Management of background copy is an optional provider capability indicated to a client for each supported CopyType value using properties in StorageReplicationCapabilities. Deferred background copy for snapshots is supported if SupportedModifyOperations[] includes “Start Copy” and “Stop Copy”. Deferred background copy for mirrors is supported if InitialSynchronizationDefault has a value other than “Not Managed” or “Not Applicable”. Copy priority can be managed for any CopyType if ReplicationPriorityDefault has a value other than “Not Managed” or “Not Applicable”.

A ModifySynchronization Operation value of “Start Copy” or “Stop Copy” may be invoked for snapshots. A “Start Copy” operation causes a snapshot to transition from “Idle” state to “Copy In Progress” state to “Frozen” state. A “Stop Copy” operation causes a snapshot to transition from “Copy In Progress” state to “Idle” state.

If initial background copy is not initiated when a mirror is created, a subsequent sequence of ModifySynchronization operations that may include Prepare and Resync should start a background copy operation.

The InitialSynchronization property in the goal parameter may be set to indicate whether or not an initial background copy operation is initiated at the time a replica is created. The ReplicationPriority property in the goal parameter may be set to override the default copy I/O rate priority.

A client may invoke ModifyInstance to modify the value of CopyPriority for a StorageSynchronized association. This allows a client to manage the copy I/O rate and the priority of peer I/O operations relative to host I/O operations. CopyPriority may be modified before or during a background copy operation. Standard CopyPriority values are:

- Low – peer I/O is lower priority than host I/O
- Medium – peer I/O is the same priority as host I/O
- High – peer I/O is higher priority than host I/O

EXPERIMENTAL

9.6.9 Recipes

The recipes show how the extrinsic methods of the subprofile may be used. The ModifySynchronization method is the only mandatory method of the subprofile. Instances of StorageReplicationCapabilities indicate to the client which of the optional extrinsic methods are supported by a provider. If the provider supports an extrinsic method, the corresponding recipe shows required behavior.

9.6.10 Replica Modification

This recipe shows how to use the mandatory ModifySynchronization method.

```
// NAME: Replica Modification
// FILE: CopyServicesSP_ModSync
//
// DESCRIPTION: The synchronization state of an associated replica target
// is modified by invocation of the ModifySynchronization extrinsic
// method. The client verifies that the requested operation is supported
// by the provider before the method is invoked. The client does not proceed
// if an invalid state transition is attempted.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $StgSync is a reference to a StorageSynchronized association to be
// modified. #Operation is the ModifySynchronization operation to be
// executed. If the operation completes successfully, the $StgSync
// instance is refreshed to get the current SyncState and WhenSynced
// values.
//
// Locate the instance of StorageConfigurationService to be used for
// method invocation. Locate the instance of StorageReplicationCapabilities
// to be used for validity checking.
//
$SourceVol = GetInstance (
    $StgSync.SystemElement,
    false, false, false,
    {"SystemName", "Usage"})
$TargetVol = GetInstance (
    $StgSync.SyncedElement,
    false, false, false,
    {"SystemName", "Usage"})
$Host = Associators (
```

Copy Services Subprofile

```

$SourceVol->,
"CIM_SystemDevice",
"CIM_ComputerSystem",
null, null, false, false, null)
$SCS = Associators (
    $Host->,
    "CIM_HostedService",
    "CIM_StorageConfigurationService",
    null, null, false, false, null)
$L[] = Associators (
    $SCS->,
    "CIM_ElementCapabilities",
    "CIM_StorageReplicationCapabilities",
    null, null, false, false, null)

// Map StorageSynchronized.CopyType to
//   StorageReplicationCapabilities.SupportedSynchronizationType.

#CT_to_SST_map = {2, 4, 6, 8, 9}
#CT = $StgSync.CopyType
#SST = #CT_to_SST_map[#CT] // 1st mapping step
if ($TargetVol.Usage == 12) {#SST++} // 2nd step -- delta snapshot
if ($TargetVol.SystemName != $SourceVol.SystemName) {#SST++}
                                // 3rd step -- remote mirror

// finished with mapping

for #i in $L[]
{ // find the correct instance of StorageReplicationCapabilities
    if ($L[#i].SupportedSynchronizationType == #SST)
    {
        $SRC = $L[#i]
        break
    }
}

//
// Verify that the requested operation is supported by the provider.
//
if (!contains(#Operation, $SRC.SupportedModifyOperations[]))
{
    <error: requested ModifySynchronization operation unsupported>
}

//
// Verify that StorageSynchronized.SyncState is in a valid state for
// entry to the state diagram for the requested operation. A client
// should construct a valid transition table for each provider. The

```

```

// following tables are based on state transition diagrams in the Copy
// Services subprofile specification. The tables are indexed by #Operation.
// Values in the table are:
//     0: Invalid value, will not match
//     1: Operation can start from any steady state
//     2-15: specific prerequisite SyncState values
// Most operations can start from one or two steady states.
//

#Detach = 2 // Detach operation can start from any state
// Cannot start other operations from "in progress" states
#InProgress[] = {3, 5, 7, 8, 10, 15}
#StartState_Mirror1 = {0, 0, 0, 9, 4, 13, 2, 4, 6, 9, 1, 1, 0, 0}
#StartState_Mirror2 = {0, 0, 0, 9, 4, 13, 13, 4, 6, 9, 1, 1, 0, 0}
#StartState_Snapshot1 = {0, 0, 0, 0, 4, 11, 2, 4, 11, 0, 0, 0, 11, 15}
#StartState_Snapshot2 = {0, 0, 0, 0, 4, 14, 9, 4, 14, 0, 0, 0, 11, 15}

#SS = $StgSync.SyncState
if (#Operation != #Detach) // Detach can start from any state
{
    if (contains(#SS, #InProgress[]))
    {
        <error: invalid state transition attempted>
    }
    if (#CT == 4) // Check snapshot state transitions
    {
        if ((#SS != #StartState_Snapshot1[#Operation]) &&
            (#SS != #StartState_Snapshot2[#Operation]))
        {
            <error: invalid state transition attempted>
        }
    } else // Check mirror state transitions
    {
        if ((#SS != #StartState_Mirror1[#Operation]) &&
            (#SS != #StartState_Mirror2[#Operation]) &&
            (#StartState_Mirror1[#Operation] != 1))
        {
            <error: invalid state transition attempted>
        }
    }
}

//
// Passed all validation checks. Invoke ModifySynchronization.
%InArguments["Synchronization"] = $StgSync->
%InArgument["Operation"] = #Operation
#r = InvokeMethod(

```

```

$SCS->,
"ModifySynchronization",
%InArguments,
%OutArguments)
if (#r != 0 && #r != 4096)
{
    <error: modify failed, stop recipe and examine CIM_Error>
}
if (#r == 4096)
{
    $Job-> = %OutArguments["Job"]
    <wait for instance modification indication for job completion>
    $Job = GetInstance(
        $Job->,
        false, false, false, null)
    if ($Job.JobState != 7) // is it "Completed"?
    {
        <error: modify job failed, stop and examine CIM_Error>
    }
}
if (#Operation != #Detach)
// if not "detach" get a fresh copy of StorageSynchronized
{
    $StgSync = GetInstance(
        $StgSync->,
        false, false, false,
        {"WhenSynced", "SyncState"})
    if (contains($StgSync.SyncState, #InProgress)) // In a transition?
    { // This is an optional wait for an instance mod indication
        // showing a steady state.
        <wait for instance mod indication for $StgSync.SyncState>
        $StgSync = GetInstance( // refresh to show steady state
            $StgSync->,
            false, false, false,
            {"WhenSynced", "SyncState"})
    }
}

// Recipe complete -- $StgSync is now the StorageSynchronized association
// instance showing the final SyncState for the modify operation unless
// the operation was "detach". If the operation is detach, the
// StorageSynchronized association was deleted.

```

9.6.11 Replica Creation Or Attachment

This recipe shows how to use the CreateReplica and the AttachOrModifyReplica methods.

```

// NAME: Replica Creation Or Attachment
// FILE: CopyServicesSP_CreateOrAttach

```

```

//
// DESCRIPTION: Create a new replica target element or attach an
// existing element eligible to be used as a replica target. The client
// performs a sequence of validation steps to ensure that the operation will
// succeed using the selected input elements.
// The recipe supports both the CreateReplica and the AttachOrModifyReplica
// extrinsic methods.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SourceElement is the replication source and must be supplied.
// $TargetElement is an optional element selected as the replication target.
// If $TargetElement is supplied, the "attach" method will be invoked.
// If not, the "create" method will be invoked.
// $SCC is the instance of CIM_StorageConfigurationCapabilities
// controlling the recipe.
// $SRC is the instance of CIM_StorageReplicationCapabilities
// controlling the recipe.
// $SCS is the instance of CIM_StorageConfigurationService controlling
// the recipe.
// $TargetPool is an optional pool where a new replica is created.
// $TargetPool is not supplied if $TargetElement is supplied.
// $Goal is an optional instance of StorageSetting to be used as a goal
// parameter for a replica created in $TargetPool.
// $Pipe is an optional instance of ReplicationPipe that may be supplied
// for remote replication operations.
//

// Map StorageReplicationCapabilities.SupportedSynchronizationType value to the
// corresponding StorageSynchronized.CopyType value.
#SST_to_CT_map[] = {0, 0, 2, 2, 3, 3, 4, 4, 5, 6}
#SST = $SRC.SupportedSynchronizationType
#CT = #SST_to_CT_map[#SST]

// Create or Attach?
if ($TargetElement == null)
{ // Use CreateReplica
    if ((!contains(2, $SRC.SupportedAsynchronousActions[]) &&
        (!contains(2, $SRC.SupportedSynchronousActions[]))
        (
            <error: replica creation not supported>
        )
        #attach = false
    } else
    (
        if ((!contains(4, $SRC.SupportedAsynchronousActions[]) &&
            (!contains(4, $SRC.SupportedSynchronousActions[]))

```

Copy Services Subprofile

```

(
    <error: replica attachment not supported>
)
#attach = true
}

// Tables for checking $TargetElement.Usage value
#AllUsage[] = {2, 8, 9, 10, 11, 12}
#RemoteUsage[] = {2, 9, 11}
#LocalUsage[] = {2, 8, 10}
#DeltaSnapshotUsage = 12
// Table for checking $TargetPool.Usage value
#PoolUsage[] = {0, 0, 6, 7, 6, 7, 6, 4, 6, 5}

if (#attach)
{ // validation checks for replica attachment
    $Refs[] = ReferenceNames(
        $TargetVol.getObjectPath(),
        "CIM_StorageSynchronized",
        null)
    if ($Refs[].size() != 0) // element already a replica source or target
    {
        <error: invalid replication target element.
    }
    if ((#SST != 6) && (#SST != 7)) // Check size unless creating a snapshot
    {
        #SourceSize = $SourceElement.NumberOfBlocks * $SourceElement.BlockSize
        #TargetSize = $TargetElement.NumberOfBlocks * $TargetElement.BlockSize
        if (#TargetSize < #SourceSize)
        (
            <error: replication target element has insufficient size>
        )
    }
    if ((#SST != 3) && (#SST != 5))
    ( // remote replication -- source and target must have different hosts
        if ($SourceVol.SystemName == $TargetVol.SystemName)
        {
            <error: target must be located on a remote host>
        }
    } else
    { // local replication -- source and target must have the same host
        if ($SourceVol.SystemName != $TargetVol.SystemName)
        {
            <error: target and source must be located on the same host>
        }
    }
    #Usage = $TargetElement.Usage
}

```

```

if ((contains(#Usage, #AllUsage[]) && (#Usage != 2))
{ // continue unless Usage is "Unrestricted"
    if (#SST == 7) // Delta snapshot
    {
        if (#Usage != #SnapshotUsage)
        {
            <error: invalid usage restriction for target element>
        }
    } else
    {
        if ((#SST == 3) || (#SST == 5) // remote replica
        {
            if (!contains(#Usage, #RemoteUsage[]))
            {
                <error: invalid usage restriction for target element>
            }
        } else
        { // all local replica types except delta snapshots
            if (!contains(#Usage, #LocalUsage[]))
            {
                <error: invalid usage restriction for target element>
            }
        }
    }
}
if ($Pipe != null)
{
    if ($Pipe.AggregationBehavior != 4)
    {
        <error: not a top-level replication pipe>
    }
}
} else
{ // validation checks for replica creation
    if ($TargetPool != null)
    {
        #Usage = $TargetPool.Usage
        if ((#Usage != 2) && (#Usage != #PoolUsage[#SST]))
        {
            <error: invalid usage restriction for target pool>
        }
    }
}
} // completed all validation checks

// Invoke either AttachOrModifyReplica or CreateReplica
if (#Attach)
{

```

Copy Services Subprofile

```

%InArguments["SourceElement"] = $SourceElement->
%InArguments["TargetElement"] = $TargetElement->
%InArguments["CopyType"] = #CT
%InArguments["ReplicationPipe"] = $Pipe
#r = InvokeMethod(
    $SCS->,
    "AttachOrModifyReplica",
    %InArguments,
    %OutArguments)
} else
{
    %InArguments["SourceElement"] = $SourceElement->
    %InArguments["CopyType"] = #CT
    %InArguments["TargetSettingGoal"] = $Goal->
    %InArguments["TargetPool"] = $TargetPool->
    #r = InvokeMethod(
        $SCS->,
        "CreateReplica",
        %InArguments,
        %OutArguments)
}
if (#r != 0 && #r != 4096)
{
    <error: replication method failed, stop and examine CIM_Error>
}
if (#r == 4096)
{
    $Job-> = %OutArguments["Job"]
    <wait for instance modification indication for job completion>
    $Job = GetInstance(
        $Job->,
        false, false, false, null)
    if ($Job.JobState != 7) // "Completed"?
    {
        <error: replication job failed, stop and examine CIM_Error>
    }
    if (!#Attach)
    {
        $TL[] = Associators(
            $Job->,
            "CIM_AffectedJobElement",
            "CIM_StorageVolume",
            null, null, false, false, null)
        $TargetElement = $TL[0]
    }
} else

```



```

{
    if (!#Attach)
    {
        $TargetElement-> = %OutArguments["TargetElement"]
        $TargetElement = GetInstance(
            $TargetElement->,
            false, false, false, null)
    }
}

if (#CT != 5) // not "UnSyncUnAssoc"
{ // locate new StorageSynchronized association for the target
    $SL[] = References(
        $TargetElement->,
        "CIM_StorageSynchronized",
        "SyncedElement",
        false, false, null)
    $StgSync = $SL[0]
}

// End of recipe. If successful, $TargetElement is the target replica
// instance and $StgSync is an instance of the StorageSynchronized
// association between $SourceElement and $TargetElement.

```

9.6.12 Create Remote Replication Connection

This recipe shows how to use the `CreateOrModifyReplicationPipe` method.

```

// NAME: Create Remote Replication Connection
// FILE: CopyServicesSP_CreateConnection
//
// DESCRIPTION: Establish a peer-to-peer connection between two storage
// arrays. The connection is used to manage data transport for remote
// replication. The user selects two array systems to be connected using
// symmetric endpoint sets for each array. Endpoint sets are optional and
// may be null lists. The recipe supplies a number of validity checks
// to ensure that the two arrays are compatible.
// The CreateOrModifyReplicationPipe method is invoked to establish
// the connection. Output is a ReplicationPipe composition with one top-level
// ReplicationPipe element and one or more second-level ReplicationPipe
// elements corresponding to the number of supplied endpoint pairs.
// IF $Pipe is supplied as an input parameter along with a #Modifier
// parameter, an existing connection is modified.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SourceSRC and $TargetSRC are instances of StorageReplicationCapabilities
// supporting the "ReplicationPipe Creation" action for remote replication.
// $SCS is the instance of StorageConfigurationService to be used for method
// invocation.

```

Copy Services Subprofile

```

// $SourceSystem is a top-level or leaf ComputerSystem
// $TargetSystem is a top-level or leaf ComputerSystem
// $SourceEnd[] is a list of ProtocolEndpoint instances for ports on
// the source array
// $TargetEnd[] is a list for the target array
// If $Pipe is not null on input, an existing connection is modified using
// #Modifier as a parameter
//
// OUTPUT: $Pipe is the new top-level ReplicationPipe when a new connection
// is created.
//

// Verify that the peer systems are compatible for remote replication
if (($SourceSRC.PeerConnectionProtocol != TargetSRC.PeerConnectionProtocol) ||
    ($SourceSRC.BidirectionalConnectionsSupported !=
        $TargetSRC.BidirectionalConnectionsSupported) ||
    ($SourceSRC.RemoteReplicationServicePointAccess !=
        $TargetSRC.RemoteReplicationServicePointAccess))
{
    <error: peer systems not compatible>
}

// All verification checks passed. Invoke CreateOrModifyReplicationPipe
%InArguments["SourceSystem"] = $SourceSystem->
%InArguments["TargetSystem"] = $TargetSystem->
%InArguments["SourceEndpoint"] = $SourceEnd->[]
%InArguments["TargetEndpoint"] = $TargetEnd->[]
if ($Pipe != null)
{
    %InArguments["ReplicationPipe"] = $Pipe->
    %InArguments["Modifier"] = #Modifier
}
#r = InvokeMethod(
    $SCS->,
    "CreateOrModifyReplicationPipe",
    %InArguments,
    %OutArguments)
if (#r != 0)
{
    <error: failed to establish connection>
}

// Get the new top-level pipe.
if ($Pipe != null)
{
    $Pipe-> = %OutArguments["ReplicationPipe"]
    $Pipe = GetInstance(

```

```

        $Pipe->,
        false, false, false, null)
    }

    // Connection is established or modified and remote replicas can now
    // be created for this connected pair of arrays. The connection is either
    // bi-directional or uni-directional as set by the provider.

```

9.6.13 Create Remote Replication Buffer

This recipe shows how to use the CreateReplicationBuffer method.

```

// NAME: Create Remote Replication Buffer
// FILE: CopyServicesSP_CreateBuffer
//
// DESCRIPTION: Create a private element used as a write-ahead buffer
// for CopyType "Async" or "Sync" for remote replication. The buffer is
// modeled as an instance of CIM_Memory created by the
// CreateReplicationBuffer extrinsic method. The flow of the recipe
// is directed by properties in StorageReplicationCapabilities:
// RemoteBufferElementType and RemoteBufferLocation.
// The output is new instance, $Buffer.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SRC is an instance of StorageReplicationCapabilities supporting the
// "Buffer Creation" action for remote replication.
// $SCS is the instance of StorageConfigurationService associated with $SRC.
// $System is a top-level or leaf ComputerSystem that will host the buffer
// as a system device.
// $Pipe is an optional top-level ReplicationPipe representing the connection.
// $Pipe may be null unless RemoteBufferHost == "Pipe".
// $Pool is the optional target pool for the buffer.
// $Element is the optional target element for the buffer. Can be a StorageVolume
// or LogicalDisk with Usage of "Unrestricted" or "Element Component".
//
// Select element as replication buffer host.
if ($SRC.RemoteBufferHost == 4)
{
    $Host = $Pipe
} else
{
    $Host = $System
}
if ($Host == null)
{
    <error: invalid host element supplied>
}

```

```

// Select element component. If both are null, provider will select the
// component element and size.
if ($SRC.RemoteBufferElementType == 3) // pool
{
    %InArguments["TargetPool"] = $Pool->
}
if ($SRC.RemoteBufferElementType == 4) // storage element
{
    %InArguments["TargetElement"] = $Element->
}

%InArguments["Host"] = $Host->
#r = InvokeMethod(
    $SCS->,
    "CreateReplicationBuffer",
    %InArguments,
    %OutArguments)
if (#r != 0 && #r != 4096)
{
    <error: buffer creation failed, stop and examine CIM_Error>
}
if (#r == 4096)
{
    $Job-> = %OutArguments["Job"]
    <wait for instance mod indication for job completion>
    $Job = GetInstance(
        $Job->,
        false, false, false, null)
    if ($Job.JobState != 7) // "Completed"?
    {
        <error: creation job failed, stop and examine CIM_Error>
    }
    $L[] = Associators(
        $Job->,
        "CIM_AffectedJobElement",
        "CIM_Memory",
        null, null, false, false, null)
    $Buffer-> = $L->[0]
} else
{
    $Buffer-> = %OutArguments["ReplicaBuffer"]
}
$Buffer = GetInstance(
    $Buffer->,
    false, false, false, null)

```

9.7 Registered Name and Version

Copy Services version 1.2.0

9.8 CIM Elements

Table 150: CIM Elements for Copy Services

Element Name	Requirement	Description
CIM_AssociatedMemory (9.8.1)	Optional	Associates remote replication buffer to a peer system element or a replication pipe.
CIM_BasedOn () (9.8.2)	Optional	May be used for replica buffer element created from a concrete extent.
CIM_ConcreteDependency (9.8.3)	Optional	Associates remote replica target element to a peer-to-peer connection.
CIM_ElementCapabilities (Associates StorageReplicationCapabilities and StorageConfigurationService) (9.8.4)	Mandatory	
CIM_EndpointOfNetworkPipe (9.8.5)	Optional	Associates peer-to-peer ProtocolEndpoint to lower-level NetworkPipe.
CIM_HostedNetworkPipe (9.8.6)	Optional	Associates replication NetworkPipe to replication Network domain element.
CIM_Memory (9.8.7)	Optional	Write ahead buffer element for a remote replication service.
CIM_Network (9.8.8)	Optional	Specialized peer-to-peer network for a remote replication service.
SNIA_ReplicationPipe (9.8.9)	Optional	Identifies peer-to-peer connection for a remote replication service.
CIM_NetworkPipeComposition (9.8.10)	Optional	Associates one top-level NetworkPipe to one or more lower-level pipes.
CIM_ProtocolEndpoint (9.8.11)	Optional	Base definition is in Fabric Profile. Special purpose endpoint that controls a peer-to-peer connection.
CIM_ReplicaPoolForStorage (9.8.12)	Optional	Associates special storage pool for delta replicas to a source element.
CIM_StorageCapabilities (9.8.13)	Mandatory	Base definition is in Block Services Package.
CIM_StorageConfigurationCapabilities (9.8.14)	Mandatory	Base definition is in Block Services Package. Adds two properties.
CIM_StorageConfigurationService (9.8.15)	Mandatory	Base definition is in Block Services Package. Methods are described in the Extrinsic Methods clause.
CIM_StoragePool (9.8.16)	Mandatory	Base definition is in Block Services Package.
CIM_StorageReplicationCapabilities (9.8.17)	Mandatory	A set of properties that describe the capabilities of a copy services provider.
CIM_StorageSetting (9.8.18)	Mandatory	Base definition is in Block Services Package.

Table 150: CIM Elements for Copy Services

Element Name	Requirement	Description
CIM_StorageSynchronized (Between LogicalDisk elements) (9.8.19)	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory. Associates replica target element to a source element.
CIM_StorageSynchronized (Between StorageVolume elements) (9.8.20)	Conditional	Associates replica target element to a source element. Property definitions and descriptions are identical to those for LogicalDisk usage.
CIM_SystemComponent (9.8.21)	Optional	Associates peer system element to remote replication network domain.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageSynchronized	Mandatory	All instance creation indications for StorageSynchronized.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageSynchronized	Mandatory	All instance deletion indications for StorageSynchronized.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageSynchronized AND SourceInstance.CIM_StorageSynchronized::SyncState <> PreviousInstance.CIM_StorageSynchronized::SyncState	Optional	Experimental CQL - Synchronization state transition for a replica association.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageSynchronized AND SourceInstance.SyncState <> PreviousInstance.SyncState	Mandatory	Deprecated WQL - Synchronization state transition for a replica association.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_Memory	Optional	All instance creation indications for replication buffers.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_Memory	Optional	All instance deletion indications for replication buffers.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_ReplicationPipe AND SourceInstance.SNIA_ReplicationPipe::OperationalStatus <> PreviousInstance.SNIA_ReplicationPipe::OperationalStatus	Optional	Experimental CQL - Change of peer-to-peer connection operational status.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_ReplicationPipe AND SourceInstance.SyncState <> PreviousInstance.SyncState	Optional	Deprecated WQL - Synchronization state transition for a replica association.

9.8.1 CIM_AssociatedMemory

Created By: Extrinsic: CreateReplicationBuffer

Modified By: Extrinsic: Not modifiable

Deleted By: Static

Class Mandatory: Optional

Table 151 describes class CIM_AssociatedMemory.

Table 151: SMI Referenced Properties/Methods for CIM_AssociatedMemory

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

9.8.2 CIM_BasedOn ()

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 152 describes class CIM_BasedOn ().

Table 152: SMI Referenced Properties/Methods for CIM_BasedOn ()

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

9.8.3 CIM_ConcreteDependency

Created By: Extrinsic: CreateReplica, AttachReplica, AttachOrModifyReplica

Modified By: Static

Deleted By: Extrinsic: ModifySynchronization

Class Mandatory: Optional

Table 153 describes class CIM_ConcreteDependency.

Table 153: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	Top-level ReplicationPipe identifying the connection.
Dependent		Mandatory	Remote replica target element.

9.8.4 CIM_ElementCapabilities (Associates StorageReplicationCapabilities and StorageConfigurationService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 154 describes class CIM_ElementCapabilities (Associates StorageReplicationCapabilities and StorageConfigurationService).

Table 154: SMI Referenced Properties/Methods for CIM_ElementCapabilities (Associates StorageReplicationCapabilities and StorageConfigurationService)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	
ManagedElement		Mandatory	

9.8.5 CIM_EndpointOfNetworkPipe

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 155 describes class CIM_EndpointOfNetworkPipe.

Table 155: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Properties	Flags	Requirement	Description & Notes
SourceOrSink		Optional	Indicates endpoint for source or target host when the connection is uni-directional. Values: 0: Unknown 1: Source 2: Target (sink) 3: Not applicable
Dependent		Mandatory	
Antecedent		Mandatory	

9.8.6 CIM_HostedNetworkPipe

Created By: Extrinsic: CreateOrModifyReplicationPipe

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 156 describes class CIM_HostedNetworkPipe.

Table 156: SMI Referenced Properties/Methods for CIM_HostedNetworkPipe

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

9.8.7 CIM_Memory

A remote replication buffer element may be created from a concrete StorageExtent element, in volatile DRAM, in a StoragePool or in a way opaque to a client. The buffer is a private element never exposed as a LUN. A buffer element has one AssociatedMemory association to either a hosting system or a top-level replication connection pipe.

Created By: Extrinsic: CreateReplicationBuffer

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 157 describes class CIM_Memory.

Table 157: SMI Referenced Properties/Methods for CIM_Memory

Properties	Flags	Requirement	Description & Notes
DeviceID		Mandatory	
SystemCreationClass Name		Mandatory	
CreationClassName		Mandatory	
SystemName		Mandatory	

9.8.8 CIM_Network

Providers that support a remote replication service scoped by managed peer-to-peer connections must provide a primordial instance of CIM_Network discovered at the same time that all associated peer systems are discovered. All peer systems eligible to use a remote replication service must have a SystemComponent association to the Network instance.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 158 describes class CIM_Network.

Table 158: SMI Referenced Properties/Methods for CIM_Network

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
Name		Mandatory	
NameFormat		Mandatory	
ElementName		Optional	

9.8.9 SNIA_ReplicationPipe

A two-level NetworkPipe composition identifies a peer-to-peer connection between two peer system elements. Two peer protocol endpoints are associated to each lower-level pipe. All of the lower-level pipes are associated to a top-level pipe. The top-level pipe properties allow the connection state and directionality to be monitored and managed.

Created By: Extrinsic: CreateOrModifyReplicationConnection

Modified By: Extrinsic: CreateOrModifyReplicationConnection

Deleted By: Static

Class Mandatory: Optional

Table 159 describes class SNIA_ReplicationPipe.

Table 159: SMI Referenced Properties/Methods for SNIA_ReplicationPipe

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName	MN	Mandatory	
OperationalStatus		Mandatory	Described in Health and Fault Management clause.
AggregationBehavior		Mandatory	Identifies pipe as top-level or lower-level: 2: lower-level pipe with two endpoints 4: top-level pipe
Directionality		Mandatory	Indicates bi-directional or uni-directional connection. Values: 0: unknown 2: bi-directional 3: uni-directional
LongBusyInterval	M	Mandatory	Long busy interval in seconds.
LongBusyEnabled		Mandatory	

9.8.10 CIM_NetworkPipeComposition

Created By: Extrinsic: CreateOrModifyReplicationPipe

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 160 describes class CIM_NetworkPipeComposition.

Table 160: SMI Referenced Properties/Methods for CIM_NetworkPipeComposition

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

9.8.11 CIM_ProtocolEndpoint

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 161 describes class CIM_ProtocolEndpoint.

Table 161: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Properties	Flags	Requirement	Description & Notes
ProtocolIFType		Mandatory	Value is always "Other".
OtherTypeDescription		Mandatory	String identifying the peer-to-peer connection protocol.

9.8.12 CIM_ReplicaPoolForStorage

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 162 describes class CIM_ReplicaPoolForStorage.

Table 162: SMI Referenced Properties/Methods for CIM_ReplicaPoolForStorage

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

9.8.13 CIM_StorageCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 163 describes class CIM_StorageCapabilities.

Table 163: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
DeltaReservationMin		Mandatory	Refer to property descriptions for CIM_StorageSetting class.
DeltaReservationMax		Mandatory	
DeltaReservationDefault		Mandatory	Initial value for CIM_StorageSetting.DeltaReservationGoal.

9.8.14 CIM_StorageConfigurationCapabilities

This class is only defined to maintain SMI-S 1.0 backward compatibility. SMI-S 1.1 providers indicate copy services capabilities using instances of the StorageReplicationCapabilities class.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 164 describes class CIM_StorageConfigurationCapabilities.

Table 164: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedAsynchronousActions	N	Mandatory	Identify replication methods using job control. Values: 8: Replica Creation 9: Replica Modification 10: Replica Attachment
SupportedSynchronousActions	N	Mandatory	Identify replication methods not using job control. Values: 8: Replica Creation 9: Replica Modification 10: Replica Attachment
SupportedStorageElementTypes		Mandatory	Storage element types that can be replicated. Values: 2: Storage Volume 4: Logical Disk

Table 164: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedCopyTypes		Mandatory	CopyType values: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc
InitialReplicationState		Mandatory	The initial SyncState when replica creation is completed. Values: 2: Initialized 3: Prepared 4: Synchronized

9.8.15 CIM_StorageConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 165 describes class CIM_StorageConfigurationService.

Table 165: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Properties	Flags	Requirement	Description & Notes
CreateOrModifyReplicationPipe()		Optional	
CreateReplicationBuffer()		Optional	
ModifySynchronization()		Mandatory	
CreateReplica()		Optional	
AttachReplica()		Optional	Defined for 1.0.2 downward compatibility. Behaves as AttachOrModifyReplica without Goal and ReplicationPipe parameters.
AttachOrModifyReplica()		Optional	

9.8.16 CIM_StoragePool

LowSpaceWarningThreshold only applies to specialized pools created as containers for delta replica elements using dynamic, variable space consumption. The specialized pool is associated to either the StorageConfigurationService or to a single replica source element.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 166 describes class CIM_StoragePool.

Table 166: SMI Referenced Properties/Methods for CIM_StoragePool

Properties	Flags	Requirement	Description & Notes
LowSpaceWarningThreshold	M	Optional	Percentage of TotalManagedSpace triggering an alert indication. When RemainingManagedSpace reaches or falls below this percentage, the indication is generated.

9.8.17 CIM_StorageReplicationCapabilities

This class defines all of the capability properties for a replication service. A provider must supply one instance for each SupportedSynchronizationType value supported.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 167 describes class CIM_StorageReplicationCapabilities.

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedSynchronizationType		Mandatory	Provider must supply one instance of this class for each supported value. Values: 2: Async-Local 3: Async-Remote 4: Sync-Local 5: Sync-Remote 6: UnSyncAssoc-Full 7: UnSyncAssoc-Delta 8: UnSyncUnAssoc 9: Migrate
SupportedAsynchronousActions	N	Mandatory	Identify replication methods using job control. Values: 2: Replica Creation 3: Replica Modification 4: Replica Attachment 5: Buffer Creation 6: Migration
SupportedSynchronousActions	N	Mandatory	Identify replication methods not using job control. Values: 2: Replica Creation 3: Replica Modification 4: Replica Attachment 5: Buffer Creation 6: Migration 7: ReplicaPipe Creation
InitialReplicationState		Mandatory	The initial SyncState when replica creation is completed. Values: 2: Initialized 3: Prepared 4: Synchronized 5: Idle

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedModifyOperations		Mandatory	<p>Identify ModifySynchronization operations supported for this CopyType. Values:</p> <ul style="list-style-type: none"> 2: Detach 3: Fracture 4: Resync 5: Restore 6: Prepare 7: Unprepare 8: Quiesce 9: Unquiesce 10: Reset To Sync 11: Reset To Async 12: Start Copy 13: Stop Copy
ReplicaHostAccessibility		Mandatory	<p>Host access restrictions. Values:</p> <ul style="list-style-type: none"> 2: Not accessible 3: Any host may access 4: Only accessible by the associated source element host 5: Accessible by hosts other than the source element host

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
HostAccessibleState		Mandatory	Associated replicas are host accessible for these SyncState values: 2: Initialized 3: Prepare In Progress 4: Prepared 5: Resync In Progress 6: Synchronized 7: Fracture In Progress 8: Quiesce In Progress 9: Quiesced 10: Restore In Progress 11: Idle 12: Broken 13: Fractured 14: Frozen 15: Copy In Progress
LocalMirrorSnapshotSupported		Conditional	Conditional requirement: Local or remote mirrors supported. Only valid for CopyType "Sync" and "Async": true: local mirror replicas can be snapshot source element false: local mirrors cannot be snapshot source
RemoteMirrorSnapshotSupported		Conditional	Conditional requirement: Local or remote mirrors supported. Only valid for CopyType "Sync" and "Async": true: remote mirror replicas can be snapshot source element false: remote mirrors cannot be snapshot source
BidirectionalConnectionsSupported		Conditional	Conditional requirement: Peer-to-peer connections supported. Only valid for CopyType "Sync" and "Async" remote replicas: true: connections are bi-directional false: connections are uni-directional
MaximumReplicasPerSource		Mandatory	Maximum replicas of all types allowed for one source element.
MaximumPortsPerConnection		Conditional	Conditional requirement: Peer-to-peer connections supported. Maximum ports assigned to one peer-to-peer connection.

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
MaximumConnectionsPerPort		Conditional	Conditional requirement: Peer-to-peer connections supported. Maximum peer-to-peer connections for one port.
MaximumPeerConnections		Conditional	Conditional requirement: Peer-to-peer connections supported. Maximum peer-to-peer connections for one system element.
MaximumLocalReplicationDepth		Conditional	Conditional requirement: Local or remote mirrors supported. Volume A mirrors Volume B mirrors Volume C to this maximum allowable depth.
MaximumRemoteReplicationDepth		Conditional	Conditional requirement: Local or remote mirrors supported.
InitialSynchronizationDefault		Conditional	Conditional requirement: Managed background copy operations supported. Refer to CIM_StorageSetting.InitialSynchronization
ReplicationPriorityDefault		Conditional	Conditional requirement: Managed background copy operations supported. Refer to CIM_StorageSetting.ReplicationPriority
LowSpaceWarningThresholdDefault		Conditional	Conditional requirement: Snapshots supported. Default value for LowSpaceWarningThreshold. Percentage value between 0 and 100.
RemoteReplicationServicePointAccess		Conditional	Conditional requirement: Remote mirrors supported. Primary SAP for a remote replication service. Values: 2: Not specified 3: Source hosting system 4: Target hosting system 5: Proxy service
AlternateReplicationServicePointAccess		Conditional	Conditional requirement: Remote mirrors supported. Alternate SAP for a remote replication service. Used when primary SAP is not available. Values: 2: No alternate SAP 3: Source hosting system 4: Target hosting system 5: Proxy service

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
DeltaReplicaPoolAccess		Conditional	Conditional requirement: Snapshots supported. Indicates if a specialized pool is required as a container for delta replicas. Values: 2: Any pool may contain delta replicas 3: Exclusive special pool per source element 4: Shared special pool for all source elements
RemoteBufferElementType		Conditional	Conditional requirement: Remote replication buffers supported. Indicates target container element type for remote replication buffers. Values: 2: Not specified by client 3: Storage pool 4: Storage extent
RemoteBufferHost		Conditional	Conditional requirement: Remote replication buffers supported. Indicates hosting element for replication buffers. Values: 2: Associated to top-level system element 3: Associated to a ComponentCS system element 4: Associated to a NetworkPipe element
RemoteBufferLocation		Conditional	Conditional requirement: Remote replication buffers supported. Indicates location for replication buffers. Values: 2: Source systems only 3: Target systems only 4: Both source and target require a buffer
RemoteBufferSupported		Conditional	Conditional requirement: Remote replication buffers supported. Indicates if a peer-to-peer connection may use a remote replication buffer. Values: 2: Required 3: Optional
UseReplicationBufferDefault		Conditional	Conditional requirement: Remote replication buffers supported. Indicates if replication buffer usage is managed by individual replica pairs. Values: 0: Not managed at the pair level 1: Pair uses the buffer 2: Pair does not use the buffer

Table 167: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Properties	Flags	Requirement	Description & Notes
PeerConnectionProtocol		Conditional	Conditional requirement: Peer-to-peer connections supported. Opaque string identifying the peer-to-peer transport protocol supported by the hosting system.

9.8.18 CIM_StorageSetting

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 168 describes class CIM_StorageSetting.

Table 168: SMI Referenced Properties/Methods for CIM_StorageSetting

Properties	Flags	Requirement	Description & Notes
DeltaReservationMin	M	Mandatory	Minimum space reserved for a delta replica at time of creation. Value 0 to 100 is a percentage of the source element size.
DeltaReservationMax	M	Mandatory	Maximum space reserved for a delta replica at time of creation. Value 0 to 100 is a percentage of the source element size.
DeltaReservationGoal	M	Mandatory	Goal for space reserved for a delta replica at time of creation. Value 0 to 100 is a percentage of the source element size.
UseReplicationBuffer	M	Optional	Indicates that a remote mirror pair may use a replication buffer. Values: 0: Not applicable 1: Not managed 2: Use the buffer 3: Do not use the buffer
InitialSynchronization	M	Optional	Indicates that the source element should be fully copied to the target element when a replica is created. Values: 0: Not applicable 1: Not managed 2: Start copy operation 3: Do not start copy operation

Table 168: SMI Referenced Properties/Methods for CIM_StorageSetting

Properties	Flags	Requirement	Description & Notes
ReplicationPriority	M	Optional	Priority of copy engine I/O relative to host I/O. Values: 0: Not applicable 1: Not managed 0: Not managed 2: Lower than host I/O 3: Same as host I/O 4: Higher than host I/O

9.8.19 CIM_StorageSynchronized (Between LogicalDisk elements)

Created By: Extrinsic: CreateReplica, AttachReplica, AttachOrModifyReplica

Modified By: Extrinsic: ModifySynchronization

Deleted By: Extrinsic: ModifySynchronization

Class Mandatory: LVM

Table 169 describes class CIM_StorageSynchronized (Between LogicalDisk elements).

Table 169: SMI Referenced Properties/Methods for CIM_StorageSynchronized (Between LogicalDisk elements)

Properties	Flags	Requirement	Description & Notes
WhenSynced	N	Mandatory	If the replica is a PIT image, this value is the date/time created.
SyncMaintained		Mandatory	Boolean indicating whether synchronization is maintained.
CopyType		Mandatory	Type of association between source and target. Values: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc 6: Migrate

Table 169: SMI Referenced Properties/Methods for CIM_StorageSynchronized (Between LogicalDisk elements)

Properties	Flags	Requirement	Description & Notes
ReplicaType		Optional	Informational property describing the type of replication. Values: 0: Not specified 2: Full Copy 3: Before Delta 4: After Delta 5: Log
SyncState		Mandatory	State of the association between source and target. Values: 2: Initialized 3: PrepareInProgress 4: Prepared 5: ResyncInProgress 6: Synchronized 7: FractureInProgress 8: QuiesceInProgress 9: Quiesced 10: RestoreInProgress 11: Idle 12: Broken 13: Fractured 14: Frozen 15: CopyInProgress
CopyPriority	M	Optional	Priority of copy engine I/O relative to host I/O. Values: 0: Not managed 1: Lower than host I/O 2: Same as host I/O 3: Higher than host I/O
SyncedElement		Mandatory	
SystemElement		Mandatory	

9.8.20 CIM_StorageSynchronized (Between StorageVolume elements)

Created By: Extrinsic: CreateReplica, AttachReplica, AttachOrModifyReplica

Modified By: Extrinsic: ModifySynchronization

Deleted By: Extrinsic: ModifySynchronization

Class Mandatory: Array|Virt

Table 170 describes class CIM_StorageSynchronized (Between StorageVolume elements).

Table 170: SMI Referenced Properties/Methods for CIM_StorageSynchronized (Between StorageVolume elements)

Properties	Flags	Requirement	Description & Notes
WhenSynced	N	Mandatory	
SyncMaintained		Mandatory	
CopyType		Mandatory	
ReplicaType		Optional	
SyncState		Mandatory	
CopyPriority	M	Optional	
SyncedElement		Mandatory	
SystemElement		Mandatory	

9.8.21 CIM_SystemComponent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 171 describes class CIM_SystemComponent.

Table 171: SMI Referenced Properties/Methods for CIM_SystemComponent

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

STABLE

DEPRECATED

Clause 10: Disk Drive Subprofile

The functionality of the Disk Drive Subprofile has been subsumed by the Clause 11: Disk Drive Lite Subprofile.

The Disk Drive Subprofile is defined in section 7.3.3.4 of SMI-S 1.0.2.

DEPRECATED

STABLE**Clause 11: Disk Drive Lite Subprofile****11.1 Description**

The Disk Drive Lite subprofile is used to model disk drive devices. This subprofile assumes the drive is linked to a larger system (e.g. Array, SDE). The model supports asset information, health and status, and Physical information. The model also supports external links to Pool membership, extent mapping, backend port modeling, SCSI buss and address mapping, and physical containment in system packages. The subprofile also includes active management of an optional location indicator.

11.1.1 Base model

A disk drive is modeled as a single MediaAccessDevice (DiskDrive) That shall be linked to a single StorageExtent (representing the storage in the drive) by a MediaPresent association. The StorageExtent class represents the storage of the drive and contains its size. Other classes further refine the model. PhysicalPackage contains asset information for the device and shall be connected by a Realizes association. The model can optionally contain SoftwareIdentity that contains information about the firmware and is linked by a DeviceSoftware association.

Disk Drive Lite also has an optional set of classes to model the ports on the drive. These classes include LogicalPort and ProtocolEndpoint.

11.1.2 Associations to external classes

The Disk Drive subprofile ties into the rest of the system via a number of key associations.

- ConcreteComponent - Is used to associate the StorageExtent to the StoragePool that the disk is part of. Required when used with Block Services profile
- BasedOn - Is used to associate The StorageExtent exported by the Disk Drive to another (higher level) extent (or a Volume).
- Container - Is used to associate the physical package of the disk drive to the physical package of the system.
- SystemDevice - Is used to scope the Disk to the system containing it and is mandatory.
- ProtocolControllerAccessUnit - Is used to link the Disk to system port(s) it is accessed through.
- SCSIInitiatorTargetLogicalUnitPath or MemberOfCollection may be used with Initiator Port Profiles.
- MemberOfCollection - Is used with Storage Device Enclosure.

11.1.3 Active Management

The DiskDrive class has been enhanced by the addition of a property (LocationIndicator) to read or set the state of a location indicator. When read the property returns a value that can be used to determine of the indicator is support and it's value. When written the indicator's state is set.

Instance Diagram

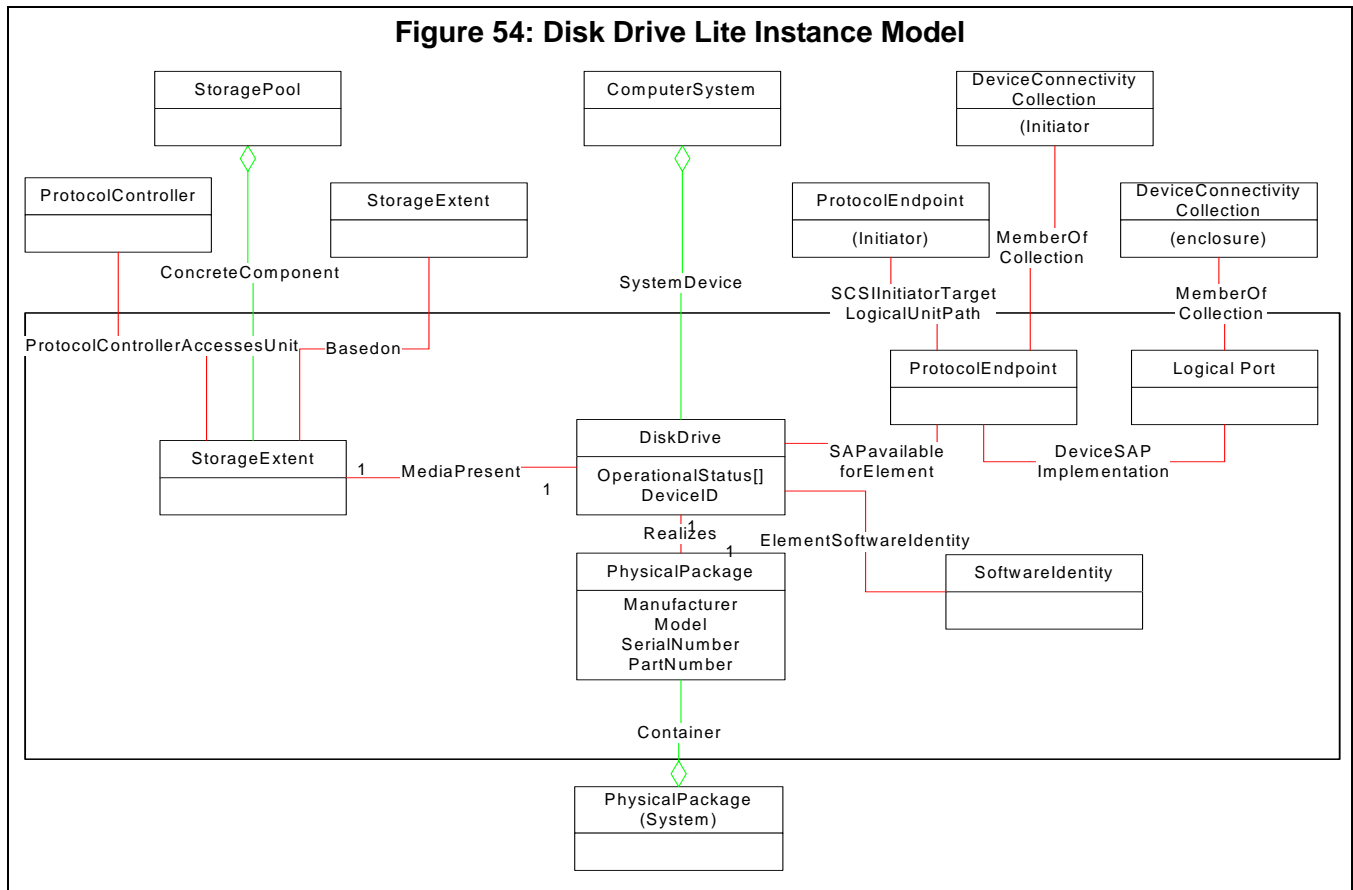


Figure 54 illustrates a sample instance diagram.

Durable Names and Correlatable IDs of the Profile

None.

Required External Associations

When implementing the Disk Drive Lite subprofile and the Block Services Package, the **ConcreteComponent** association to **StoragePools** is mandatory (because **LogicalStorage** is required in the Profile).

Optional External Associations

The **BasedOn** association that ties a Disk Drive extent to a higher level extent (or volume) is only required if the **ExtentMapping** subprofile is also implemented.

The **SCSIIInitiatorTargetLogicalUnitPath** or **MemberOfCollection** from **CIM_ProtocolEndPoint** may be used with Initiator Port Profiles.

The **MemberOfCollection** association from the **LogicalPort** is used with enclosure profiles.

11.2 Health and Fault Management Considerations

The DiskDrive.OperationalStatus contains the overall status of the disk, summarized in Table 172..

Table 172: OperationalStatus For DiskDrive

Primary Operational Status	Subsidiary Operational Status	Description
2 "OK"		Disk Drive is online.
6 "Error"		Disk Drive is no longer functioning.
8 "Starting"		Disk Drive is becoming enabled.
9 "Stopping"		Disk Drive is being disabled.
10 "Stopped"		Disk Drive is disabled.
Predictive Failure		Disk Drive is functionality nominally but is predicting a failure in the near future

11.3 Cascading Considerations

Not defined in this standard.

11.4 Supported Subprofiles and Packages

Not defined in this standard.

11.5 Methods of this Profile

Not defined in this standard.

11.6 Registered Name and Version

Disk Drive Lite version 1.2.0

11.7 CIM Elements

Table 173: CIM Elements for Disk Drive Lite

Element Name	Requirement	Description
CIM_ConcreteComponent (11.7.1)	Optional	
CIM_Container (11.7.2)	Optional	
CIM_DeviceSAPImplementation (11.7.3)	Optional	
CIM_DiskDrive (11.7.4)	Mandatory	
CIM_ElementSoftwareIdentity (11.7.5)	Mandatory	
CIM_LogicalPort (11.7.6)	Optional	
CIM_MediaPresent (11.7.7)	Mandatory	
CIM_ProtocolControllerAccessesUnit (11.7.8)	Optional	
CIM_ProtocolEndpoint (11.7.9)	Optional	
CIM_PhysicalPackage (11.7.10)	Mandatory	
CIM_Realizes (11.7.11)	Mandatory	
CIM_SAPAvailableForElement (11.7.12)	Optional	
CIM_SCSIInitiatorTargetLogicalUnitPath (11.7.13)	Optional	
CIM_SoftwareIdentity (11.7.14)	Mandatory	
CIM_StorageExtent (11.7.15)	Mandatory	
CIM_SystemDevice (11.7.16)	Mandatory	

11.7.1 CIM_ConcreteComponent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 174 describes class CIM_ConcreteComponent.

Table 174: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

11.7.2 CIM_Container

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 175 describes class CIM_Container.

Table 175: SMI Referenced Properties/Methods for CIM_Container

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

11.7.3 CIM_DeviceSAPImplementation

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 176 describes class CIM_DeviceSAPImplementation.

Table 176: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

11.7.4 CIM_DiskDrive

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 177 describes class CIM_DiskDrive.

Table 177: SMI Referenced Properties/Methods for CIM_DiskDrive

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
Name		Mandatory	
OperationalStatus		Mandatory	

11.7.5 CIM_ElementSoftwareIdentity

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 178 describes class CIM_ElementSoftwareIdentity.

Table 178: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

11.7.6 CIM_LogicalPort

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

11.7.7 CIM_MediaPresent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 179 describes class CIM_MediaPresent.

Table 179: SMI Referenced Properties/Methods for CIM_MediaPresent

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

11.7.8 CIM_ProtocolControllerAccessesUnit

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 180 describes class CIM_ProtocolControllerAccessesUnit.

Table 180: SMI Referenced Properties/Methods for CIM_ProtocolControllerAccessesUnit

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

11.7.9 CIM_ProtocolEndpoint

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

11.7.10 CIM_PhysicalPackage

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 181 describes class CIM_PhysicalPackage.

Table 181: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
Tag		Mandatory	
Manufacturer		Mandatory	
Model		Mandatory	
SerialNumber		Optional	
PartNumber		Optional	

11.7.11 CIM_Realizes

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 182 describes class CIM_Realizes.

Table 182: SMI Referenced Properties/Methods for CIM_Realizes

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

11.7.12 CIM_SAPAvailableForElement

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 183 describes class CIM_SAPAvailableForElement.

Table 183: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Properties	Flags	Requirement	Description & Notes
AvailableSAP		Mandatory	
ManagedElement		Mandatory	

11.7.13 CIM_SCSIInitiatorTargetLogicalUnitPath

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 184 describes class CIM_SCSIInitiatorTargetLogicalUnitPath.

Table 184: SMI Referenced Properties/Methods for CIM_SCSIInitiatorTargetLogicalUnitPath

Properties	Flags	Requirement	Description & Notes
Initiator		Mandatory	
Target		Mandatory	
LogicalUnit		Mandatory	Shall reference the -StorageExtent associated to the DiskDrive.

11.7.14 CIM_SoftwareIdentity

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 185 describes class CIM_SoftwareIdentity.

Table 185: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
VersionString		Mandatory	
Manufacturer		Optional	

Table 185: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Properties	Flags	Requirement	Description & Notes
BuildNumber		Optional	
MajorVersion		Optional	
RevisionNumber		Optional	
MinorVersion		Optional	

11.7.15 CIM_StorageExtent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 186 describes class CIM_StorageExtent.

Table 186: SMI Referenced Properties/Methods for CIM_StorageExtent

Properties	Flags	Requirement	Description & Notes
SystemCreationClass Name		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks as reported by the hardware.
ConsumableBlocks		Mandatory	The number of usable blocks.
Primordial		Mandatory	Shall be true.
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

11.7.16 CIM_SystemDevice

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 187 describes class CIM_SystemDevice.

Table 187: SMI Referenced Properties/Methods for CIM_SystemDevice

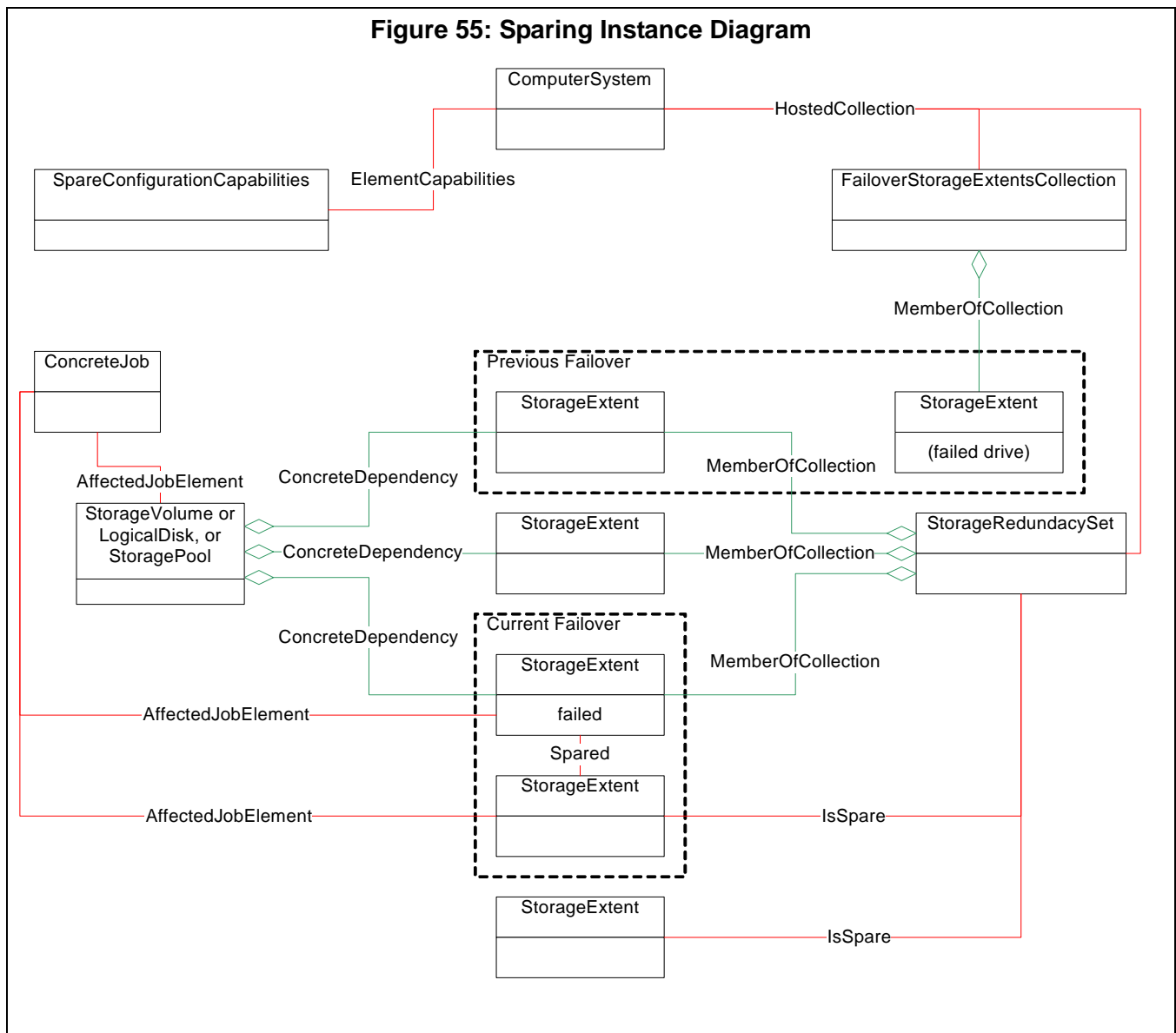
Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

STABLE

IMPLEMENTED**Clause 12: Disk Sparing Subprofile****12.1 Description**

Many block service systems enhance availability by providing backup storage capacity to be used in place of a failed component. The failure of the component may be caused by the failure of a physical component that realizes that component or the invalidation or corruption of the component itself.

The end result of the failure is that block server is degraded by performance or spare redundancy. In the first case, it is important that the cause of the performance degradation is known so the appropriate response may be taken. In the second case, the administrator will have to know of the loss of redundancy. The administrator can then plan to replace the used redundancy and fix the broken component. A sample instance diagram is provided in Figure 55.



Clause 15:, "Extent Composition Subprofile" focuses on the mapping of storage to storage elements, StorageVolume and LogicalDisk. This subprofile enhances that picture by representing how spare physical storage components like disk drives or purely logical constructs like LUNs or even host partitions, can be used to provide redundancy for storage elements. The spare elements are represented as StorageExtents themselves.

Clause 11:, "Disk Drive Lite Subprofile" can be used to supplement this subprofile by explicitly listing the changes in operational status resulting from the failure of disks and the affect of this failure on the StorageVolumes or LogicalDisks they support. In conjunction with *Storage Management Technical Specification, Part 2 Common Profiles* Clause 28: Health Package and the RelatedElementCausingError association, a client can tell, unambiguously the effect and cause of the storage component failure.

Fail Over is the name of the process by which the capacity provided by one StorageExtent is replaced by that of the spare StorageExtent. The block contents of the original StorageExtent is copied to the replacement StorageExtent. During this process a ConcreteJob shall be created to represent this process and report the progress and status of the fail over.

The functionality provided by this subprofile includes:

- The representation of the current state of the spares whether they are not in use, are in use, or in transition from not in use to being put into service. All three of these states can be present at once.
- The detection of the addition of another spare element and whether the implementation requires client intervention to assign the spare element.
- Client initiated fail over. A client may cause the fail over process to start.
- Client initiated rebuild of Extent data.
- Client initiated check and rebuild of Extent parity.

12.1.1 Durable Names and Correlatable IDs of the Profile

The StorageVolumes are required to provide the correlatable ID, Name. See *Storage Management Technical Specification, Part 1 Common Architecture* 7.2.

12.1.2 Sparing Model

StorageExtents are used as the unit of redundancy in this model. StorageExtents can be said to be a grouping of capacity. For the question of what component of the system has failed, the StorageExtent should be realized by a DiskDrive or some of component to which the failure is meaningful. This model represents how the capacity is used in the protection of the data. Other models define how StorageExtents are realized by other components or devices.

A *spare* is, functionally, the union of the StorageExtent representation and the associated component representation that realizes the Extent. This subprofile uses this term in this union.

The sparing model provides for mechanisms to:

- Group StorageExtents that have failed.
- Group new StorageExtents that have been added to the Block Server, but not yet assigned to a Primordial StoragePool.
- Group spares that can be used to replace failed components. The group of spares may be shared across StorageVolumes, LogicalDisks, or StoragePools.
- Report what component is being spared or replaced by the spare
- Report the process of a fail over, sparing reconfiguration, storage extent rebuild, or parity check

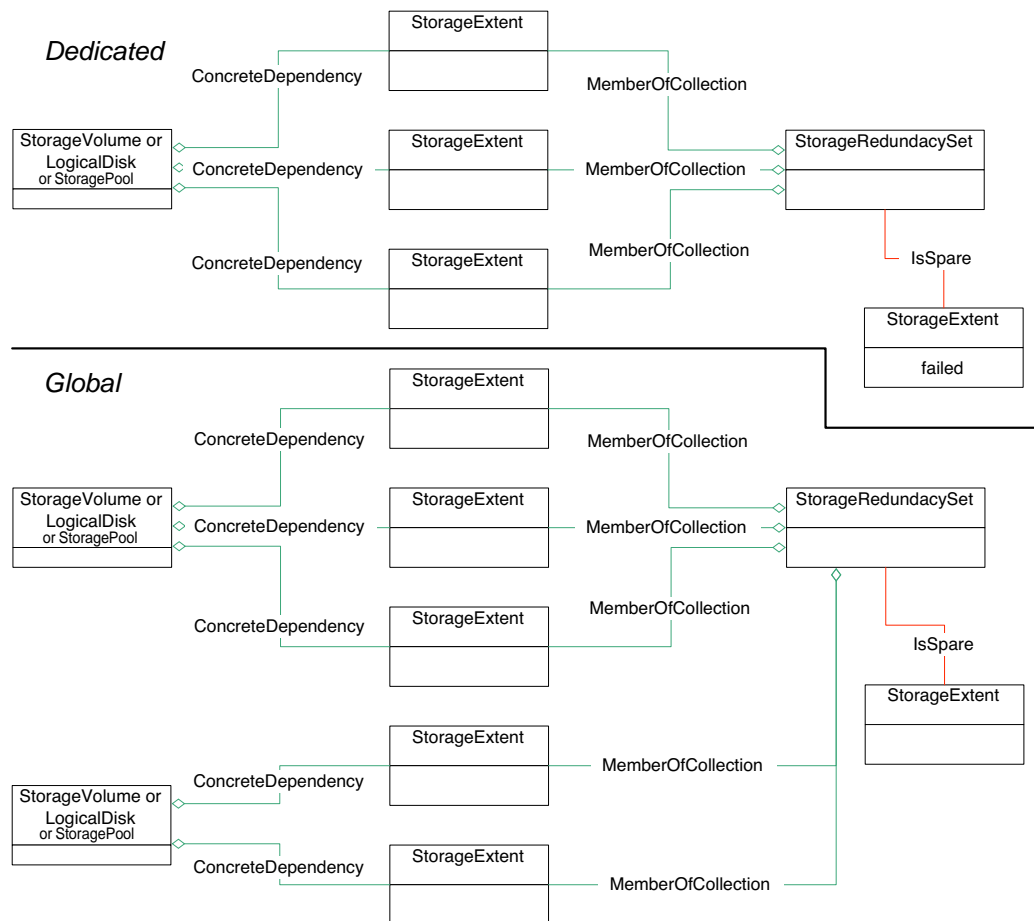
- Report the capabilities of the Sparing implementation

StorageVolume and LogicalDisk is used to represent that capacity exposed to clients of the Block Server.

StorageExtent is used to represent that capacity on which the StorageVolume or LogicalDisk is based. The StorageExtent is the representation of a component that can fail can potential cause data loss. It is likely that these StorageExtents will be primordial StorageExtents. These StorageExtents shall not be the representation of RAID stripes, but the StorageExtents may be copies of the data.

The StorageRedundancySet class is used to group spares. There may be a single StorageRedundancySet per StorageVolume or LogicalDisk. Multiple StorageVolumes or LogicalDisks may share a single StorageRedundancySet. In the first case, the spares grouping can be said to be *dedicated* to that StorageVolume or LogicalDisk. In the second case, the spares grouping can be said to be *global*; that is, the spares will be used for all the StorageVolumes or LogicalDisks that are associated to a StorageRedundancySet. This is illustrated in Figure 56.

Figure 56: Variations of RS per Storage Element



In the case where spares are not dedicated, the decision to group Extents with a given StorageRedundancySet depends of the rules of the implementation. Some implementations require particular types of spares to be used together. For example, some implementations may require that a DiskDrive is spared by another DiskDrive of the same size and/or type. If an implementation supports such rules then a StorageRedundancySet shall be created per rule. When StorageVolumes or LogicalDisk are created or modified, the implementation can select the StorageRedundancySet to associate to the created or modified storage element using on the PackageRedundancy

Goal. An implementation that supports *global* spares that supported both the Clause 5:, "Block Services Package" and this subprofile, would match this Goal with StorageRedundancySet that had at least that number of spares.

12.8.10 is used to collect the spares that have failed and the Extents that are not yet assigned to a Primordial StoragePool (PSP). These are the components that need to be diagnosed, repaired, and, possibly, replaced or assigned to the PSP.

12.8.12 is used to report the capabilities of the implementation. Not all sparing functionality is required. This class is used to report what optional functionality is implemented. The properties and methods of the class are defined later. Table 188 describes how to map the functionality names in the SupportedSynchronousActions and SupportedAsynchronousActions arrays.

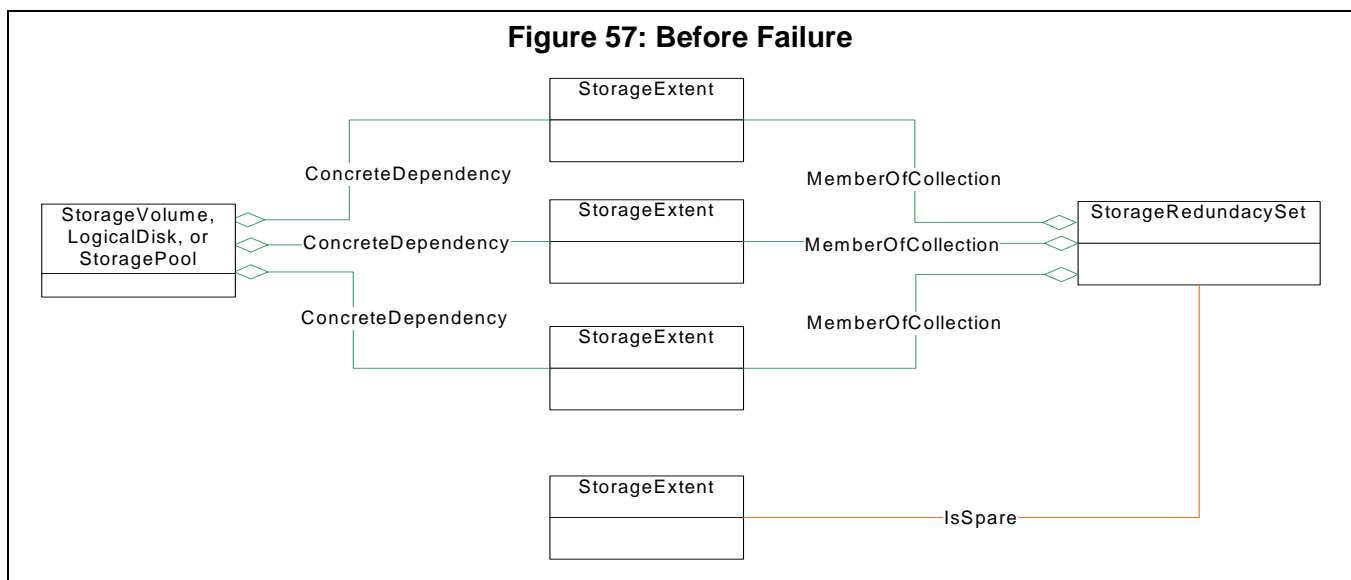
Table 188: Supported Methods to Method Mapping

Action	Method
Assign Spares	SpareConfigurationService.AssignSpares
Unassign Spares	SpareConfigurationService.UnassignSpares
Rebuild Storage Extent	SpareConfigurationService.RebuildStorageExtent
Check Parity Consistency	SpareConfigurationService.CheckParityConsistency
Repair Parity	SpareConfigurationService.RepairParity
Fail Over	StorageRedundancySet.Failover

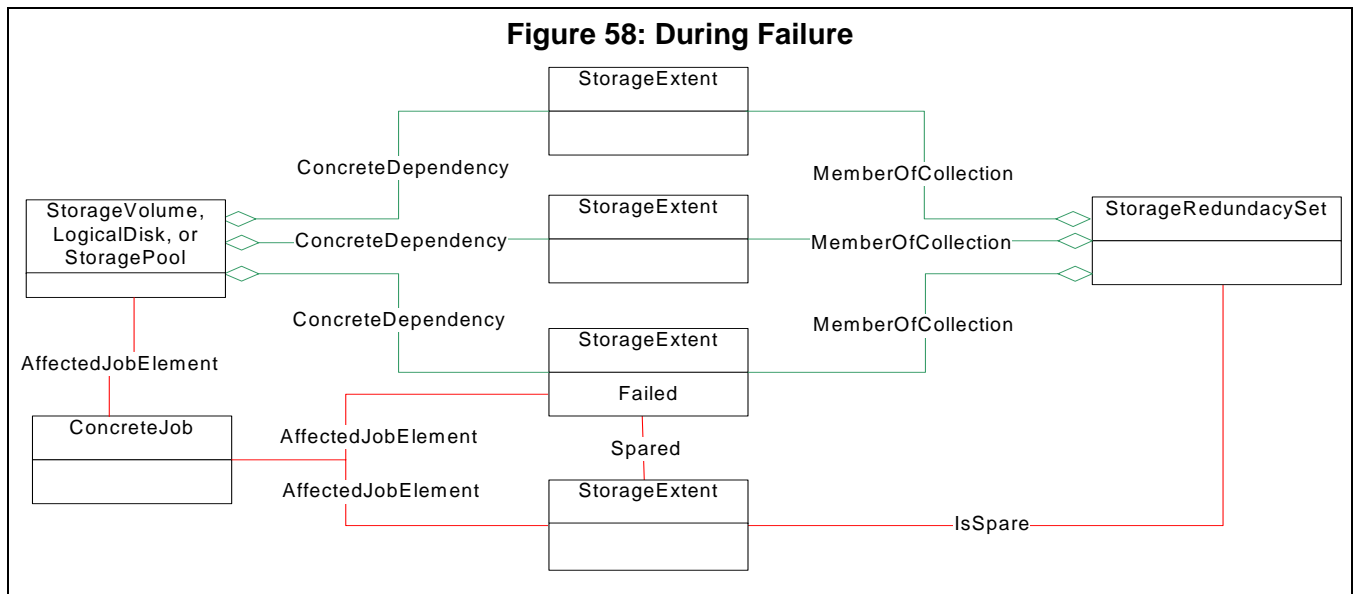
12.1.3 Modeling Fail Over, Past and Present

This section illustrates the requirements for modeling spare fail over in three cases, before the failure, during the fail over, and after the fail over.

Figure 57 shows a dedicated RedundancySet with a single spare.



Once the failure has occurred, a ConcreteJob is created to represent the fail over process, as shown in Figure 58.

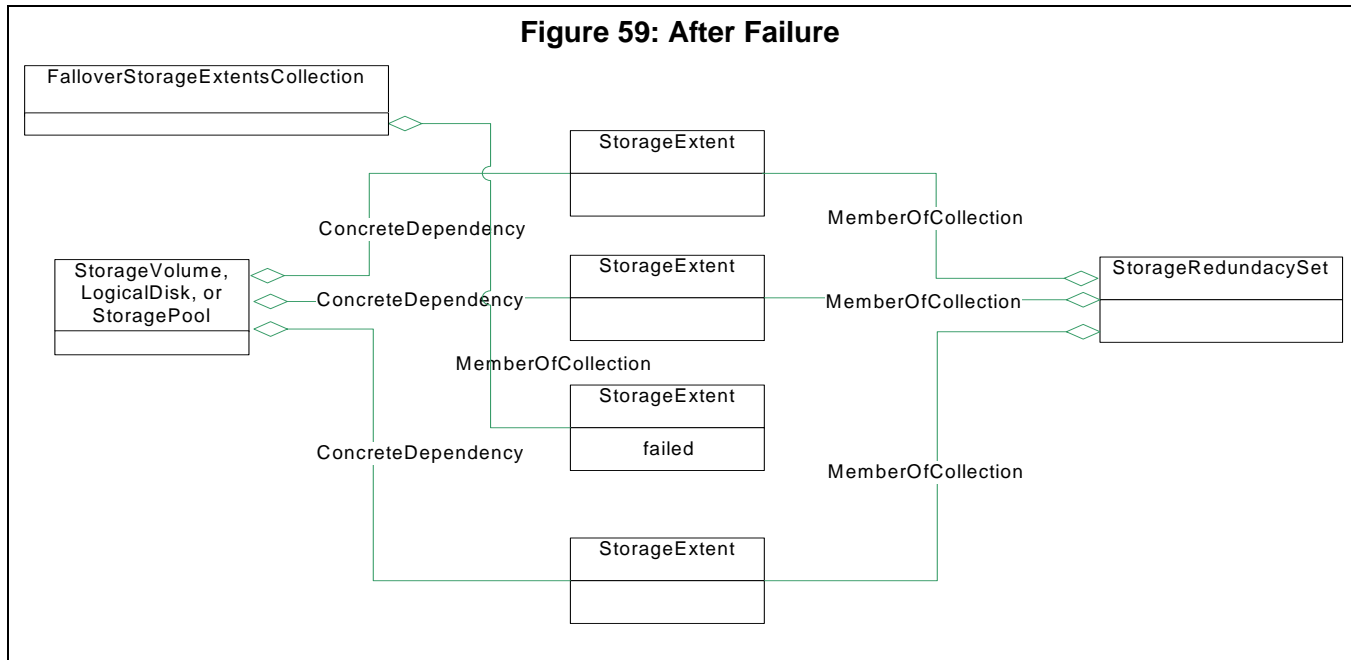


The AffectJobElement association shall associate the LogicalDisk or StorageVolume that is being failed over, the StorageExtent that has failed and is causing the fail over, and the spare StorageExtent. The associations shall remain for some period of time as per the rules in the *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30:; "Job Control Subprofile". For these rules consider the two extents as Input values to the `StorageRedundancySet.Failover()` method.

This subprofile supports fail over initiated by the implementation or by the client. So that an observer can tell what this fail over ConcreteJob is doing, the implementation shall model the ConcreteJob as if another client initiated the fail over, even though the implementation did the initiation. In other words, the ConcreteJob shall be associated to the StorageRedundancySet associated to the two Extents in question via the OwningJobElement association. The MethodResult instance, as defined in *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30:; "Job Control Subprofile", shall contain the `StorageRedundancySet.Failover()` method name and parameters.

Once the fail over is complete and some time has passed, the failed Extent shall no longer have a MemberOfCollection association to StorageVolume or LogicalDisk that was once based on it. The spare StorageExtent shall now participate in a MemberOfCollection associated to the StorageRedundancySet instead of the IsSpare association. The failed over Volume or LogicalDisk shall now participate in a ConcreteDependency

relationship with the spare Extent. The failed Extent shall now participate in a MemberOfCollection association with the FailoverStorageExtentsCollection, illustrated in Figure 59.



EXPERIMENTAL

12.1.4 Sparing Configuration and Control

All five methods defined or used in this subprofile, **AssignSpares**, **UnassignSpares**, **RebuildStorageExtent**, **CheckParityConsistency**, and **RepairParity** can be initiated by the implementation or the client. If the method execution is not instantaneous, then information about what method invocation gave rise to the job follows the rules in *Storage Management Technical Specification, Part 2 Common Profiles* Clause 30: "Job Control Subprofile". These methods can also be initiated by the implementation itself. The implementation shall represent the execution of the job, job name, and method parameters in said manner even it initiated the Job. If the implementation supports this functionality but does not allow the client to initiate the action, it shall still represent the execution of the functionality, as represented by a method execution, in said manner.

The purpose of these rules to allow an observer to tell that, for example, a **RepairParity** task is executing.

EXPERIMENTAL

12.2 Health and Fault Management Considerations

One of the primary reasons for this subprofile to allow a client to determine if the cause of performance degradation of a block server is caused by spare fail over, volume rebuild, or parity repair.

There are several failure cases possible with this subprofile:

- There may be failures of the several configuration and control methods of this subprofile for reasons other than the parameters provided by the client.

The **StorageExtents** used in the configuration and control methods may be invalid.

12.3 Cascading Conjunctions

Not defined in this standard.

12.4 Supported Subprofiles and Packages

Not defined in this standard.

12.5 Methods of the Profile

12.5.1 Assign Spares

```
uint32 AssignSpares(
    [Out] CIM_ConcreteJob REF Job
    [In] CIM_StoragePool REF InPool
    [In] CIM_StorageExtent REF InExtents[]
    [In] CIM_StorageRedundancySet REF RedundancySet)
```

This method is used to assign spares to a particular RedundancySet. If there is more than one primordial StoragePool in this implementation, then the arguments to the method shall contain the references to StorageExtents and references to the primordial StoragePools of which they are components. This method shall not permit the assignment of spare from more than one Primordial StoragePool. This method is more directed at an implementation whose spare Extents are Pyramidal Extents

This method may return the follow error codes. Many of the return codes are normal return codes; see 5.5.3 for a description of the unlisted return codes. The following documents the return codes unique to this method. This method shall not return vendor specific return codes.

```
ValueMap { "0", "1", "2", "3", "4", "5", "6", "..", "4096",
            "4097", "4098", "4099", "4101..32767", "32768..65535" },
Values { "Job Completed with No Error", "Not Supported",
          "Unknown", "Timeout", "Failed", "Invalid Parameter",
          "In Use", "DMTF Reserved",
          "Method Parameters Checked - Job Started",
          "Multiple Primordial StoragePools",
          "Spares Are Not Compatible",
          "StorageExtent is in use",
          "Method Reserved", "Vendor Specific" }
```

- 4097, "Multiple Primordial StoragePools", means the client passed Extents that are components of more than one Primordial StoragePool.
- 4098, "Spares Are Not Compatible", means the client pass Extents that may not be used together. There is no mechanism at this time to tell a client, through the model, what spares can be used together.
- 4099, "StorageExtent is in use", means that one or more of the Extents passed are already in use as a spare or as part of a StorageVolume or LogicalDisk.

12.5.2 UnassignSpares

```
uint32 UnassignSpares(
    [Out] CIM_ConcreteJob REF Job
    [In] CIM_StoragePool REF InPool
    [In] CIM_StorageExtent REF InExtents[])
```

This method is used to remove a spare from a StorageRedundancy and also unassign that Extent as a spare. The unassigned spare shall end up as a member of the FailoverStorageExtentsCollection. The rules for the

parameters and the same descriptions of AssignSpared are true for the parameters and return codes shared between the two method definitions. This method shall not return vendor specific return codes.

12.5.3 RebuildStorageExtent

```
uint32 RebuildStorageExtent(
    [Out] CIM_ConcreteJob REF Job
    [In] CIM_StorageExtent REF Target)
```

This method is used to rebuild the data distribution on the passed Extent with the other member Extents associated to a single StorageRedundancySet. If the Job execution fails, then use ConcreteJob.GetError() to get the CIM_Error that states what the error was. In this case, the Target Extent shall report the appropriate, non "OK", OperationalStatus.

The method may return the following error codes. Many of the return codes a normal return codes; see 5.5.3 for a description of the unlisted return codes. The following documents the return codes unique to this method. This method shall not return vendor specific return codes.

```
ValueMap { "0", "1", "2", "3", "4", "5", "6", "..", "4096",
            "4097", "4098", "4101..32767", "32768..65535" },
Values { "Job Completed with No Error", "Not Supported",
          "Unknown", "Timeout", "Failed", "Invalid Parameter",
          "In Use", "DMTF Reserved",
          "Method Parameters Checked - Job Started",
          "Target is Not a Member of a StorageRedundancySet",
          "Rebuild already in Progress",
          "Method Reserved", "Vendor Specific" }
```

- 4097 "Target is Not a Member of a StorageRedundancySet", means that the Extent passed is not a member of StorageRedundancySet
- 4098 "Rebuild already in Progress", means that a rebuild of the data and/or parity on the passed Extent or one or more of the other member Extents of the same StorageRedundancySet is already in progress.

12.5.4 CheckParityConsistency

```
uint32 CheckParityConsistency(
    [Out] CIM_ConcreteJob REF Job
    [In] CIM_StorageExtent REF Target)
```

This method is used to check of the parity distribution on the passed Extent with the other member Extents associated to a single StorageRedundancySet. If the Job execution fails, then use ConcreteJob.GetError() to get the Error that states what the error was. In this case, the Target Extent shall report the appropriate, non "OK", OperationalStatus. If method execution determines that the parity is inconsistent, the ConcreteJob shall report successful completion and one of Operational Statuses of the passed Extent shall be 6 "Error".

The method may return the following error codes. Many of the return codes a normal return codes; see 5.5.3 for a description of the unlisted return codes. The following documents the return codes unique to this method. This method shall not return vendor specific return codes.

```
ValueMap { "0", "1", "2", "3", "4", "5", "6", "..", "4096",
            "4097", "4098", "4099..32767", "32768..65535" },
Values { "Job Completed with No Error", "Not Supported",
          "Unknown", "Timeout", "Failed", "Invalid Parameter",
          "In Use", "DMTF Reserved",
          "Method Parameters Checked - Job Started",
          "Consistency Check Already in Progress",
          "No Parity to Check",
```



```
"Method Reserved", "Vendor Specific" }
```

- 4097 "Consistency Check Already in Progress", means that a check and rebuild of the data parity on the passed Extent or one or more of the other member Extents of the same StorageRedundancySet is already in progress.
- 4098 "No Parity to Check", means that the member Extents of the StorageRedundancySet are not build with parity distribution. Recheck the Virtualization modeled.

12.5.5 RepairParity

```
uint32 RepairParity(
    [In] CIM_ConcreteJob REF Job,
    [Out] CIM_StorageExtent REF Target)
```

This method is used to rebuild of the parity distribution on the passed Extent with the other member Extents associated to a single StorageRedundancySet. The intent is that this method would be run after finding out that the CheckParityConsistency) reported that the Extent pair is inconsistent. If the Job execution fails, then use ConcreteJob.GetError() to get the Error that states what the error was. In this case, the Target Extents shall report the appropriate, non "OK", OperationalStatus and HealthState.

The method may return the following error codes. Many of the return codes are normal return codes; see 5.5.3 for a description of the unlisted return codes. There are no return codes that are unique to this method. This method shall not return vendor-specific return codes.

12.6 Client Considerations and Recipes

The sparing implementation may cause the sparing configuration changes (i.e. jobs start and run) on its own in response to other clients.

The number of StorageRedundancySets may change over time because the physical components, realizing the spare StorageExtent, like disk drives are added or remove from the block server. Additionally, purely logical realizations of the spare StorageExtent may change as well. The StorageRedundancySets themselves once empty may remain in the model, but be empty, or may be removed from the model entirely for this or other reasons.

The sparing implementation shall report the correct RedundancyStatus, either 'Unknown' 0, 'Redundant' 1, or 'Redundancy Lost' 2. See property description below for details.

12.6.1 Determine if spare model is constructed correctly

```
// DESCRIPTION
// The goal of this recipe is to verify that the Sparing model
// is correctly instantiated.
// This type of instance traversal would be used by a client
// to determine if a particular storage element has spare
// coverage.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.A reference to a storage element, either a StorageVolume,
// a LogicalDisk, or a StoragePool, is previously defined in the
// $StorageElement-> variable

$SparedExtents->[] =
    AssociatorNames($StorageElement->,
        "CIM_ConcreteDependency",
        "CIM_StorageExtent",
```

```

        "Dependent", "Antecedent")
for i in SparedExtents->[] {
    #RedundancySets->[] =
        AssociatorNames($SparedExtents->[#i],
            "CIM_MemberOfCollection",
            "CIM_StorageRedundancySet",
            "Member", "Collection")
    // We should find at least one RS per spared SE
    if(1 > #RedundancySets.length) {
        <ERROR! There should be at least one RedundancySet per spared
            StorageExtent>
    }
    for j in RedundancySets->[] {
        #SpareSEs->[] =
            AssociatorNames($RedundancySets->[#j],
                "CIM_IsSpare",
                "CIM_StorageExtent",
                "Dependent", "Antecedent") // SRE has the Dependent role
        if (0 < #SpareSEs->[]) {
            <EXIT: Successfully found at least one spare StorageExtent
        }
        else {
            <ERROR! The SRE associated to the subject StorageElement
                must have at least one Spare>
        }
    }
}
<ERROR! At least one Spared Extent MUST have been found.
If one or more was found, an successful exit would have occurred
before this point in the code.>

```

12.7 Registered Name and Version

Disk Sparing version 1.3.0

12.8 CIM Elements

Table 189: CIM Elements for Disk Sparing

Element Name	Requirement	Description
CIM_StorageRedundancySet (12.8.1)	Mandatory	
CIM_IsSpare (12.8.2)	Mandatory	Represents the spare that may be used as a spare for any StorageExtents that is not a spare.
CIM_Spared (12.8.3)	Mandatory	Represents the relationship between the spare and the StorageExtent that has failed and is being spared
CIM_MemberOfCollection (12.8.4)	Mandatory	
CIM_StorageExtent (12.8.5)	Mandatory	
CIM_StorageVolume (12.8.6)	Conditional	Commonly known as a LUN but without the semantics of mapping to a host (which is covered by Masking and Mapping).
CIM_StoragePool (12.8.7)	Mandatory	Elements to Primordial and Concrete Pools.
CIM_ConcreteDependency (Extent to StorageVolume) (12.8.8)	Conditional	
CIM_ConcreteDependency (Extent to LogicalDisk) (12.8.9)	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory.
CIM_ConcreteDependency (Extent to Pool) (12.8.10)	Mandatory	
CIM_LogicalDisk (12.8.11)	Conditional	Conditional requirement: Referenced from Volume Management - LogicalDisk is mandatory.
SNIA_FailoverStorageExtentsCollection (12.8.12)	Optional	The collection of StorageExtents that have failed or are not yet assigned to a Primordial StoragePool.
CIM_HostedCollection (12.8.13)	Optional	Associates FailoverStorageExtentsCollection with the Block Server's ComputerSystem instance.
CIM_HostedCollection (12.8.14)	Mandatory	Associates StorageRedundancySet with the Block Server's ComputerSystem instance.
SNIA_SpareConfigurationCapabilities (12.8.15)	Optional	Instances of this class define the behavior supported by this sparing implementation.
SNIA_SpareConfigurationService (12.8.16)	Optional	Control the spares configuration and control of StorageExtent data and parity consistency.
CIM_ElementCapabilities (12.8.17)	Optional	Associates SpareConfigurationCapabilities with the Block Server's ComputerSystem instance.

Table 189: CIM Elements for Disk Sparing

Element Name	Requirement	Description
CIM_HostedCollection (12.8.18)	Optional	Associates SpareConfigurationService with the Block Server's ComputerSystem instance.

12.8.1 CIM_StorageRedundancySet

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 190 describes class CIM_StorageRedundancySet.

Table 190: SMI Referenced Properties/Methods for CIM_StorageRedundancySet

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
RedundancyStatus		Mandatory	The redundancy status shall be either 'Unknown' 0, 'Redundant' 2, or 'Redundancy Lost' 3. The implementation should report 2 or 3 most of the time, although it may report 0 sometimes. It should report 2 when there is at least one spare per the StorageRedundancySet. It should report 3 when there are no more spares (via IsSpare association) per the StorageRedundancySet.
TypeOfSet		Mandatory	'Limited Sparing', 5, is the type of sparing supported in the subprofile
MinNumberNeeded		Mandatory	
MaxNumberSupported		Mandatory	
Failover()		Mandatory	For block servers that do not do automatically fail over failed components, this method is used to cause the fail over to occur. More commonly, block server implementations automatically maintain the availability of their capacity. In this case, the method would only be used to cause fail back to occur, if that also does not occur automatically.

12.8.2 CIM_IsSpare

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 191 describes class CIM_IsSpare.

Table 191: SMI Referenced Properties/Methods for CIM_IsSpare

Properties	Flags	Requirement	Description & Notes
SpareStatus		Mandatory	
FailoverSupported		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

12.8.3 CIM_Spared

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 192 describes class CIM_Spared.

Table 192: SMI Referenced Properties/Methods for CIM_Spared

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

12.8.4 CIM_MemberOfCollection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 193 describes class CIM_MemberOfCollection.

Table 193: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Properties	Flags	Requirement	Description & Notes
Member		Mandatory	
Collection		Mandatory	

12.8.5 CIM_StorageExtent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 194 describes class CIM_StorageExtent.

Table 194: SMI Referenced Properties/Methods for CIM_StorageExtent

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
HealthState		Mandatory	Reports the state of the StorageExtents underlying component.
OperationalStatus		Mandatory	Reports the operational status of the StorageExtent.

12.8.6 CIM_StorageVolume

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Array|Virt

Table 195 describes class CIM_StorageVolume.

Table 195: SMI Referenced Properties/Methods for CIM_StorageVolume

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Optional	User friendly name
Name		Mandatory	VPD 83 identifier for this volume (ideally a LUN WWN)
NameFormat		Mandatory	Format for name
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

12.8.7 CIM_StoragePool

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 196 describes class CIM_StoragePool.

Table 196: SMI Referenced Properties/Methods for CIM_StoragePool

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
PoolID		Mandatory	A unique name in the context of this system that identifies this Pool.

12.8.8 CIM_ConcreteDependency (Extent to StorageVolume)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Array|Virt

Table 197 describes class CIM_ConcreteDependency (Extent to StorageVolume).

Table 197: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to StorageVolume)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

12.8.9 CIM_ConcreteDependency (Extent to LogicalDisk)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LVM

Table 198 describes class CIM_ConcreteDependency (Extent to LogicalDisk).

Table 198: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to LogicalDisk)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

12.8.10 CIM_ConcreteDependency (Extent to Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 199 describes class CIM_ConcreteDependency (Extent to Pool).

Table 199: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to Pool)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	

Table 199: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Extent to Pool)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	

12.8.11 CIM_LogicalDisk

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: LVM

Table 200 describes class CIM_LogicalDisk.

Table 200: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
SystemCreationClass Name		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Optional	User friendly name
Name		Mandatory	OS Device Name
NameFormat		Mandatory	Format for name
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks that make of this LogicalDisk.
IsBasedOnUnderlyin gRedundancy		Mandatory	
NoSinglePointOfFailu re		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	

12.8.12 SNIA_FailoverStorageExtentsCollection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

12.8.13 CIM_HostedCollection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 201 describes class CIM_HostedCollection.

Table 201: SMI Referenced Properties/Methods for CIM_HostedCollection

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	Indicates which FailoverStorageExtentsCollections are part of Disk Sparing implementation.

12.8.14 CIM_HostedCollection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 202 describes class CIM_HostedCollection.

Table 202: SMI Referenced Properties/Methods for CIM_HostedCollection

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	Indicate which StorageRedundancySets are part of Disk Sparing implementation.

12.8.15 SNIA_SpareConfigurationCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 203 describes class SNIA_SpareConfigurationCapabilities.

Table 203: SMI Referenced Properties/Methods for SNIA_SpareConfigurationCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedAsynchronousActions	N	Mandatory	Enumeration indicating what operations will be executed as asynchronous jobs. If an operation is included in both this and SupportedSynchronousActions then the underlying implementation is indicating that it may or may not create
SupportedSynchronousActions	N	Mandatory	Enumeration indicating what operations will be executed without the creation of a job. If an operation is included in both this and SupportedAsynchronousActions then the underlying instrumentation is indicating that it may or may not create a job.
SystemConfiguredSpares		Mandatory	Set to true if this storage system automatically configures spares. If set to false, the client must use the extrinsic methods AssignSpares and UnassignSpares.
AutomaticFailOver		Mandatory	Set to true if this storage system automatically fails over. If set to false, the client must use the FailOver extrinsic method, although that method may not be supported.
MaximumSpareStorageExtents		Mandatory	States the maximum number of StorageExtents that can be configured as spares for the entire block server. A 0 means that all primordial StorageExtents can be configured as spares.

12.8.16 SNIA_SpareConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 204 describes class SNIA_SpareConfigurationService.

Table 204: SMI Referenced Properties/Methods for SNIA_SpareConfigurationService

Properties	Flags	Requirement	Description & Notes
AssignSpares()		Mandatory	
UnassignSpares()		Mandatory	
RebuildStorageExtent()		Mandatory	
CheckParityConsistency()		Mandatory	
RepairParity()		Mandatory	

12.8.17 CIM_ElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 205 describes class CIM_ElementCapabilities.

Table 205: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The hosting System.
Capabilities		Mandatory	The support spare configuration capabilities.

12.8.18 CIM_HostedCollection

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 206 describes class CIM_HostedCollection.

Table 206: SMI Referenced Properties/Methods for CIM_HostedCollection

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The hosting System.
Dependent		Mandatory	The support spare configuration service.

IMPLEMENTED

EXPERIMENTAL

Clause 13: Erasure Profile

13.1 Description

The Erasure profile describes how data on a storage element (StorageVolume or LogicalDisk) may be erased. As data is replicated, migrated and archived throughout its lifecycle, there is a need to ensure that residual and superseded copies or versions of the data that remain on storage media are destroyed in line with business policies for privacy, confidentiality and security.

Erasure will be required whenever it is deemed that the data on a volume is sufficiently sensitive or of competitive value that the media cannot be reused, redeployed or made redundant without ensuring that the data is destroyed.

As part of the data lifecycle, data will potentially be replicated and migrated several times throughout their life before final destruction, as a result of media and technology change or management policies.

Common situations would include:

- Migration to secondary or tertiary archive storage followed by deletion of the source data
- Movement of data from a failing device to a spare.
- Migration and cut-over to new target media, retaining the source media for a "fall back" for some period then reuse (or resale) of the source media.

13.1.1 Existing Erasure standards

There are numerous erasure standards in the industry. These techniques generally involve writing a bit pattern to the storage media and in most cases require multiple passes of overwriting of these bit patterns. The following is an incomplete list of erasure techniques to illustrate the variety that exists today.

- HMG Infosec Standard 5, The Baseline Standard.
- HMG Infosec Standard 5, The Enhanced Standard.
- Peter Gutmann's algorithm.
- U.S.Department of Defense Sanitizing (DOD 5220.22-M)
- Bruce Schneier's algorithm.
- Navy Staff Office Publication (NAVSO P-5239-26) for RLL.
- The National Computer Security Center (NCSC-TG-025).
- Air Force System Security Instruction 5020.
- US Army AR380-19.
- German Standard VSIT
- OPNAVINST 5239.1A.

Because there is such a wide variety of techniques, this subprofile does not dictate which technique shall be used. The instrumentation shall tell the client which methods are supported. Since erasure of data on a volume may be a lengthy process and will most likely be a background task, the volume may provide the status of the erasure and may provide notification via an Indication of the erasure completion.

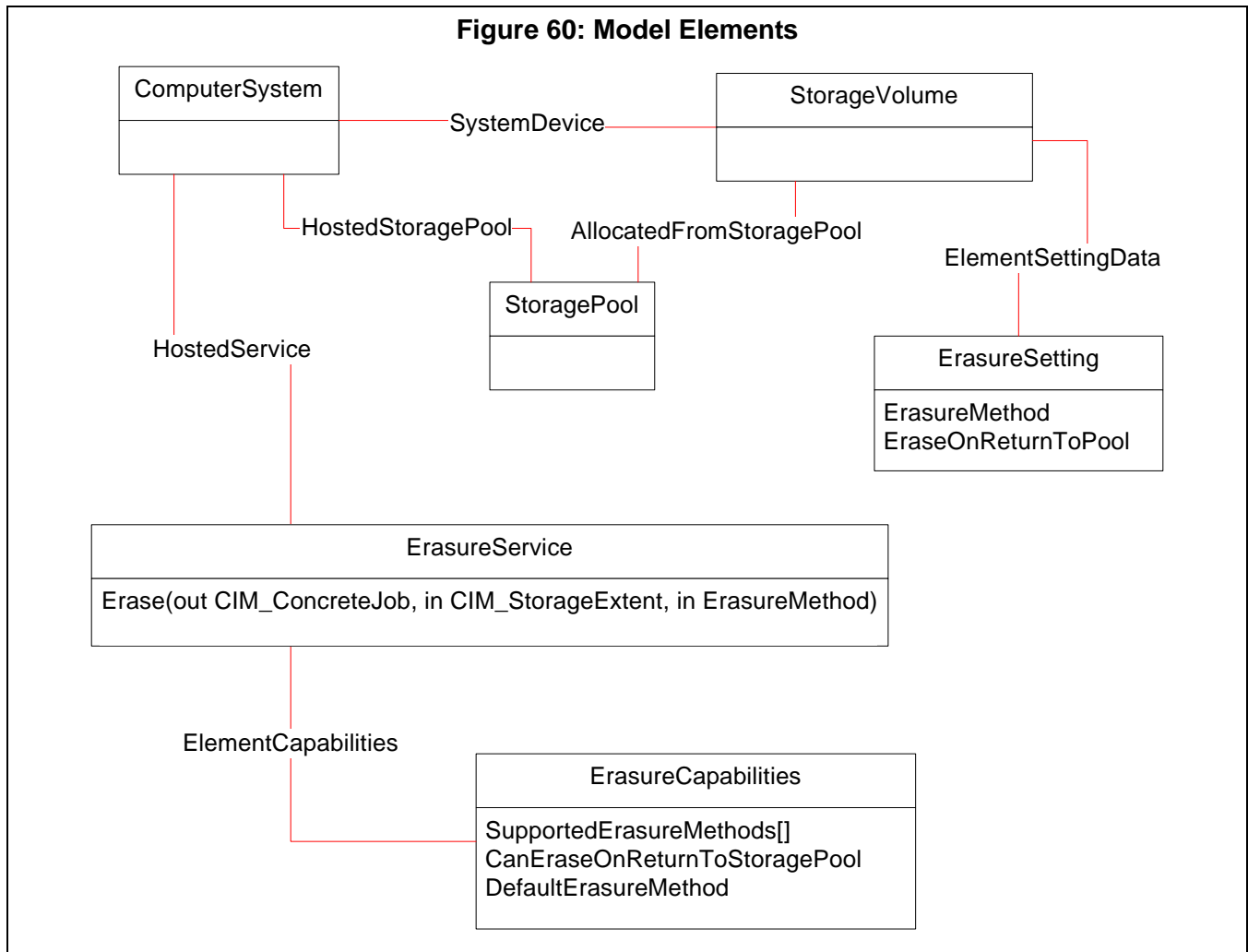
To support this profile, instrumentation shall provide a list of supported erasure methods in the `ErasureCapabilities.SupportedErasureMethods` property. If the instrumentation supports erasing a volume upon return to a storage pool, then the `ErasureCapabilities.CanEraseOnReturnToStoragePool` property shall be set to true. If the instrumentation does not support this capability, then the value shall be false (the default value). The `ErasureCapabilities` shall be associated to the `ErasureService` via the `ElementCapabilities` association.

If `CanEraseOnReturnToStoragePool` is true, then the `ErasureCapabilities.DefaultErasureMethod` shall be used to erase the element, unless the `ErasureSetting.ErasureMethod` is non-NULL. The instrumentation may provide a default value for this properties. A client may be able to change the `ErasureCapabilities.DefaultErasureMethod` and `ErasureSetting.ErasureMethod`.

To erase the volume explicitly, the user shall call the `ErasureService.Erase` method, passing in the volume to erase and the erasure method to use. The erasure method shall be one of the erasure methods the instrumentation supports. A NULL may be passed in as the `ErasureMethod`, in which case, the instrumentation shall use the `DefaultErasureMethod` from the capabilities as the erasure method. To erase a volume implicitly, it is required that the `CanEraseOnReturnToStoragePool` shall be true and that the `ErasureSetting` associated to the volume has the `EraseOnReturnToPool` value set to true. If these conditions are met, then when the user calls the `ReturnToStoragePool` method, the volume shall be erased before being returned to the pool.

If a `ConcreteJob` has been started as a result of the erasure (either from calling `Erase` or `ReturnToStoragePool`), then the `ConcreteJob` shall have an `AffectedJobElement` association to the `StorageVolume` being erased.

Table 60 shows the new properties and method introduced by this subprofile. While a `StorageVolume` is shown, the same shall apply to `LogicalDisk`.

Figure 60: Model Elements

13.2 Health and Fault Management Considerations

Not defined in this standard.

13.3 Cascading Considerations

Not applicable

13.4 Supported Profiles, Subprofiles, and Packages

Not defined in this standard.

13.5 Methods of the Profile

The Erase method in the ErasureService shall erase the contents of the volume using the specified erasure method. The erasure methods that the instrumentation supports shall be found in the ErasureCapabilities.SupportedErasureMethods property.

Table 207: Erase Method

Method: Erase			
Return Values:			
Value		Description	
0: Job completed		Job completed with no error	
1: Not supported		Method not supported	
2: Unspecified Error			
3: Timeout			
4: Failed		Refer to instance of CIM_Error	
5: Invalid parameter		Refer to instance of CIM_Error	
6: In Use			
7..4095		DMTF Reserved	
4096: Job started		REF returned to started ConcreteJob	
Errors:			
(status):registry:MessageID		ErrorName:MessageArguments	
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	CIM_ConcreteJob REF	Returned if job started.
IN, REQ	Extent	CIM_StorageExtent REF	Extent (volume) to erase
IN, REQ	Type	uint16	Type of extent (StorageVolume or LogicalDisk)
IN, REQ	ErasureMethod	uint32	Erasure method to use

13.6 Client Considerations and Recipes

These cases can be generalized into the explicit case of "Volume Erasure" and the implicit case of "Volume Deletion".

13.6.1 Recipe 1: Volume Erasure

This is the case where it is determined that the contents of a storage volume must be erased. This requires the client to call the ErasureService method Erase() to specify the StorageVolume and the ErasureMethod.

```
// DESCRIPTION:
//
// Erase a volume
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The ErasureService has been found and the object path
//    value is stored in $ErasureService->
// 2. The ErasureCapabilities associated to the
//    ErasureService has been found and the instance stored
```

Erasure Profile

```

//    in $ControllerCapabilities
// 3. The StorageVolume to use has been identified and the object path
//    values are stored in $Volume->
// 4. The erasure method to use has been determined and it's value
//    stored in #ErasureMethod

// Determine if there is a job created by method
// and wait for the job to complete
// Input:
//    #ReturnCode : The return code of the method
//    $ConcreteJob-> :The output parameter that may have a ConcreteJob REF.
// This method will return control if the recipe was not exited because of error
sub void WaitForJob(#ReturnCode, $ConcreteJob->) {
    if (4096 == #ReturnCode) {
        if ($ConcreteJob-> != null) {
            /*Wait until the completion of the job using $ConcreteJob-> as
            a filter Verify that the OperationalStatus contains 2 ("OK"),
            or 17 ("Completed") */
            $JobInstance = GetInstance($ConcreteJob->,
                false, false, false, null)
            if ($JobInstance.JobStatus != 7) { // 7 - Completed
                <ERROR! Job failed! >
            }
        } else {
            <ERROR! Missing Job reference>
        }
    }
}

// Step 1. Erase the volume

if ( (#ErasureMethod != NULL) &&
    (contains(#ErasureMethod,
        $ControllerCapabilities.SupportedErasureMethods[]) == false) ) {
    <ERROR! Invalid Erasure method>
}

%InputArguments["Extent"]      = { $Volume-> }
%InputArguments["Type"]        = 1 // StorageVolume
%InputArguments["ErasureMethod"] = #ErasureMethod

#ReturnCode = InvokeMethod($ErasureService->,
    "Erase",
    %InputArguments,
    %OutputArguments)
// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"

```

```

if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

$Job-> = %OutputArguments["Job"]
if ($Job-> != null) {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $Job->)
}

```

13.6.2 Recipe 2: Volume Deletion

This case is where a volume is being returned to the storage pool, and it needs to be erased. The client needs to check the `CanEraseOnReturnToStoragePool` property to see if this is possible. If it is, then the client looks for an `ErasureSetting` associated to the volume, creating one if necessary. The client sets the `ErasureSetting.EraseMethod` and `ErasureSetting.EraseOnReturnToStoragePool` for the setting associated to the volume, then calls `ReturnToStoragePool`.

```

// DESCRIPTION:
//
// Erase a volume as a byproduct of being deleted
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The ErasureService has been found and the object path
//    value is stored in $ErasureService->
// 2. The ErasureCapabilities associated to the
//    ErasureService has been found and the instance stored
//    in $ControllerCapabilities
// 3. The StorageConfigurationService has been found and the object path
//    value is stored in $StorageConfigService->
// 4. The StorageVolume to use has been identified and the object path
//    values are stored in $Volume->
// 5. The erasure method to use has been determined and it's value
//    stored in #ErasureMethod
//
// Determine if there is a job created by method
// and wait for the job to complete
// Input:
//   #ReturnCode : The return code of the method
//   $ConcreteJob-> :The output parameter that may have a ConcreteJob REF.
// This method will return control if the recipe was not exited because of error
sub void WaitForJob(#ReturnCode, $ConcreteJob->) {
    if (4096 == #ReturnCode) {
        if ($ConcreteJob-> != null) {
            /*Wait until the completion of the job using $ConcreteJob-> as
            a filter Verify that the OperationalStatus contains 2 ("OK"),

```

```

        or 17 ("Completed") */
        $JobInstance = GetInstance($ConcreteJob->,
            false, false, false, null)
        if ($JobInstance.JobStatus != 7) { // 7 - Completed
            <ERROR! Job failed! >
        }
    } else {
        <ERROR! Missing Job reference>
    }
}

// Step 1. Check capabilities

if ( $ControllerCapabilities.CanEraseOnReturnToStoragePool == false) {
    <ERROR! Implicit erasure not supported. Use Erase() method >
}

// Step 2. Find/create setting
$Setting[] = Associators($Volume->,
    "CIM_ElementSettingData",
    "SNIA_ErasureSetting",
    null, null,
    false, false, null)
if ($Setting[].length == 0) {
    // Create setting
    $TheSetting = newInstance("SNIA_ErasureSetting")
    $TheSetting.InstanceID = "SNIA:0001" // create unique ID
    $TheSetting.ErasureMethod = #ErasureMethod
    $TheSetting.EraseOnReturnToStoragePool = true
    $instance-> = CreateInstance($TheSetting)
}
else {
    $Setting[0].ErasureMethod = #ErasureMethod
    $Setting[0].EraseOnReturnToStoragePool = true
    ModifyInstance($Setting[0])
}

// Step 3 Delete the volume
%InArguments["TheElement"] = $Volume->

#ReturnCode = InvokeMethod($StorageService->,
    "ReturnToStoragePool",
    %InArguments,
    %OutArguments)

```

```
// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

$Job-> = %OutputArguments["Job"]
if ($Job-> != null) {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $Job->)
}
```

13.7 Registered Name and Version

Erasure version 1.2.0

13.8 CIM Elements

Table 208: CIM Elements for Erasure

Element Name	Requirement	Description
CIM_AllocatedFromStoragePool (13.8.1)	Mandatory	
SNIA_ErasureCapabilities (13.8.2)	Mandatory	
SNIA_ErasureService (13.8.3)	Mandatory	
CIM_StoragePool (13.8.4)	Mandatory	
CIM_StorageVolume (13.8.5)	Conditional	
CIM_LogicalDisk (13.8.6)	Conditional	
SNIA_ErasureSetting (13.8.7)	Mandatory	

13.8.1 CIM_AllocatedFromStoragePool

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 209 describes class CIM_AllocatedFromStoragePool.

Table 209: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

13.8.2 SNIA_ErasureCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 210 describes class SNIA_ErasureCapabilities.

Table 210: SMI Referenced Properties/Methods for SNIA_ErasureCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	User friendly name for this instance of Capabilities.
InstanceID		Mandatory	Unique identifier for the instance
ErasureMethods		Mandatory	Indicates erasure methods supported
DefaultErasureMethod		Mandatory	Erasure method to use if none specified in the volume's setting
CanEraseOnReturnToStoragePool		Mandatory	Indicates that the volume can be erased when deleted

13.8.3 SNIA_ErasureService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 211 describes class SNIA_ErasureService.

Table 211: SMI Referenced Properties/Methods for SNIA_ErasureService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The scoping System CreationClassName
SystemName		Mandatory	The scoping System Name
CreationClassName		Mandatory	The name of the concrete subclass
Name		Mandatory	Unique identifier for the Service
Erase()		Mandatory	This service contains the Erase method used to erase storage elements

13.8.4 CIM_StoragePool

Created By: External

Modified By: External

Deleted By: External

Class Mandatory:

Table 212 describes class CIM_StoragePool.

Table 212: SMI Referenced Properties/Methods for CIM_StoragePool

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	
TotalManagedSpace		Mandatory	
RemainingManagedSpace		Mandatory	

13.8.5 CIM_StorageVolume

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Conditional

Table 213 describes class CIM_StorageVolume.

Table 213: SMI Referenced Properties/Methods for CIM_StorageVolume

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

13.8.6 CIM_LogicalDisk

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Conditional

Table 214 describes class CIM_LogicalDisk.

Table 214: SMI Referenced Properties/Methods for CIM_LogicalDisk

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

13.8.7 SNIA_ErasureSetting

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 215 describes class SNIA_ErasureSetting.

Table 215: SMI Referenced Properties/Methods for SNIA_ErasureSetting

Properties	Flags	Requirement	Description & Notes
ErasureMethod		Mandatory	Erasure method to use. Must be one of the erasure methods supported by the instrumentation
EraseOnReturnToPool		Mandatory	Indicates if this volume should be erased when deleted. Default is false

EXPERIMENTAL

DEPRECATED

Clause 14: Extent Mapping Subprofile

The functionality of the Extent Mapping Subprofile (Section 7.3.3.5 of SMI-S 1.0.2) has been subsumed by the Extent Composition Subprofile (8.3.1.15).

DEPRECATED

STABLE**Clause 15: Extent Composition Subprofile****15.1 Description**

The Extent Composition Subprofile allows an implementation that supports the Block Services package to optionally provide an abstraction of how it virtualizes exposable block storage elements from the underlying Primordial storage pool. The abstraction is presented to the client as a representative hierarchy of extents. These extents are instances of `CompositeExtents` and `StorageExtents` linked by a combination of `CompositeExtentBasedOn` and `BasedOn` associations. The foundation of the hierarchy is a set of Primordial extents.

This subprofile is used optionally with the Array, Virtualization, Self-Contained NAS, NAS Head, and Volume Management profiles.

A Primordial storage extent can represent a Disk Drive in the Array or Self-contained NAS, a downstream virtualized Volume used by the Virtualizer or NAS Head profiles, or a OS Logical Disk in the Volume Management profile.

An exposable block storage element as used in this subprofile is defined as a Storage Volume or a Logical Disk.

In the presented hierarchy each extent (the dependent) is formed from those that it “precede” it (the antecedents) by a process of either decomposition or composition.

15.1.1 Decomposition

Decomposition is used to allocate space from an antecedent extent, in order to form a new dependent extent. This allocation may be partial or complete consumption. Complete consumption is the degenerate case in which all space in the antecedent extent is used. In this case the decomposed dependent extent may be either modeled even though it is one to one with the antecedent extent or omitted and the antecedent extent used in its stead.

15.1.2 Composition

Composition is used to form an a dependent extent from antecedent extents for the purpose of either concatenating the antecedent blocks to achieve a size goal, or to achieve a Quality Of Service goal such as mirroring the antecedent extents for redundancy, striping the antecedent extents for performance, or striping the antecedent extents with the addition of parity to achieve redundancy.

These extent “productions” can be assembled in a multi-layer hierarchy.

15.1.3 Model Element Summary

This subprofile uses the following CIM Classes:

`LogicalDisk` & `StorageVolume` - These are used to model the exposable block storage element.

`StorageExtent` - Used to represent the decomposition (partial allocation) of an Antecedent extent.

`CompositeExtent` - Used to represent the composition of several antecedent extents into a virtualized set of blocks with desired size and Quality-Of-Service.

`BasedOn` - Used to associate a Dependent and Antecedent extent in the subprofile hierarchy for both composition and decomposition. It is also used in one special case as a one-to-one (neither composing or decomposing), always associating the `StorageVolume` or `LogicalDisk` to the antecedent `CompositeExtent`. This is because, as a sibling of `StorageExtent` and `LogicalDisk`, `CompositeExtent` cannot be exposed directly.

CompositeExtentBasedOn - A subclass of **BasedOn** that is used in a composition production when the **Dependent** is a **CompositeExtent** which is describing striping; it contains **Stripe Depth** information. **Stripe Depth** is the number of blocks written to an **Antecedent** extent before moving on to the next extent. Although this property is on the association class, its values shall be the same for each instance of the association with the same **Dependent CompositeExtent**.

ConcreteComponent - Used to associate extents that are playing the **Pool Component** role to their parent **StoragePool** (See 15.1.4.2).

StoragePool and **AllocatedStoragePool** are shown in instance diagrams for context but are part of the **Block Service** package **Read Only** sub-package.

Please refer to the section “Required CIM Elements” for detailed class descriptions.

15.1.4 Relation to other Packages and Subprofiles

15.1.4.1 Block Services StoragePool hierarchy.

The **Block Services** package defines the model for the hierarchy of pools from the exposable storage element to the **Primordial Pool**. The hierarchy defined in this subprofile parallels that pool hierarchy and is layered so that the virtualization can be presented within the pool level in which it actually takes place.

15.1.4.2 Component Extents

Component Extents of a pool are the most dependent extents in the pool; they are also the only extents that are directly *manageable* by the methods in the **Block Services** Package. They are also the only extents that figure into the reconciliation of managed space in the pool (see 15.1.4.3).

Although a given implementation may choose a low level (i.e., detailed) or high-level presentation of how it virtualizes a storage element from a pool, or how space in a pool is itself virtualized, the **Pool Component** extents that are part of an exposable block storage element’s hierarchy shall be modeled along with their associations to the parent pool.

15.1.4.3 Block Services Extent Conservation

The **Block Services** package describes the concept of **Extent Conservation**, which describes the result of allocating storage from **Pool Component** extents using “**Remain Space Extents**”. These extents are not modeled by the **Extent Composition** subprofile, they are discoverable by the **GetAvailableExtents** method in **Block Services**.

15.1.4.4 Block Services Common RAID Levels

The **Block Services** Package describes a set of **RAID Levels** and in addition, properties on **StorageSetting** such as **ExtentStripeLength** and **UserDataStripeDepth** which allow creation of a subset of those **RAID** levels, using **CreateOrModifyElementFromElements**.

However, the **Extent Composition** subprofile is capable of describing general organizations, such as heterogeneous, multi-layer **RAID** such as can be create by the **Volume Management** profile. An example of this would be a **RAID 5** mirrored against a **RAID 0**, a **RAID (5,0)+1**. Another example would be a three layer **RAID** organization such as a **RAID 10** where the bottom layer **RAID 1** members were concatenations of available extents.

15.1.5 Scenarios

The following example scenarios are common abstractions of the use-cases that were used when this subprofile was being defined. The scenarios are not intended to cover all possible variations of the use of **Extent Composition**.

15.1.5.1 Volume Composition

Figure 61 shows extent composition when a single **RAID QOS/Level** is applied directly to the construction of a **StorageVolume**. The **Storage Volume** or **Logical Disk** and the underlying **CompositeExtent** represent the same

virtual extent and range of blocks; The initial BasedOn association between them is a one-to-one “dummy” association. The Storage Volume and Logical Disk classes do not have the necessary properties to describe the RAID information and the CompositeExtent which is a sibling class of StorageVolume and LogicalDisk, cannot be directly exposed. This Based on association does not represent composition or decomposition, but the main recipe (see 15.6.1) for this subprofile makes use of the decomposition function (i.e., complete consumption) to make this initial traversal.

Figure 61: Volume Composition from General QOS Pool

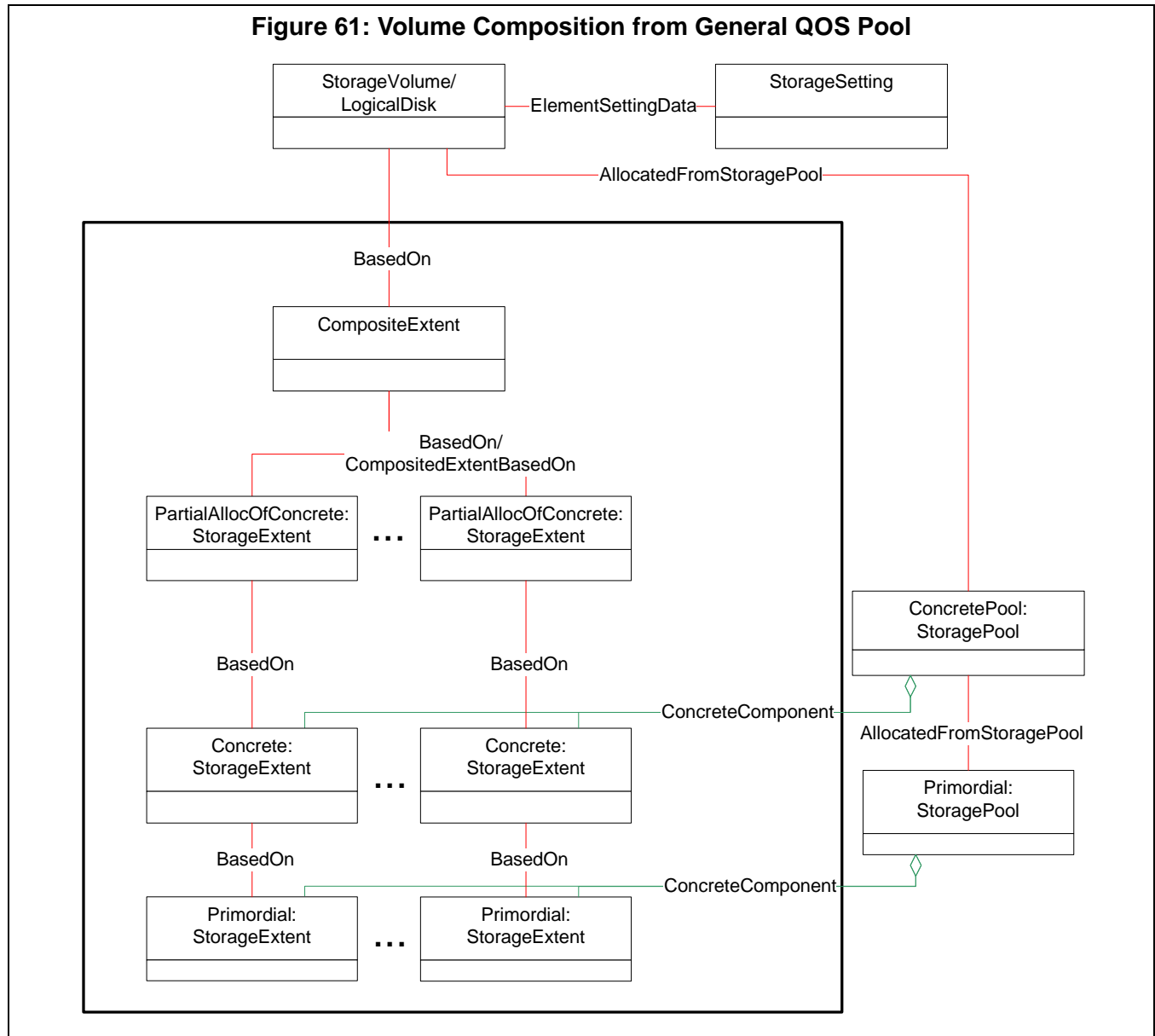


Figure 62 shows a single composition (such as a RAID 5 or RAID 1). Not shown is the scenario where there may be two or more such back to back productions (such as a RAID 10). Also not shown is the scenario where the two productions may be in different concrete pools in the hierarchy. A RAID 10 Volume may be constructed as a RAID 0 composition from a concrete pool that is itself a RAID 1 pool (see 15.1.5.2).

In this scenario, note that the extents below the StorageVolume and the Component Extents are not part of the pool, but allocated from it.

In fact this StorageVolume and its companion CompositeExtent could be composed from member extents (labeled PartialAllocOfConcrete in the diagram) from different pools.

15.1.5.2 Pool Composition

Certain pools can be created or modified to contain one or more extents each with a single specific quality of service. These extents are known as Raid Groups. The bound space in each of these RAID Groups is represented by this sub-profile as a single CompositeExtent at the top of an extent sub-hierarchy in that pool. Volumes created from this type of Pool are partially allocated (decomposed) from the CompositeExtent playing the role of the RAIDGroup.

Figure 62: Single QOS Pool Composition (RAID Groups)

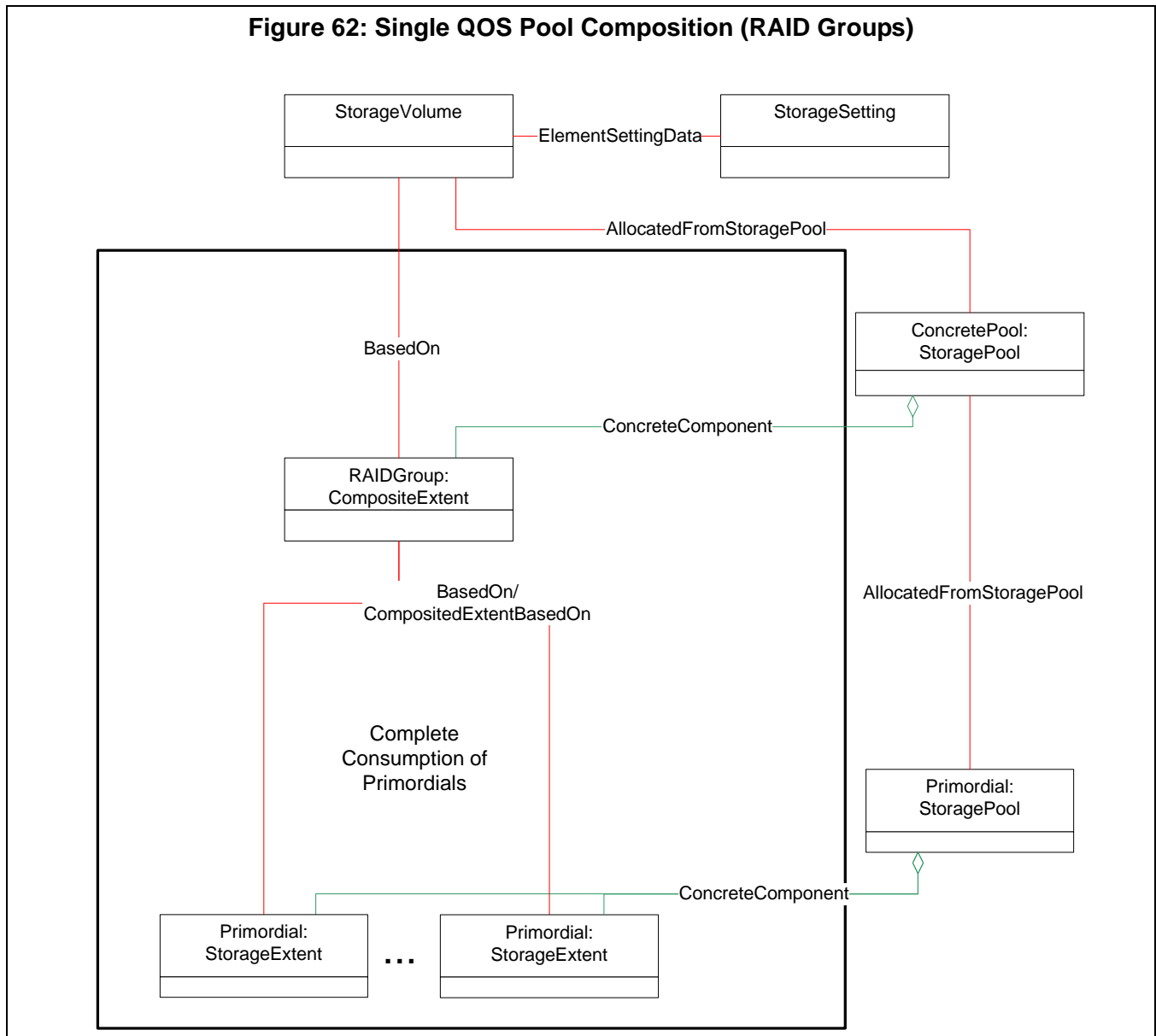
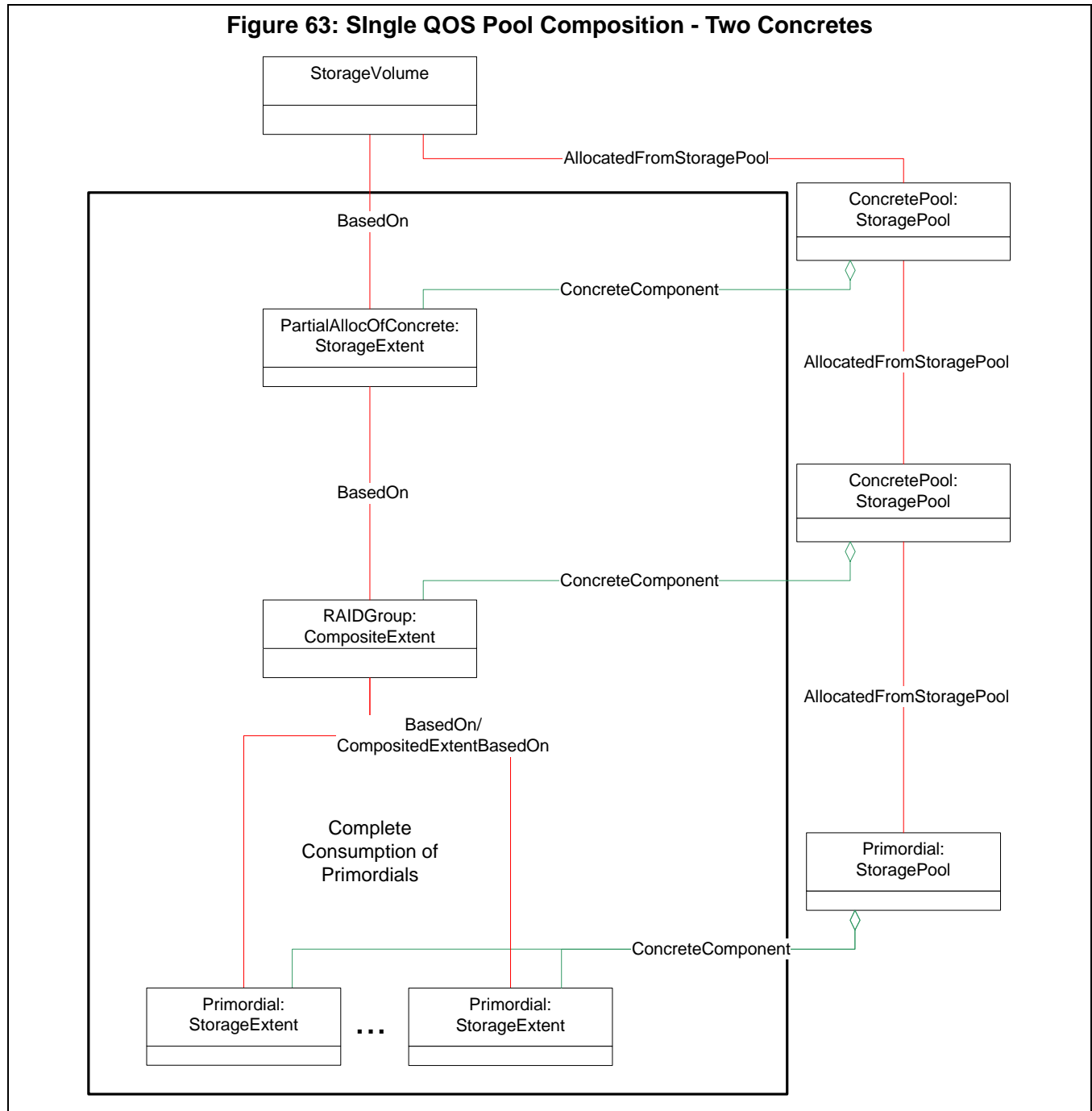


Figure 63 extends this scenario by allocating a child concrete pool from the RAID Group instead of a Volume and then allocating the Volume from the child concrete. In this example the child pool contains a single component extent that has a single Quality of Service (that of the parent RAID Group concrete pool). The Storage Volume or Logical Disk is allocated or decomposed directly from the pool component extent.



15.1.5.3 Example RAID Compositions from Block Services

Table 216 is an abridged version of the RAID Mapping table in Block Services. The RAID Mapping table describes the RAID Levels commonly used at the time this version of SMI-S was released. Table 216 lists the subset of those RAID Levels that can be modeled by using the Extent Composition subprofile, and the Properties used to distinguish them.

Following the table are some example instance diagrams, showing the use of CompositeExtent, StorageExtent, BasedOn and CompositeExtentBasedOn to represent the construction of many of the RAID levels. In these cases there will be at most, two levels of CompositeExtent and CompositeExtentBasedOn/BasedOn.

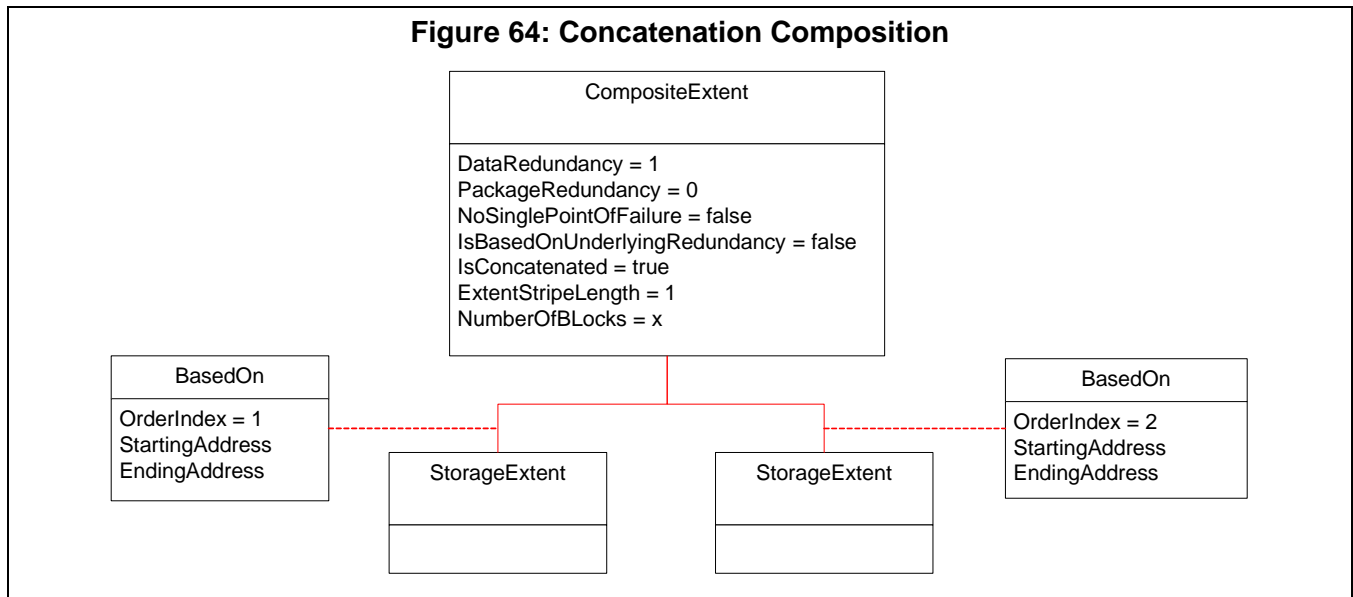
In complex compositions, such as RAID 10, there is no intermediate decomposition modeled; each extent Antecedent to the top level CompositeExtent is itself a CompositeExtent.

Table 216: Supported Common RAID Levels

RAID Level	Package Redundancy	Data Redundancy	Extent Stripe Length	User Data Stripe Depth
JBOD	0	1	1	Null
0 (Striping)	0	1	2 - n	Vendor Dependent
1	1	2 - n	1	Null
10	1	2 - n	2 - n	Vendor Dependent
0+1	1	2 - n	2 - n	Vendor Dependent
3 or 4	1	1	3 - n	Vendor Dependent
4DP	2	1	4 - n	Vendor Dependent
5 (3/5)	1	1	3 - n	Vendor Dependent
6, 5DP	2	1	4 - n	Vendor Dependent
15	2	2 - n	3 - n	Vendor Dependent
50	1	1	3 - n	Vendor Dependent
51	2	2 - n	3 - n	Vendor Dependent

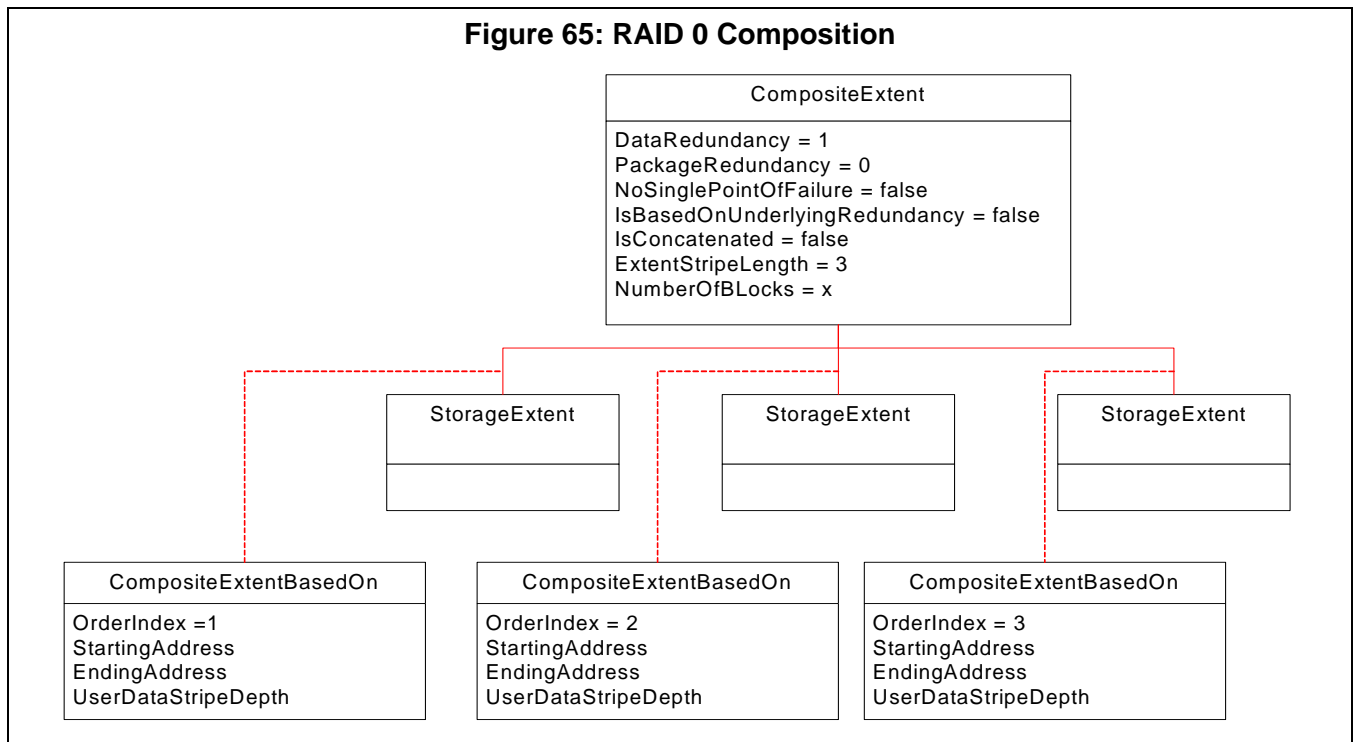
15.1.6 JBOD (Concatenation)

Figure 64 shows an partial instance diagram for a JBOD Volume or Pool, in which the Antecedent Extents are concatenated.



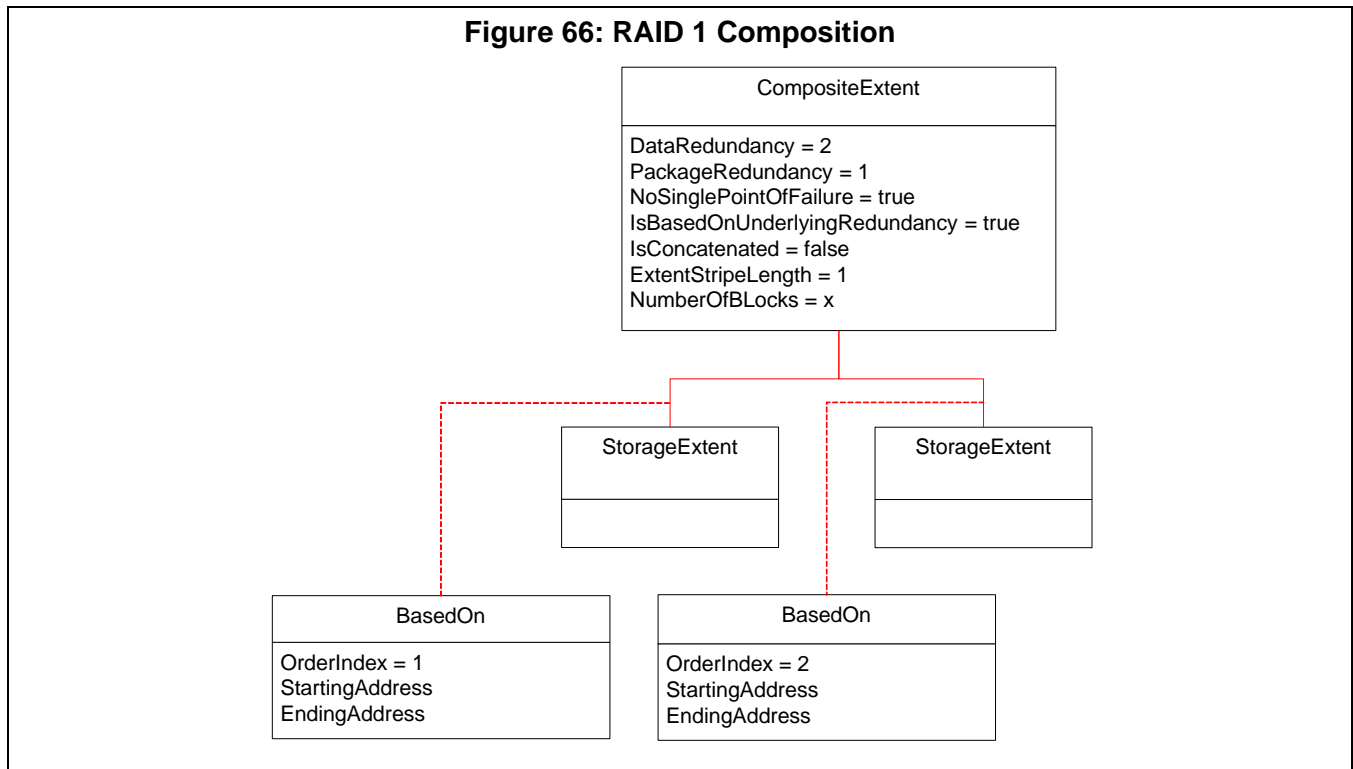
15.1.7 RAID 0 (Striping)

Figure 65 shows an partial instance diagram for a RAID 0 Volume or Pool.



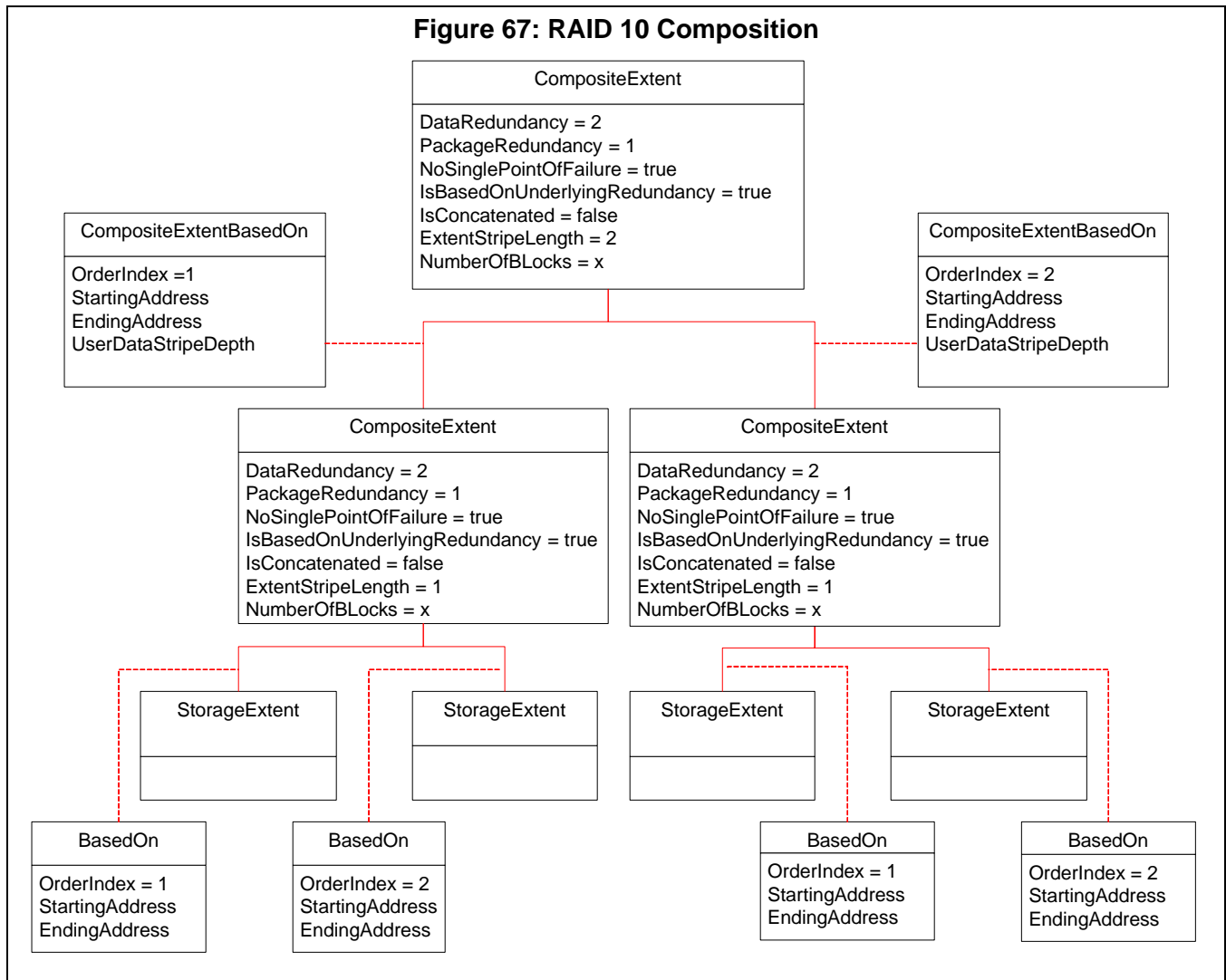
15.1.8 RAID 1

Figure 66 shows an partial instance diagram for a RAID 1 Volume or Pool.



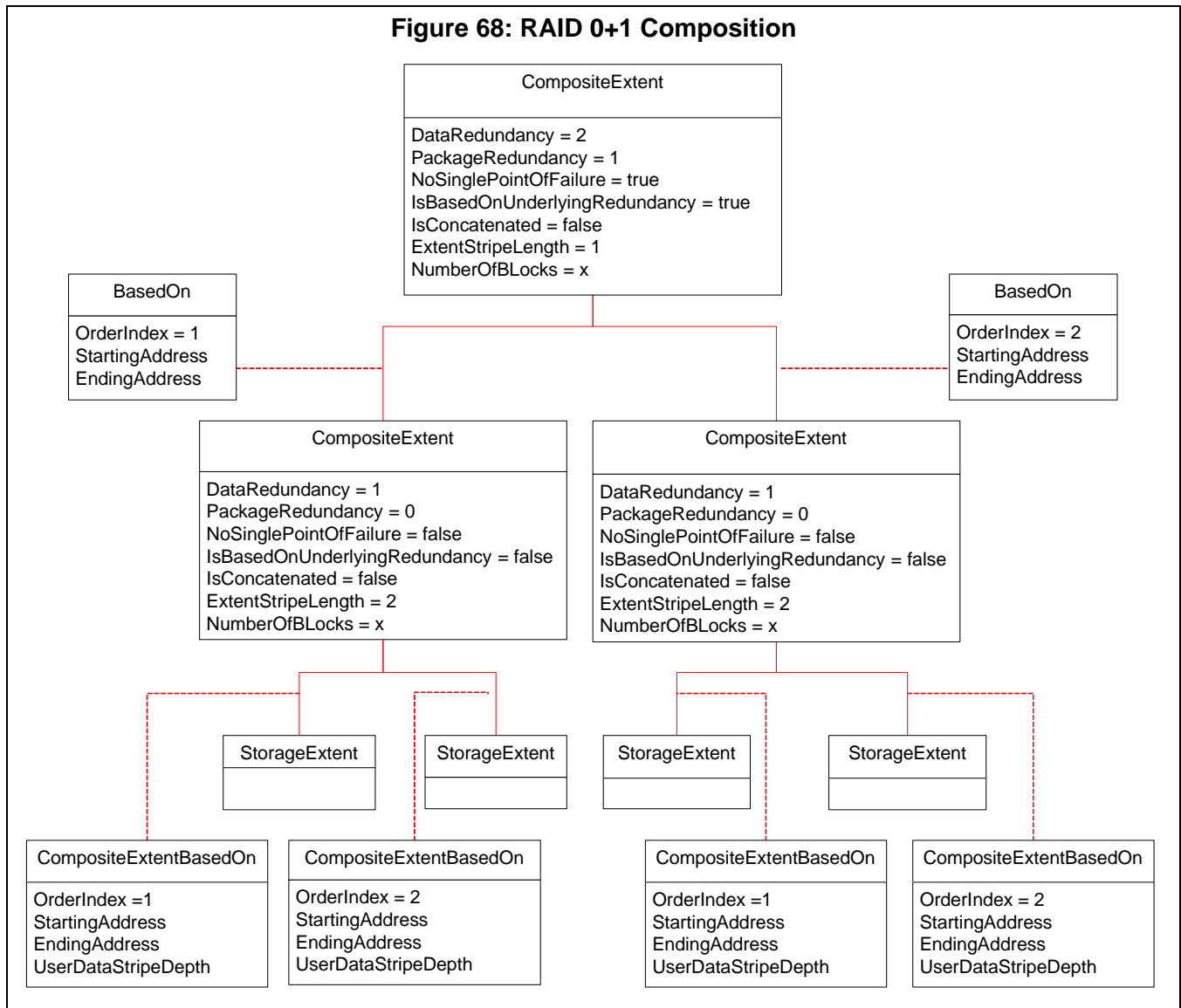
15.1.9 RAID 10

Figure 67 shows an partial instance diagram for a RAID 10 Volume or Pool. In this example the Data and Package Redundancy reflect the Quality of Service of the combined RAID Level, not just the top level composition which by itself is a non-redundant stripeset. That is, the top level is a RAID 0, but the DataRedundancy value for the corresponding CompositeExtent is 2, reflecting two complete copies of the data.

Figure 67: RAID 10 Composition

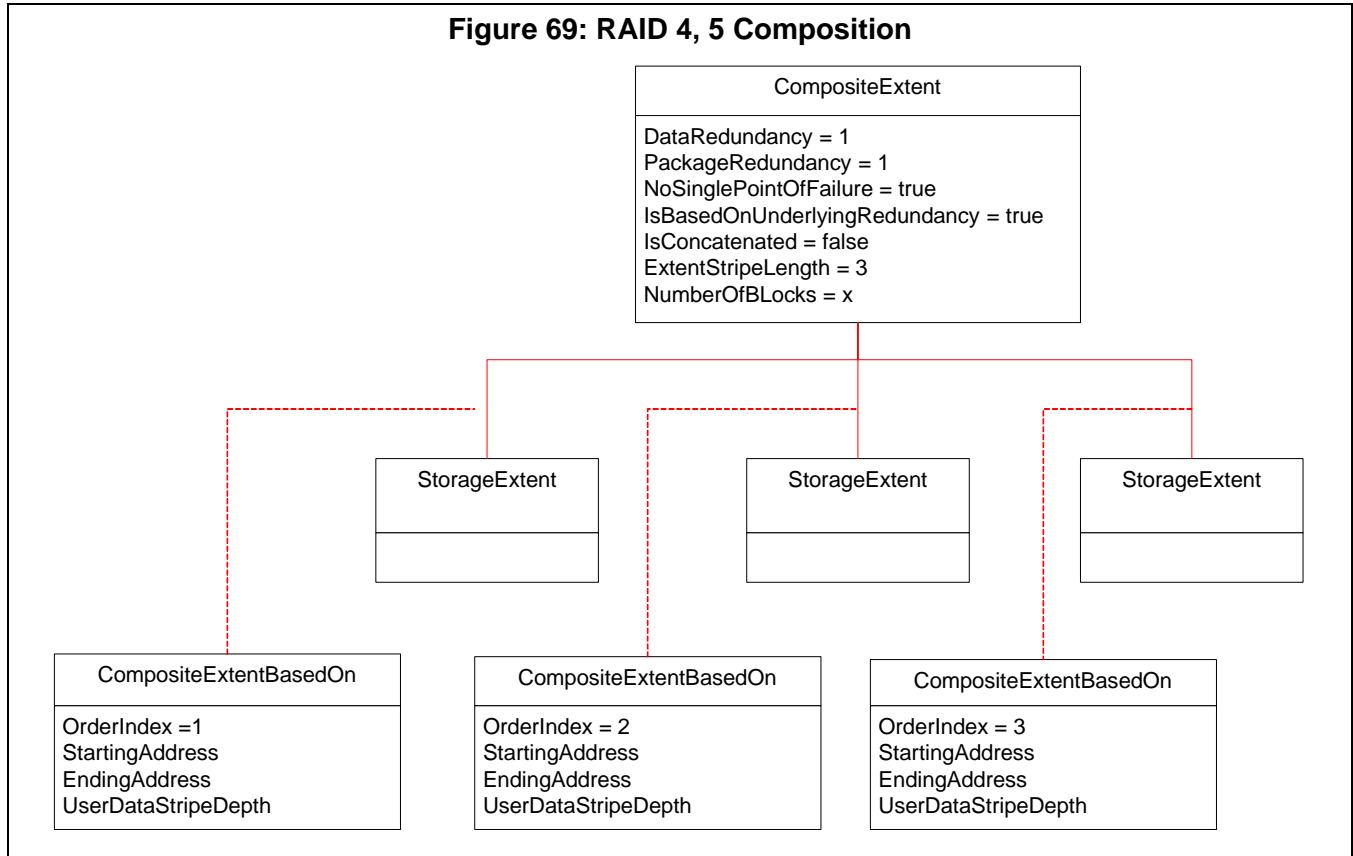
15.1.10 RAID 0+1

Figure 68 shows an partial instance diagram for a RAID 0+1 Volume or Pool



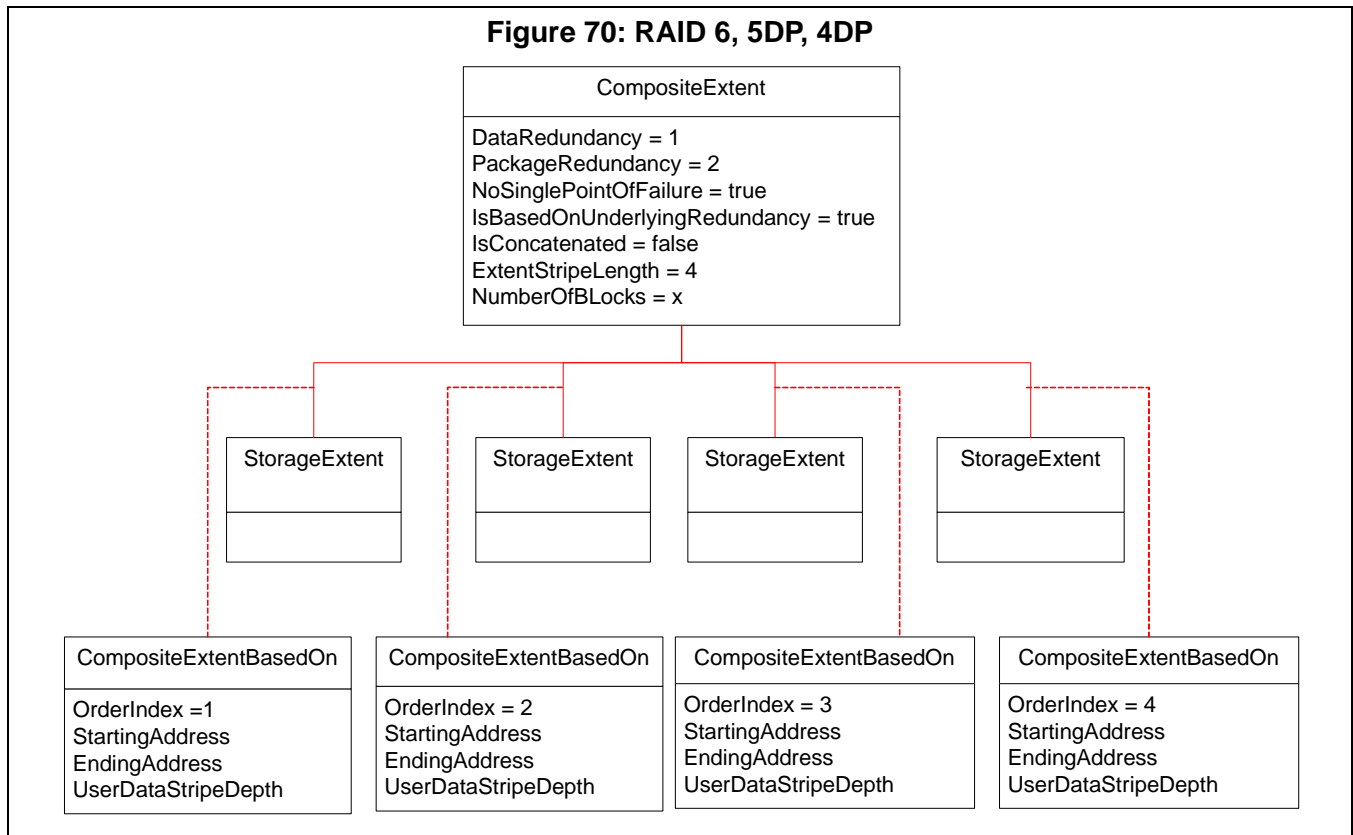
15.1.11 RAID 4 or 5

Figure 69 shows a partial instance diagram for a RAID 4 or 5 Volume or Pool.



15.1.12 RAID 6, 5DP, and 4DP

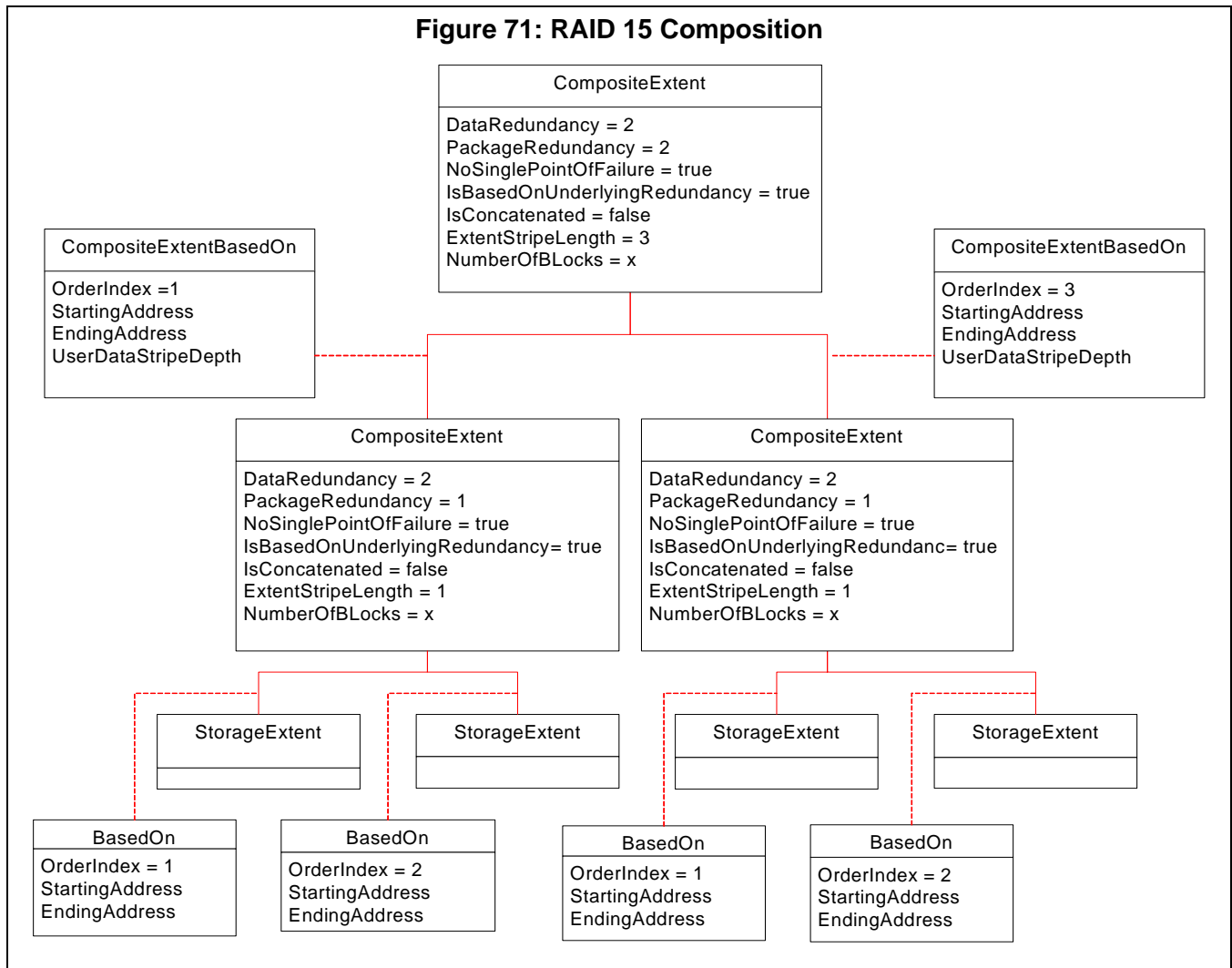
Figure 70 shows an partial instance diagram for a RAID6, 5DP, or 4DP Volume or Pool. Note that the PackageRedundancy is 2, indicating that two of the antecedent extents can fail simultaneously without loss of data. Four extents are shown, the minimum required for these double parity RAID organizations.



15.1.13 RAID 15

Figure 71 shows an partial instance diagram for a RAID 15 Volume or Pool. In this example the Data and Package Redundancy reflect the Quality of Service of the combined RAID Level, not just the top level composition which by itself is a simple RAID 5.

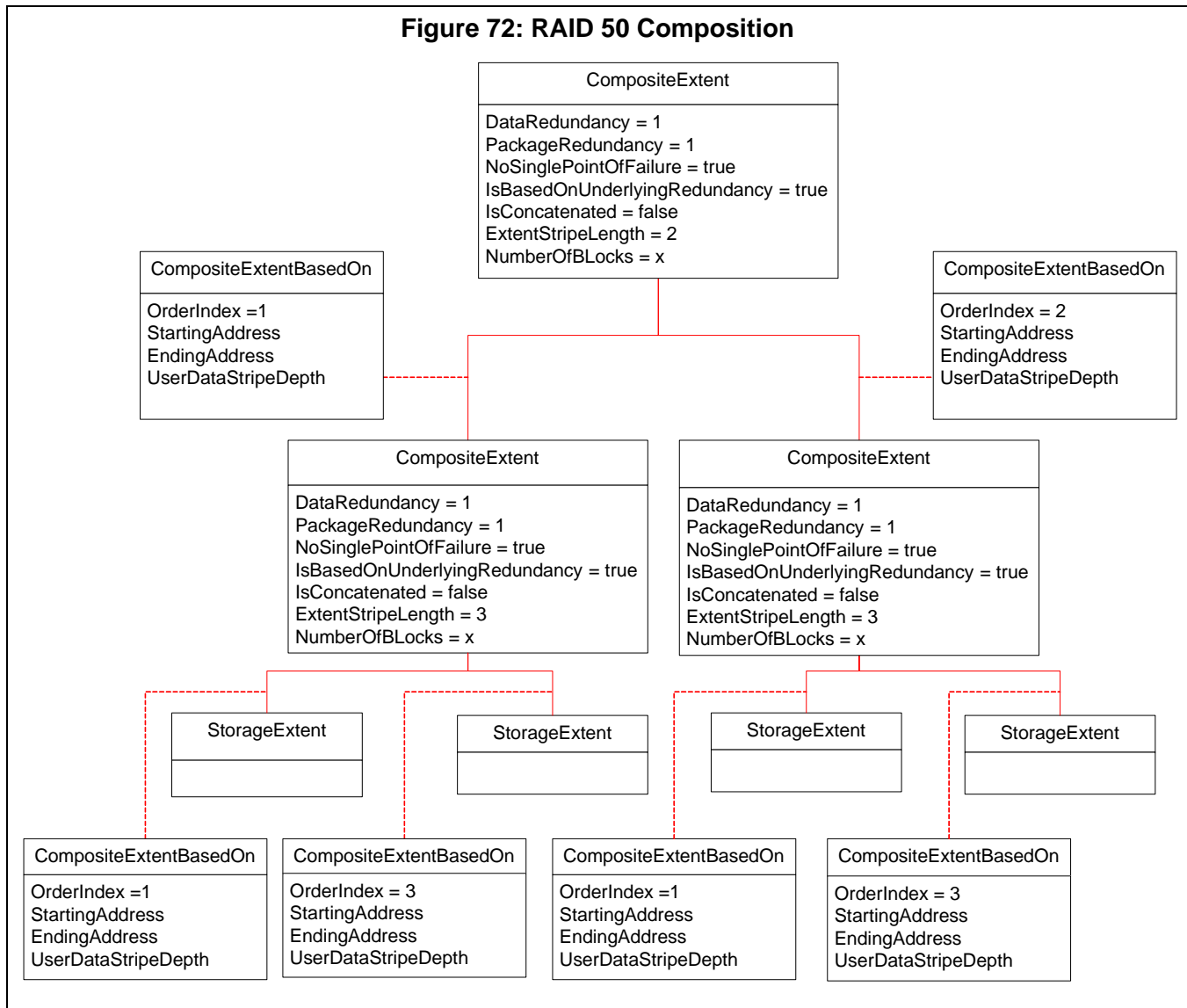
Note: Only CompositeExtent members 1 and 3 of the Raid 5 layer are shown.



15.1.14 RAID 50

Figure 72 shows an partial instance diagram for a RAID 50 Volume or Pool. In this example the Data and Package Redundancy reflect the Quality of Service of the combined RAID Level, not just the top level composition which by itself is a non-redundant stripeset.

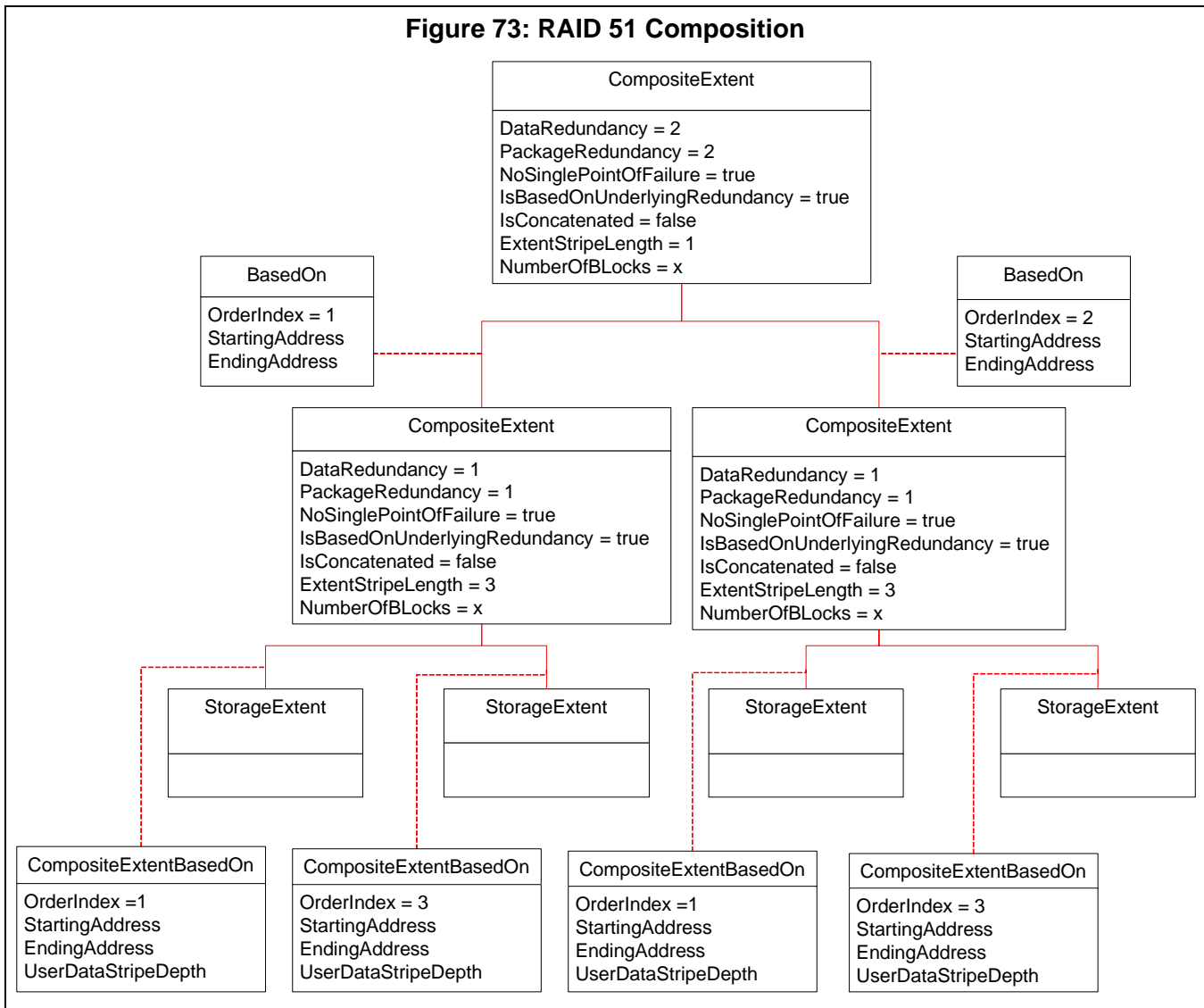
Note: In the Raid 5 layer, CompositeExtent member 2 in each stripe member is not shown.



15.1.15 RAID 51

Figure 73 shows an partial instance diagram for a RAID 51 Volume or Pool. In this example the Data and Package Redundancy reflect the Quality of Service of the combined RAID Level, not just the top level composition which by itself is a simple mirror. That is, the top level is a RAID 1, but the `PackageRedundancy` is 2, indicating the QOS for the entire hierarchy.

Note: In the Raid 5 layer, CompositeExtent member 2 in each mirror is not shown.

Figure 73: RAID 51 Composition

15.2 Health and Fault Management Considerations

Not defined in this standard.

15.3 Cascading Considerations

None.

15.4 Supported Subprofiles and Packages

None.

15.5 Methods of the Profile

None.

15.6 Client Considerations and Recipes

15.6.1 Traverse the virtualization hierarchy of a StorageVolume or LogicalDisk

```
// DESCRIPTION
//
// This recipes defines a mechanism for traversing the extent hierarchy between
// the Exposable Block Storage Element and the Primordial Extents it makes use
// of, determining the RAID level structure, Concrete and Primordial pool
// membership.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance name for an exposable block storage element (e.g.
// StorageVolume, LogicalDisk) of interest has been previously identified as
// $BlockElement->.

// This function determines if an Extent is a Primary(non-remaining) Component
// of a Pool.
//
sub boolean IsPrimaryComponent(REF $TargetExtent->) {
    $Pools->[] = AssociatorNames($TargetExtent->,
        "CIM_ConcreteComponent",
        "CIM_StoragePool",
        "PartComponent",
        "GroupComponent")

    if ($Pools->[] != null && $Pools->[].length == 1) {
        // This Extent is a Component Extent of either a Concrete
        // or Primordial pool
        return true
    }
    else
        return false
}

// This function determines the RAID Level or Quality of Service of a
// CompositeExtent and then recursively traverses the hierarchy beneath it.
//
sub void traverseComposition(REF $Composite->) {

    // See if this composite is a Primary(non-remaining) Component
    // Extent of a Pool (for information only.)
```

Extent Composition Subprofile

```
#PrimaryComponent = &IsPrimaryComponent($Composite->)

// Get the instances of the associations in which this Extent is the
// Dependent reference. The association instances retrieved should be
// either BasedOn or CompositeExtentBasedOn.
$Associations[] = References($Composite->,
    NULL,
    "Dependent",
    false,
    false,
    NULL)

// Now get the underlying extents
$TargetExtents->[] = AssociatorNames($Composite->,
    Associations[0].getClassName(),
    NULL,
    "Dependent",
    "Antecedent")

// Examine the QOS of the current level's Composite Extent
$CompositeExtent = GetInstance($Composite->,
    false,
    false,
    false,
    {"IsConcatenated", "ExtentStripeLength",
    "IsBasedOnUnderlyingRedundancy"})

if (($Associations[0] ISA CIM_CompositeExtentBasedOn)
    && ($CompositeExtent.IsConcatenated == false)
    && ($CompositeExtent.ExtentStripeLength > 1)) {

    // The TargetExtents are striped together. Get the Stripe Depth from
    // the first association. The assumption here is that this property is
    // the same for each association instance.
    #StripeDepth = $Associations[0].UserDataStripeDepth

    // Inspect the RAID level.
    #RAID = 0
    if ($CompositeExtent.IsBasedOnUnderlyingRedundancy) {
        #RAID = 5
    }
} else {
    // Associations are CIM_BasedOn, So this is either a Mirror or
    // a Concatenation
    if (($CompositeExtent.IsBasedOnUnderlyingRedundancy == true)
        && ($CompositeExtent.IsConcatenated == false)
        && ($CompositeExtent.ExtentStripeLength == 1)) {
```

Extent Composition Subprofile

```

        // The TargetExtents are mirrored together,
        // This level is a RAID 1
    } else if (($CompositeExtent.IsBasedOnUnderlyingRedundancy == false)
        && ($CompositeExtent.IsConcatenated == true)
        && ($CompositeExtent.ExtentStripeLength == 1)) {
        // The TargetExtents are concatenated together,
        // This level is a JBOD.
    } else {
        <ERROR! Illegal combination of property values; does not
            correspond to supported composition type.>
    }
}

// Now for each underlying extent at this level, traverse the sub-tree
// it is the sub-root of. If the extent is a CompositeExtent, then this
// is part of a complex RAID level; recursively invoke the Composition
// Algorithm. Otherwise it is just a regular StorageExtent and thus
// either a Primordial or decomposed from an Antecedent, so invoke the
// recursive Decomposition Algorithm.
for (#i in $TargetExtents->[]) {
    if ($TargetExtents->[#i] ISA CIM_CompositeExtent) {
        &traverseComposition($TargetExtents->[#i])
    } else {
        &traverseDecomposition($TargetExtents->[#i])
    }
}

}

// This function recursively traverses the hierarchy below a non-Composite
// Storage Extent.
sub void traverseDecomposition(REF $SubjectExtent->) {

    // See if this extent is a Primary(non-remaining) Component
    // Extent of a Pool (for information only.)
    #PrimaryComponent = &IsPrimaryComponent($SubjectExtent->)

    // Check here to see if we have reached the leaves of the hierarchy
    $SubjectExtent = GetInstance($SubjectExtent->,
        false,
        false,
        false,
        {"Primordial"})

    if ($SubjectExtent.Primordial == true) {
        // Recursion ends with each Primordial Extent.
        <EXIT: Recursion ends with each Primordial Extent.>
    }
}

```

```

} else {

    // The Subject Extent is allocated partially or in full from the
    // Antecedent Extent, so a single BasedOn is expected.
    $TargetExtents[] = Associators($SubjectExtent->,
        "CIM_BasedOn",
        "CIM_StorageExtent",
        "Dependent",
        "Antecedent",
        false,
        false,
        {"Primordial"})

    // Since the Subject Extent is allocated from the Antecedent, there can
    // only be one Antecedent.
    if ($TargetExtents[] == null || $TargetExtents[].length != 1) {
        <ERROR! Extent allocated from multiple Antecedents>
    }
    $TargetExtent = $TargetExtents[0]

    if ($TargetExtent ISA CIM_CompositeExtent) {
        // This is a Composite Extent representing a RAID Level. Since we
        // encountered the Composite in a decomposition, the
        // Dependent/Antecedent relationship falls into one of the
        // following scenarios:
        //
        // o The Subject Extent is a StorageVolume that is one-to-one with
        //   the Target Composite Extent.
        //
        // o The Subject Extent is a StorageVolume partially allocated from
        //   the Target Composite Extent, where the Composite is a RAID Group.
        //
        // o The Subject Extent is a ComponentExtent of a Concrete pool and is
        //   partially allocated from the Target Composite Extent where the
        //   Composite is a RAID Group.
        //
        // Call the (recursive) function to analyze the sub-hierarchy
        // composed by the Target Extent.
        //
        &traverseComposition($TargetExtent.GetObjectPath())
    } else {
        // The Antecedent is a regular StorageExtent and was not
        // Primordial, so it must be in turn a dependent decomposed
        // from an Antecedent, so invoke
        // ourselves recursively.
        &traverseDecomposition($TargetExtent.GetObjectPath())
    }
}

```

```

    }
}

// MAIN
// Since the exposable block element is either one-to-one with the initial
// CompositeExtent, or a partial allocation of it (in the case of a RAID Group),
// decompose the block hierarchy.
//
&traverseDecomposition($BlockElement->)

```

15.6.2 Find the Primordial Extents used by a Storage Volume or Logical Disk

A storage administrator may want the information provided by this recipe for several reasons:

Failure Exposure: To understand what Drive or virtualized Volume failures may affect the health of a block storage element, or conversely what block storage elements are affected by a given Drive failure.

Performance and Loading: To avoid locating frequently accessed Volumes on the same Disk Drive.

Utilization: To avoid locating portions of too many volumes on the same Drive while leaving other drives under utilized.

```

// DESCRIPTION
//
// This recipe defines a mechanism for finding the Primordial Storage Extents
// used by a Storage Volume in an Array or Virtualizer, or a LogicalDisk in
// a Volume Manager or NAS system.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance name for an exposable block storage element (e.g.
// StorageVolume, LogicalDisk) of interest has been previously identified as
// $BlockElement->.

// This function recursively searches for the Primordial Storage Extents that
// comprise the specified block storage element.
sub $PrimordialExtents[] findPrimordials(REF $SubjectExtent->) {

    // Get the Extents that are Antecedent to the specified Extent.
    //
    $TargetExtents[] = Associators($SubjectExtent->,
        "CIM_BasedOn",
        "CIM_StorageExtent",
        "Dependent",
        "Antecedent",
    )
}

```



```

        false,
        false,
        {"Primordial"})

// Examine each Extent at the next level to determine if its Primordial.
#i = 0
for (#j in $TargetExtents[]) {
    if ($TargetExtents[#j].Primordial == true) {
        // The Extent is Primordial, the recursion ends here. Add it to
        // the group of Primordials gathered at this level or below.
        $PrimordialExtents[#i++] = TargetExtents[j]
    } else {
        // The Extent is not Primordial, but it must be based on a
        // sub-hierarchy in which each leaf is a Primordial, so call this
        // function Recursively.
        $SubordinatePrimordialExtents[] =
            &findPrimordials(TargetExtents[#j].getObjectPath())
        if ($SubordinatePrimordialExtents[] == null
            || $SubordinatePrimordialExtents[].length == 0) {
            <ERROR! Found a Leaf Extent that is not a Primordial>
        }

        for (#k in $SubordinatePrimordialExtents[]) {
            // The recursion delivers the bottom for each branch
            // These need to be collected and added into the whole
            $PrimordialExtents[#i++] = SubordinatePrimordialExtents[#k]
        }
    }
}
return ($PrimordialExtents[])
}

// MAIN
// Make initial call to the recursive function.
$PrimordialExtents[] = &findPrimordials($BlockElement->)
if ($PrimordialExtents[] == null || $PrimordialExtents[].length == 0) {
    <ERROR! No Primordials Found>
} else {
    <EXIT: Primordial Extents accumulated>
}

```

15.7 Registered Name and Version

Extent Composition version 1.2.0

15.8 CIM Elements

Table 217: CIM Elements for Extent Composition

Element Name	Requirement	Description
CIM_StorageExtent (Allocated Extent) (15.8.1)	Mandatory	Represents the partial allocation(decomposition) or complete allocation of an Antecedent extent.
CIM_CompositeExtent (Composite Extent) (15.8.2)	Mandatory	Represent the composition of several antecedent extents into a virtual range of blocks with desired size or Quality-Of-Service.
CIM_BasedOn (Volume to Composite) (15.8.3)	Conditional	A one-to-one (neither composing or decomposing) , associating the StorageVolume to the antecedent CompositeExtent. This is because, as a sibling of StorageVolume, CompositeExtent cannot be exposed directly.
CIM_BasedOn (LogicalDisk to Composite) (15.8.4)	Conditional	A one-to-one (neither composing or decomposing) , associating the LogicalDisk to the antecedent CompositeExtent. This is because, as a sibling of LogicalDisk, CompositeExtent cannot be exposed directly.
CIM_BasedOn (Non-striping Composite to Composite Extent) (15.8.5)	Mandatory	Associates a Composite Extent representing a non-striping simple RAID organization such as mirroring or concatenation to the underlying Composite Extents that it virtualizes in order to represent a complex RAID organization such as RAID 51..
CIM_CompositeExtentBasedOn (Striping Composite to Composite Extent) (15.8.6)	Mandatory	Associates a Composite Extent representing a striping simple RAID organization such as RAID 0 or RAID 5 to the underlying Composite Extents that it virtualizes in order to represent a complex RAID organization such as RAID 10..
CIM_BasedOn (Non-striping Composite to Allocated Extent) (15.8.7)	Mandatory	Associates a Composite Extent representing a non-striping simple RAID organization such as mirroring or concatenation to the underlying Storage Extents that it virtualizes...
CIM_CompositeExtentBasedOn (Striping Composite to Allocated Extent) (15.8.8)	Mandatory	Associates a Composite Extent representing a striping simple RAID organization such as RAID 0 or RAID 5 to the underlying Storage Extents that it virtualizes.
CIM_BasedOn (Allocated Extent to Composite Extent) (15.8.9)	Mandatory	Associates a Storage Extent representing a partial allocation or decomposition to the underlying Composite Extent (such as a RAID Group) that it is allocated from..

Table 217: CIM Elements for Extent Composition

Element Name	Requirement	Description
CIM_BasedOn (Allocated Extent to Allocated Extent) (15.8.10)	Mandatory	Associates a Storage Extent representing a partial allocation or decomposition to the underlying Storage Extent(such as a Component Extent) that it is allocated from..
CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent) (15.8.11)	Conditional	Conditional requirement: References Disk Drive Lite - Antecedent Primordial is mandatory. Associates an Composite Extent or Allocated Extent(such as a Component Extent) to the underlying PrimordialExtent in one of the Block or NAS profiles
CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent) (15.8.12)	Conditional	Associates an Composite Extent or Allocated Extent(such as a Component Extent) to the underlying PrimordialExtent in one of the Block or NAS profiles
CIM_BasedOn (StorageExtent or CompositeExtent To Logical Disk) (15.8.13)	Conditional	Conditional requirement: Referenced from Volume Management - Dependent and Antecedent(input) LogicalDisk are mandatory. Associates an Composite Extent or Allocated Extent(such as a Component Extent) to the underlying LogicalDisk in the Volume Management profile
CIM_ConcreteComponent (15.8.14)	Mandatory	Associate extents that are playing the Pool Component role to their aggregating StoragePool.

15.8.1 CIM_StorageExtent (Allocated Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 218 describes class CIM_StorageExtent (Allocated Extent).

Table 218: SMI Referenced Properties/Methods for CIM_StorageExtent (Allocated Extent)

Properties	Flags	Requirement	Description & Notes
SystemCreationClass		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	

Table 218: SMI Referenced Properties/Methods for CIM_StorageExtent (Allocated Extent)

Properties	Flags	Requirement	Description & Notes
DeviceID		Mandatory	
ExtentStatus		Mandatory	
NumberOfBlocks		Mandatory	
ConsumableBlocks		Mandatory	The number of usable blocks.
BlockSize		Mandatory	
Primordial		Mandatory	

15.8.2 CIM_CompositeExtent (Composite Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 219 describes class CIM_CompositeExtent (Composite Extent).

Table 219: SMI Referenced Properties/Methods for CIM_CompositeExtent (Composite Extent)

Properties	Flags	Requirement	Description & Notes
Name	CD	Mandatory	
Primordial		Mandatory	
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
ExtentStatus		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
IsBasedOnUnderlyingRedundancy		Mandatory	
IsConcatenated		Mandatory	

Table 219: SMI Referenced Properties/Methods for CIM_CompositeExtent (Composite Extent)

Properties	Flags	Requirement	Description & Notes
ExtentStripeLength		Mandatory	
NumberOfBlocks		Mandatory	
ConsumableBlocks		Mandatory	The number of usable blocks.
BlockSize		Mandatory	

15.8.3 CIM_BasedOn (Volume to Composite)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Array|Storage Virtualizer|Host Hardware RAID

Table 220 describes class CIM_BasedOn (Volume to Composite).

Table 220: SMI Referenced Properties/Methods for CIM_BasedOn (Volume to Composite)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.4 CIM_BasedOn (LogicalDisk to Composite)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Volume Management|Self-contained NAS System|NAS Head

Table 221 describes class CIM_BasedOn (LogicalDisk to Composite).

Table 221: SMI Referenced Properties/Methods for CIM_BasedOn (LogicalDisk to Composite)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.5 CIM_BasedOn (Non-striping Composite to Composite Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 222 describes class CIM_BasedOn (Non-striping Composite to Composite Extent).

Table 222: SMI Referenced Properties/Methods for CIM_BasedOn (Non-striping Composite to Composite Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	When the association is used in a concatenation composition, indicates the order in which the extents(and thus their block ranges) are concatenated.
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.6 CIM_CompositeExtentBasedOn (Striping Composite to Composite Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 223 describes class CIM_CompositeExtentBasedOn (Striping Composite to Composite Extent).

Table 223: SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Striping Composite to Composite Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	Indicates the order in which the antecedent extents have blocks striped onto them.
UserDataStripeDepth		Mandatory	The number of blocks written to an Antecedent extent before moving on to the next extent Although this property is on the association class, its values shall be the same for each instance of the association with the same Dependent CompositeExtent.
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.7 CIM_BasedOn (Non-striping Composite to Allocated Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 224 describes class CIM_BasedOn (Non-striping Composite to Allocated Extent).

Table 224: SMI Referenced Properties/Methods for CIM_BasedOn (Non-striping Composite to Allocated Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	When the association is used in a concatenation composition, indicates the order in which the extents(and thus their block ranges) are concatenated.
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.8 CIM_CompositeExtentBasedOn (Striping Composite to Allocated Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 225 describes class CIM_CompositeExtentBasedOn (Striping Composite to Allocated Extent).

Table 225: SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Striping Composite to Allocated Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	Indicates the order in which the antecedent extents have blocks striped onto them.
UserDataStripeDepth		Mandatory	The number of blocks written to an Antecedent extent before moving on to the next extent Although this property is on the association class, its values shall be the same for each instance of the association with the same Dependent CompositeExtent.
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.9 CIM_BasedOn (Allocated Extent to Composite Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 226 describes class CIM_BasedOn (Allocated Extent to Composite Extent).

Table 226: SMI Referenced Properties/Methods for CIM_BasedOn (Allocated Extent to Composite Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.10 CIM_BasedOn (Allocated Extent to Allocated Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 227 describes class CIM_BasedOn (Allocated Extent to Allocated Extent).

Table 227: SMI Referenced Properties/Methods for CIM_BasedOn (Allocated Extent to Allocated Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.11 CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Disk Drive Lite

Table 228 describes class CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent).

Table 228: SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or Composite-Extent To Primordial Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.12 CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Storage Virtualizer|NAS Head

Table 229 describes class CIM_BasedOn (StorageExtent or CompositeExtent To Primordial Extent).

Table 229: SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or Composite-Extent To Primordial Extent)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.13 CIM_BasedOn (StorageExtent or CompositeExtent To Logical Disk)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Volume Management

Table 230 describes class CIM_BasedOn (StorageExtent or CompositeExtent To Logical Disk).

Table 230: SMI Referenced Properties/Methods for CIM_BasedOn (StorageExtent or Composite-Extent To Logical Disk)

Properties	Flags	Requirement	Description & Notes
StartingAddress		Optional	
EndingAddress		Optional	
OrderIndex		Mandatory	
Dependent		Mandatory	
Antecedent		Mandatory	

15.8.14 CIM_ConcreteComponent

The ConcreteComponents associating the PrimordialExtents to the PrimordialPool are NOT part of ExtentComposition.

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 231 describes class CIM_ConcreteComponent.

Table 231: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	
PartComponent		Mandatory	

STABLE

DEPRECATED

Clause 16: LUN Creation Subprofile

The functionality of the LUN Creation and Pool Manipulation Capabilities, and Settings Subprofiles have been subsumed by the Clause 5: Block Services Package.

The LUN Creation Subprofile is defined in section 7.3.3.11 of SMI-S 1.0.2.

DEPRECATED

DEPRECATED**Clause 17: LUN Mapping and Masking Subprofile**

The LUN Mapping and Masking Subprofile (section 7.3.3.14 in SMI-S 1.0.2) has been replaced by Clause 18: Masking and Mapping Subprofile.

17.1 Compatibility with SMI-S 1.0 clients.

Problems with the functionality and complexity of the LUN Mapping and Masking subprofile in SMI-S 1.0 required some changes that may not be backwards compatible in the 1.1.0 version. The mapping and Masking Subprofile now reduces the complexity by replacing the 1.0.2 extrinsic methods and severely constraining the valid combinations of parameters. Additionally, changes made to support non-FC transports and non-SCSI protocols also affect backwards compatibility. Specifically, associating the `SCSIProtocolController` to a `SCSIProtocolEndpoint` instead of `LogicalPort`. `SCSIProtocolEndpoint` is associated to the `LogicalPort`. Separating the port from the protocol allows the port to be used with non-SCSI protocols such as IP. Most of the model is identical, but new classes, properties, and methods have been added to simplify its operation. Some of the old methods are still used in 1.1.0.

Class and association changes to the model for 1.1.0:

- `SAPAvailableForElement` replaces the `ProtocolControllerForPort` association
- `SCSIProtocolEndpoint` replaces `LogicalPort`
- `LogicalPort` is associated to `SCSIProtocolEndpoint` via `PortImplementsEndpoint` (see Clause 17: LUN Mapping and Masking Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles*.)
- `AuthorizedPrivilege` associations to `SystemSpecificCollection` via `AuthorizedSubject` associations are no longer allowed

Instrumentation may be able to provide 1.0.2 and 1.0 compliant implementations in a single namespace, if the following conditions are met:

- `ProtocolControllerMaskingCapabilities.ProtocolControllerSupportsCollections` is `false` (`StorageHardwareID` instances are referenced directly by `AuthorizedSubject` associations).
- There is exactly a 1-1-1 relationship between instance of `AuthorizedSubject`, `AuthorizedPrivilege`, and `AuthorizedTarget`. In other words, `Privilege` instances cannot be shared.

If these criteria are not met, instrumentation could provide separate 1.0.2 and 1.1.0 implementations in separate CIM namespaces.

DEPRECATED

STABLE**Clause 18: Masking and Mapping Subprofile****18.1 Description**

Note: See 17.1 for notes on compatibility with the LUN Mapping and Masking Subprofile in SMI-S 1.0.2.

Many disk arrays provide an interface for the administrator to specify which initiators can access what volumes through which target ports. The effect is that the given volume is only visible to SCSI commands that originate from the specified initiators through specific sets of target ports. There may also be a capability to select the SCSI Logical Unit Number as seen by an initiator through a specific set of ports. The ability to limit access is called *Device Masking*; the ability to specify the device address seen by particular initiators is called *Device Mapping* (For SCSI systems, these terms are known as *LUN Masking* and *LUN Mapping*.)

Given a storage system with no LUN masking or mapping, all hosts/initiators see the same elements when they discover a storage system. In a storage system supporting LUN Masking, logical units are masked (hidden) from SCSI initiators (Host Bus Adaptors) by default. The administrator uses the Masking and Mapping subprofile to determine which logical units are visible (exposed) to specific initiators through which target ports. The LUN masking and mapping interfaces allow an administrator to customize the “view” of elements that are discovered. The effect is that the real storage system appears to be a number of subsets - each subset exposing a view customized for a particular set of initiators.

The management model is built on these “views” of a storage system - each view is a subset of components the administrator exposes to certain hosts - and the classes that model the authorization and access rights.

The model described here is generalized to include access management in disks arrays, virtualization systems, and routers used in tape libraries. The model is also generalized beyond just SCSI and Fibre Channel implementations. Many of the examples and use cases refer to LUN masking in Fibre Channel arrays, but the model is general.

18.1.1 Views and Paths

The key concepts for Device Masking and Mapping are view and path. A “view” is a list of logical units exposed to a list of initiators through a list of target ports, modeled as `SCSIProtocolController` (SPC) with associated `LogicalDevices`, `StorageHardwareIDs`, and `SCSIProtocolEndpoints`. The logical devices have logical unit numbers and access permissions relative to the view, modeled as `DeviceNumber` and `DeviceAccess` properties of the `ProtocolControllerForUnit` association. A full “path” is a combination of one each logical unit, initiator port, and target port - the concept of path is independent from a CIM model, but a view expresses a combinations of paths that comply with SCSI rules. In essence, an SPC serves as a collection of paths - each initiator ID is granted access to each logical unit through each target port.

In addition, there are partial and invalid states. A partial path is a path missing associations to instances of logical unit, initiator port, or target port. In practice, some arrays do not support partial paths and other arrays support some, but not all, configurations with partial paths. An SPC lacking associations to logical units, initiator ports, or target ports - as required by the underlying implementation - is in an invalid partial path state.

An invalid view state is a combination of classes and associations in the provider that does not map to a committed configuration of the underlying implementation. The 1.0 LUN Masking and Mapping interfaces required clients to perform multiple transactions to achieve a valid view, forcing providers to maintain invalid view states while waiting for the client to complete a sequence of transactions. This created non-interoperability when the providers only supported transactions in a certain order, and when a second client looked at the model before a sequence of transactions was completed.

An SPC with no instances of one type of association (to initiators, targets, or LUs) with support from the instrumentation is in a valid partial path state. The result is that the SPC does not expose any valid SCSI paths.

Instrumentation may support these states as convenience to clients - allowing a client to quickly activate/deactivate a configuration by adding/removing associations - or as an intermediate state between multiple `ExposePath` or `HidePath` requests. It is not mandatory in SMI-S to support these partial path states, but clients need to understand which partial path states are and are not valid.

18.1.2 Model Elements

The model uses three basic types of objects:

LogicalDevice, the superclass of volumes and tape drives representing SCSI logical units

SCSIProtocolController - models the “view” described above.

SCSIProtocolEndpoint – models the SCSI protocol aspects of a port. A `SCSIProtocolEndpoint` is associated to one or more ports (modeled as subclasses of `LogicalPort`). `SCSIProtocolEndpoint` and classes (such as `FCPort`) representing ports are part of target port subprofiles.

These objects are related by two associations:

ProtocolControllerForUnit associates a `SCSIProtocolController` with its `LogicalDevices`; the controller-relative address (such as a SCSI Logical Unit Number) is modeled as the `DeviceNumber` property of `ProtocolControllerForUnit`.

SAPAvailableForElement associates a `SCSIProtocolController` to one or more `SCSIProtocolEndpoints`.

In this subprofile, the existence of a `ControllerConfigurationService` with a `ConcreteDependency` association to a `SCSIProtocolController` governs the high-level device mapping and masking policy for that protocol controller.

If the service does not exist, then regardless of host port, the policy is that **SAPAvailableForElement** associates `SCSIProtocolController` to all `SCSIProtocolEndpoints` that represent SCSI target behavior (that is, have `Role` property set to “Target”).

If the service is present, then for a particular host port, the policy is that **SAPAvailableForElement** connects a `SCSIProtocolController` to a `SCSIProtocolEndpoint` only when access is explicitly granted.

Figure 74 and Figure 75 depict an instance diagram of a generic storage system with dual-port access to four logical devices and an implementation with no device mapping and masking services. All of the `LogicalDevices` are exposed to all initiators with the same `DeviceNumber`. Figure 74 depicts a configuration with no LUN Masking capabilities.

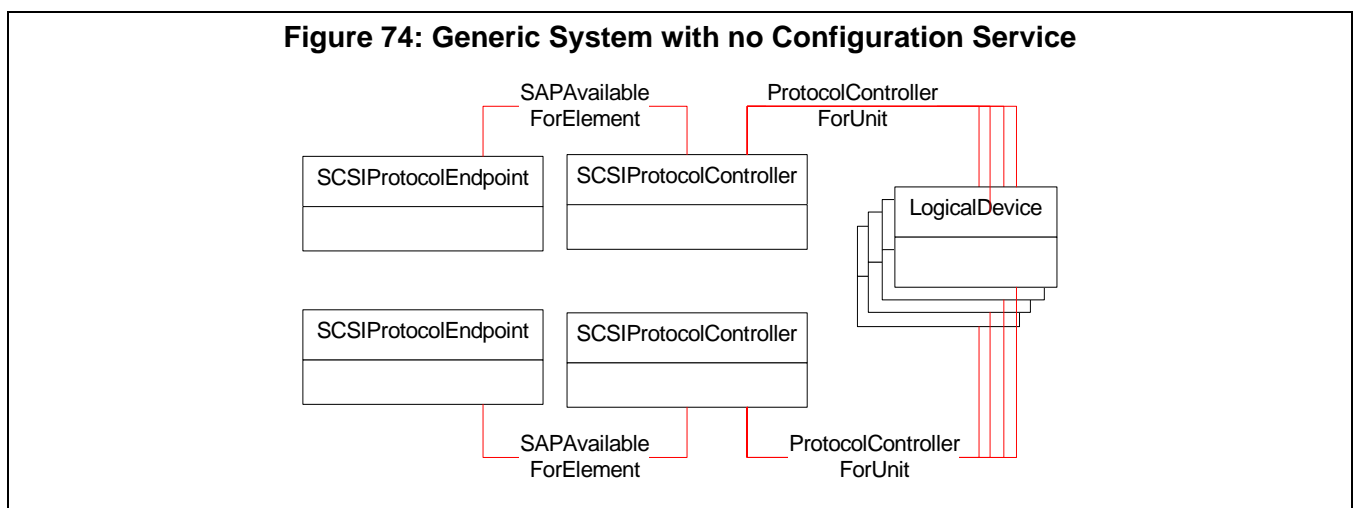
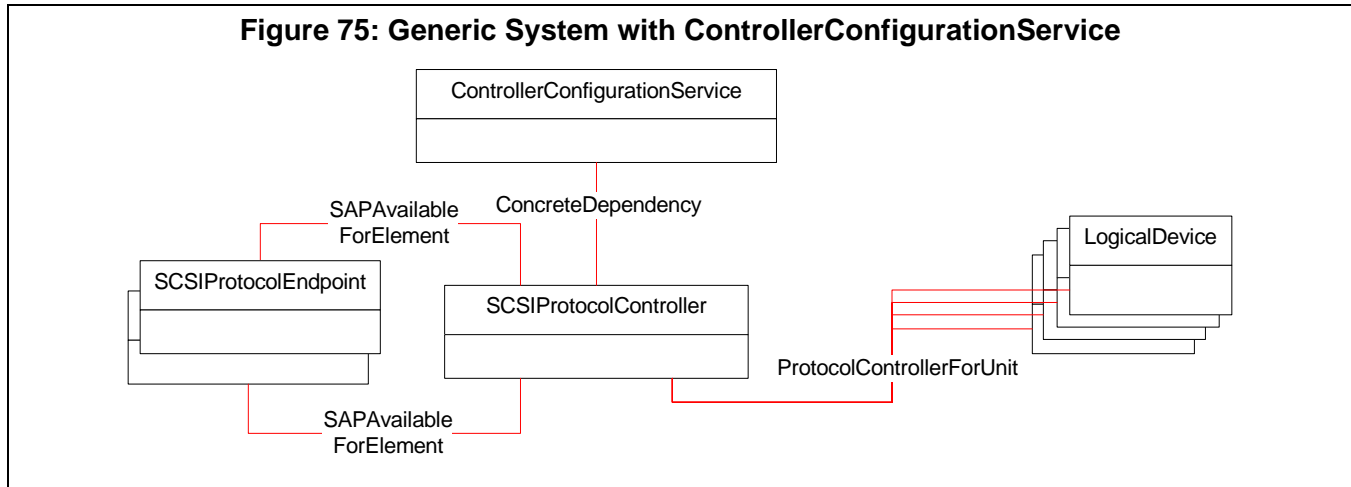


Figure 75 depicts the same configuration in an implementation with an `ControllerConfigurationService` defined. In this case, access to the `ProtocolController` is denied to each host port unless it is specifically granted access.



18.1.3 SCSIProtocolController Views

Device Masking limits the devices seen by particular host initiators (such as HBAs). For example, when a host discovers a device (using SCSI Report LUNs and Inquiry commands), it may see two of four LogicalDevices, other hosts may see no LogicalDevices, and yet other hosts may only see LogicalDevices through a subset of target ports.

Device Mapping allows the same LogicalDevice to be assigned different DeviceNumber (LUN) as seen by different host HBAs. This would allow each of four LogicalDevices to appear to be Logical Unit zero to four different hosts.

An initiator sees a single view (SCSIProtocolController) through a target port. This view includes LogicalDevices explicitly exposed to specified initiators and “default access” LogicalDevices (that are exposed to all initiators).

An administrator can use the `ControllerConfigurationService` interfaces to create “views” (SCSIProtocolControllers) of a storage system – each view exposes a subset of components that are intended to behave as a cohesive subset. In particular, a view:

- is associated with a set of LogicalDevices;
- may be exposed to zero or more host ports;
- is associated with one or more target device ports;
- shall not be exposed through a particular host / target port pair that is in use by another view. (In other words, a view corresponds to the logical unit inventory provided by SCSI REPORT LUNS and INQUIRY commands).

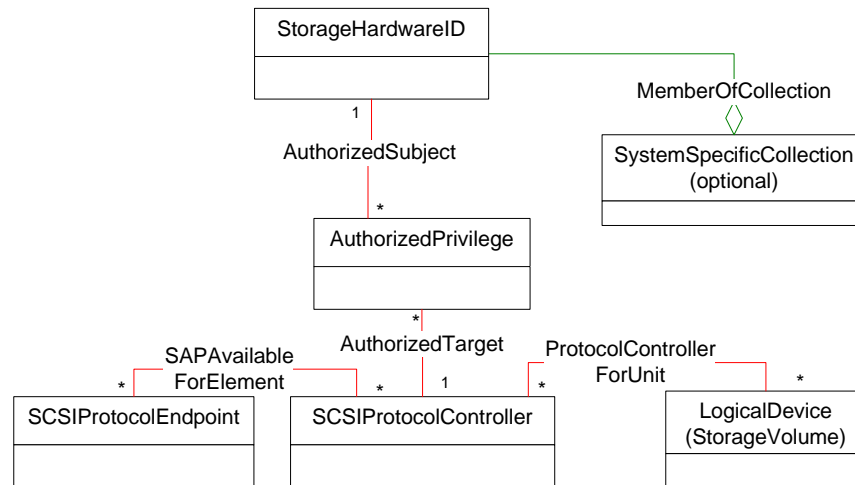
For systems where access is granted through all or no target ports (where `ProtocolControllerMaskingCapabilities.PortsPerView` is set to “All Ports share the same View”), this rule is simpler – an initiator `StorageHardwareID` shall not be associated with more than one view (SCSIProtocolController).

- each LogicalDevice in a view shall have a unique DeviceNumber (SCSI logical unit number);
- a LogicalDevice may be in multiple views, and in each may be assigned the same or different DeviceNumbers (Logical Units);

The device uses the initiator port identifier to authorize access and to determine the view to present to the HBA. The initiator ID (such as FC Port WWN) is modeled as a subclass of `Identity` called `StorageHardwareID`. As used in this subprofile, `AuthorizedSubject` associates a `AuthorizedPrivilege` with a `StorageHardwareID`. As used in this subprofile, `AuthorizedTarget` associates an `AuthorizedPrivilege` with a `SCSIProtocolController`.

In this version of the subprofile, there is exactly a one-to-one-to-one relationship between AuthorizedSubject, AuthorizedPrivilege, and AuthorizedTarget. In other words, for each StorageHardwareID associated to a SCSIProtocolController, there will be unique instances of AuthorizedSubject, AuthorizedPrivilege, and AuthorizedTarget

Figure 76: Relationship of Initiator IDs, Endpoints, and Logical Units



18.1.4 Initiator ID Collections

An implementation may optionally model collections of Initiator IDs. This is modeled as depicted in Table 76. If the implementation supports collection of initiator IDs, the instrumentation shall set ProtocolControllerMaskingCapabilities.ProtocolControllerSupportsCollections to True

18.1.5 Default View / Default Logical Unit Access

An implementation may expose some logical units to all initiators while restricting access to others. A default LUN exposes the same SCSI logical unit to all initiators, so adding a default LUN requires that the instrumentation assure that no existing logical-unit-view map uses that same logical unit address. Whenever a new SCSIProtocolController is created, it is automatically attached to all default LUNs

This is modeled with a SCSIProtocolController that is associated via AuthorizedTarget to a AuthorizedPrivilege that is associated via AuthorizedSubject to a StorageHardwareID with an Name property set to null (not the zero-length string ""). These are known as **default protocol controllers** - exposing a view that is granted by default to all initiators, regardless of masking rules. If the implementation supports default protocol controllers, the instrumentation shall instantiate at least one default protocol controller when the instrumentation starts. The instrumentation shall reject any client attempt to delete a default protocol controller.

Only one null-name StorageHardwareID is allowed. It is associated to all default SPCs. No other StorageHardwareIDs may be associated to default SPCs. A target port can be associated with at most one default SPC.

If ProtocolControllerMaskingCapabilities.PortsPerView is not set to "All Ports share the same View", the instrumentation may support multiple default protocol controllers, but a target port shall not be associated to more than one default protocol controller.

A client requests a logical unit be given default access by associating with the default protocol controller using ExposeDefaultLUs method. The instrumentation shall ensure that the requested unit number is not used in any SCSIProtocolController connected to target ports associated with the default protocol controller. If the unit number is available, the logical unit is attached to the default protocol controller and all the other protocol controllers that

share its target ports. Similarly, a client requests default access be removed from a logical unit by calling `HideDefaultLUs`, passing in a reference to the default protocol controller and the logical unit's ID.

18.1.6 Arbitrary Logical Units

If the implementation supports logical units for management (rather than storage), they shall be modeled with `SCSIArbitraryLogicalUnit`. If these management units are exposed regardless of masking access then they shall be associated to the default protocol controller.

18.1.7 Read-only verses Read-Write access

`ExposePaths` includes a `DeviceAccess` parameter that is used to set the `DeviceAccess` property of `ProtocolControllerForUnit` association.

18.1.8 Read-Only Volumes

An implementation may model a volume that is readable, but not writable to any initiator by setting `StorageVolume.Access` to "Readable" (1).

18.1.9 Finding Volumes that are not Mapped

A `StorageVolume` is considered mapped if it is exposed to an initiator. Instrumentation shall inform clients whether a volume is or is not mapped using the "In-Band Access Granted" value in `StorageVolume.ExtentStatus` array property. If a volume is associated with one or more protocol controllers and one of the associated protocol controllers is associated with one or more `StorageHardwareIDs`, the instrumentation shall set "In-Band Access Granted" in `ExtentStatus`. Otherwise, "In-Band Access Granted" shall not be set.

18.1.10 Limits on Map counts per Logical Unit

`ProtocolControllerMaskingCapabilities.MaximumMapCount` is the maximum number of times the underlying implementation allows a logical unit to be mapped (in other words, the maximum number of `ProtocolControllerForUnit` associations that can be associated to the logical unit represented by the `LogicalDevice` subclass. The instrumentation sets this to 0 if it has no limit.

18.1.11 Deactivated Logical Units

Instrumentation may describe inaccessibility of a logical unit through a path using `ProtocolControllerForUnit.AccessState`. This property may be read, but not written by clients. Possible values are Active, Inactive, "Replication In Progress", and "Mapping Inconsistency".

Since default protocol controllers were not defined in SMI-S 1.0, a client could have created a configuration that does not comply with the SMI-S 1.1.0 semantics (which are intended to mimic SCSI's). Similarly, a non-compliant configuration could have been created using non-SMI-S interfaces. Instrumentation may set `AccessState` to "Mapping Inconsistency" to express these states. A client request to set a valid mapping configuration using `ExposePaths` should clear this state and reset `AccessState` to Active.

18.1.12 SCSIProtocolController Properties**Table 232: SCSIProtocolController Property Description**

Property	Description	Impact on ExposePaths (see 1)	Impact on HidePaths
SPCAllowsNoLUs	It is valid to have no LogicalDevices associated with an SPC	If true, LUNames, DeviceNumbers, and DeviceAccess may be null. If false, LUNames and DeviceAccesses shall be non-null; DeviceNumbers depends on ClientSelectableDeviceNumbers	If true, then all associated LogicalDevices may be specified in LUNames. If false and client specifies names of all associated LUs in LUNames, then see 2
SPCAllowsNoTargets	It is valid to have no target ports associated with an SPC	If true, TargetPortIDs may be null. If false, TargetPortIDs shall be non-null.	If true, then all associated target ports may be specified in TargetPortIDs. If false, and client specifies names of all associated target ports in TargetPortIDs, then see 2
SPCAllowsNoInitiators	It is valid to have no initiator port IDs associated with an SPC	If true, InitiatorPortIDs may be null. If false, InitiatorPortIDs shall be non-null.	If true, then all associated initiator port IDs may be specified in InitiatorPortIDs. If false, and client specifies names of all associated initiator port IDs in InitiatorPortIDs, then see 2
<p>1. This only applies to the "Create a new view" use case for ExposePaths</p> <p>2. The result of this HidePaths request would be an invalid partial path state; therefore, the instrumentation shall delete the SPC and all its associations.</p>			

There are two clarifications to the property descriptions in Figure 232. If the implementation supports partial path SPCs, the intrinsic DeleteInstance is used to delete an SPC with no full paths. If DeleteInstance is called to delete an SPC with full paths, the instrumentation shall return CIM Error with CIM_ERR_FAILED status code.

18.1.13 Initiator Setting Data

Some storage systems allow a customer (or host-side agent) to provide information about OS hosting initiators. The storage system uses this information to provide OS-specialized behavior (for example, SCSI responses). Being able to identify the OS-specific operating mode ("host mode") of an element (i.e., FCPort or

SCSIProtocolController) is essential because there are variances in SCSI communications between different operating systems or even different versions of the same operating system, and having the incorrect “host mode” will cause operations to have degraded performance or even fail. This information is modeled as StorageClientSettingData. StorageClientSettingData.ClientTypes[] is an array of OS names. This array property allows a single StorageClientSettingData instance to apply to multiple OS Types. The StorageClientSettingData instances shall be scoped to a particular ComputerSystem because a CIM server hosting multiple devices will need to distinguish the valid StorageClientSettingData instances for one array from another.

The instrumentation should provide a meaningful name for each StorageClientSettingData instance; typically this will be names already exposed via existing management tools and documentation.

StorageClientSettingData instances are not created by clients; any storage system that provides OS type behavior advertises these instances (via EnumerateInstance and GetInstance) and associates them (using ElementSettingData) with elements previous configured with the setting behavior.

A client can associate StorageHardwareIDs to a StorageClientSettingData instance (when a customer or host agent maps an initiator to an OS type). This is done by specifying the Setting parameter to CreateStorageHardwareID). A client can also associate an StorageClientSettingData instance to a storage system element (such as a Port, a SCSIProtocolController, or a StorageVolume) to request that this element exhibit the setting-specific behavior. This is done by creating a new ElementSettingData association from the element to the StorageClientSettingData instance using the intrinsic CreateInstance method. If any ElementSettingData association between the element and a StorageClientSettingData instance already exists, it shall be deleted by the client before calling CreateInstance. Figure 77 provides an example.

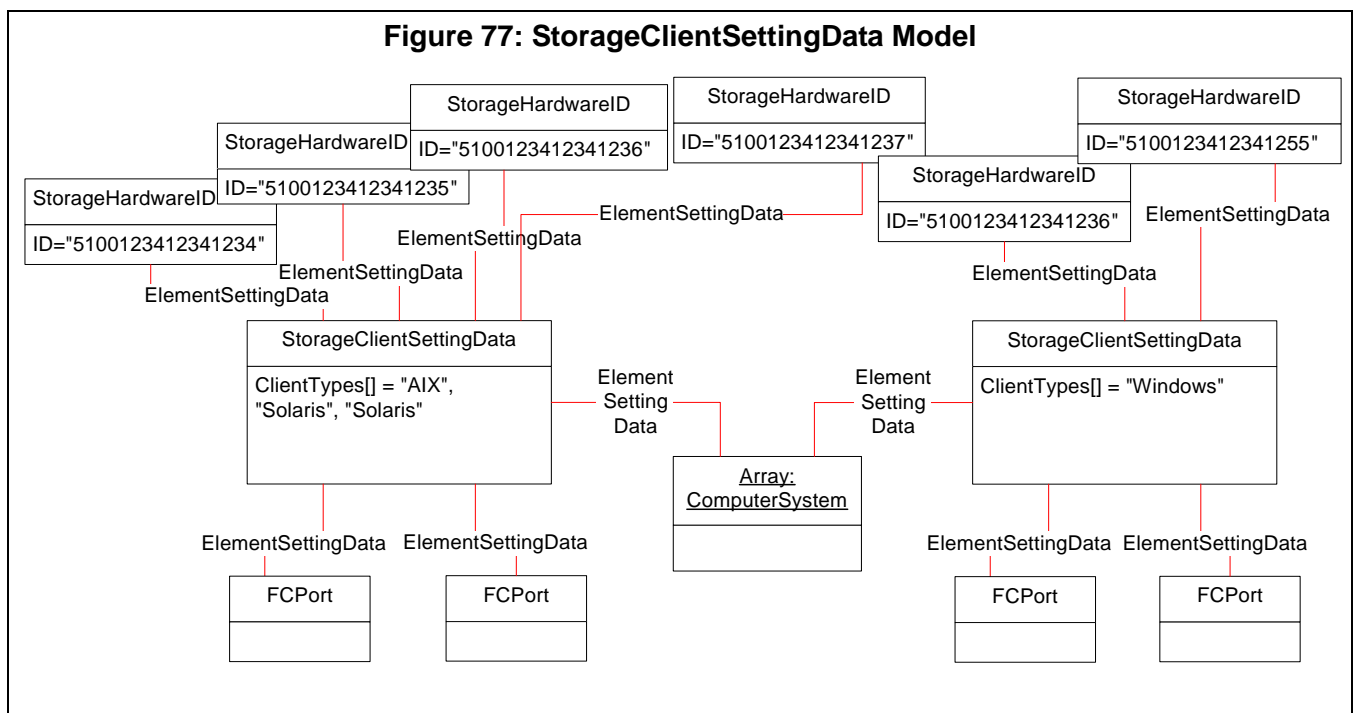
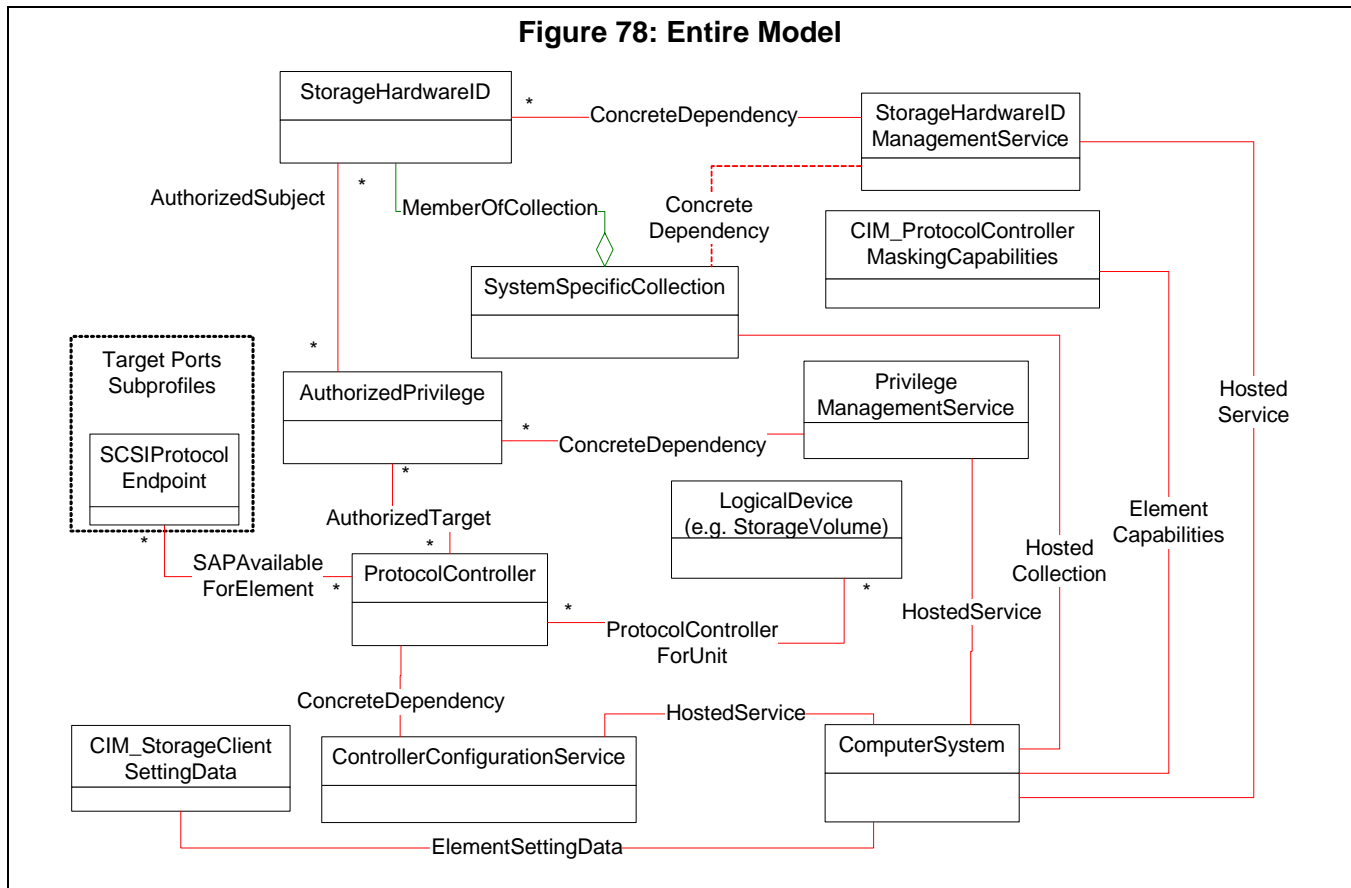


Figure 78 depicts the entire model.



18.1.14 Durable Names and Correlatable IDs of the Profile

The Masking and Mapping subprofile uses the durable names/correlatable ID for logical devices as defined by the parent profile.

18.1.15 Instrumentation Requirements

If a PrivilegeManagementService is not present, then all access is assumed. If an PrivilegeManagementService is present, then access shall be specifically granted.

A LogicalDevice may have ProtocolControllerForUnit associations to multiple SCSIProtocolControllers - this models a device shared by different subject sets.

Clients may need to know the range of possible unit numbers supported by a storage system. The agent should set `SCSIProtocolController.MaxUnitsControlled`.

EXPERIMENTAL

The two `SNIA_ProtocolControllerMaskingCapabilities` properties (`SupportedSynchronousMethods` and `SupportedAsynchronousMethods`) describe the methods that are supported by the instrumentation. These enumerations indicate what operations will be executed as asynchronous jobs or synchronously. If an operation is included in both, then the underlying implementation is indicating that it may or may not create a job. If an operation is not included in either, then the instrumentation does not implement that method. If an instrumentation does not support all of the methods as defined by this subprofile, these properties can help a client determine if there is

sufficient support to manage masking and mapping. Any instrumentation that does not support the required methods of this subprofile shall not be considered compliant even if these properties are supported.

18.1.16 Element Naming

The name of a ProtocolController, StorageHardwareID, GatewayPathID, or SystemSpecificCollection may be changed. The existence of the EnabledLogicalElementCapabilities instance associated to the element indicates that the element can be named. If ElementNameEditSupported is set to TRUE, then the ElementName of the associated element name may be modified. The ElementNameMask property provides the regular expression that expresses the limits of the name; see the class definition for EnabledLogicalElementCapabilities for details for this property.

However, this model does not indicate which element names can be used in the creation or modification of the element through the corresponding service methods if there no elements of a given type. To do this, the instrumentation shall support a single EnabledLogicalElementCapabilities for each element type. There shall be a single mask for each storage element type as well. Each of these instances shall be associated to the appropriate service as described in the table below.

Table 233: Element to Service Mapping

Element	Service	Created by Service Method
GatewayPathID	StorageHardwareIDManagementService	CreateGatewayPathID
StorageHardwareID	StorageHardwareIDManagementService	CreateStorageHardwareID
StorageHardwareID	ControllerConfigurationService	ExposePaths
SystemSpecificCollection	StorageHardwareIDManagementService	CreateHardwareIDCollection
ProtocolController	ControllerConfigurationService	ExposePaths

The EnabledLogicalElementCapabilities associated to an element shall have the ElementName property set as per Table 234.

Table 234: Element to Element Name Mapping

Element	EnabledLogicalElementCapabilities.ElementName
GatewayPathID	GatewayPathID Enabled Capabilities
StorageHardwareID	StorageHardware Enabled Capabilities
SystemSpecificCollection	SystemSpecificCollection Enabled Capabilities
ProtocolController	ProtocolController Enabled Capabilities

If the implementation supports the creation or modification of a given element type and the modification of the name of the storage element, then it shall produce an EnabledLogicalElementCapabilities instance for each of those elements.

If an storage element's name is modifiable through once of the aforementioned service methods, it shall also be modifiable through instance modification. However, a storage element's name may be modifiable through instance modification but storage element modification may not be allowed through these service methods.

EXPERIMENTAL

18.2 Health and Fault Management Considerations

None.

18.3 Cascading Considerations

None.

18.4 Supported Subprofiles, and Packages

Table 235: Supported Profiles for Masking and Mapping

Registered Profile Names	Mandatory	Version
Array	Yes	1.2.0
Generic Target Ports	Yes	1.2.0
Block Services	Yes	1.2.0

18.5 Methods of the Profile

18.5.1 ExposePaths

ExposePaths is used in place of the AssignAccess and AttachDevice methods used in 1.0. The problem with these methods was that they required the clients to perform multiple transactions to achieve a valid view. This forced providers to maintain invalid view states while waiting for the client to complete a sequence of transactions. This also created non-interoperability when the providers only supported transactions in a certain order, and when a second client looked at the model before a sequence of transactions was completed.

ExposePaths performs the mapping and masking operation in one method call. It exposes a list of SCSI logical units (such as RAID volumes or tape drives) to a list of initiators through a list of target ports, through one or more SCSIProtocolControllers (SPCs). Support for the 1.0 equivalent functionality is available by passing in an existing SCSIProtocolController.

There are two modes of operation, create and modify. If a NULL value is passed in for the SPC, then the instrumentation will create at least one SPC that satisfies the request. Depending upon the instrumentation capabilities, more than one SPC may be created. (e.g. if ProtocolControllerMaskingCapabilities.OneHardwareIDPerView is true and more than one initiatorID was passed in, then one SPC per initiatorID will be created). If an SPC is passed in, then the instrumentation attempts to add the new paths to the existing SPC. Depending upon the instrumentation capabilities, this may result in the creation of additional SPCs. The instrumentation shall return an error if honoring this request would violate SCSI semantics.

For creating an SPC, the parameters that need to be specified are dependent upon the SPCAllows* properties in ProtocolControllerMaskingCapabilities. If SPCAllowsNoLUs is false, the caller shall specify a list of LUNames. If it is true, the caller may specify a list of LUNames or may pass in null. If SPCAllowsNoTargets is false and PortsPerView is not 'All Ports share the same view' the caller shall specify a list of TargetPortIDs. If it is true, the caller may specify a list of TargetPortIDs or may pass in null. If SPCAllowsNoInitiators is false, the caller shall specify a list of InitiatorPortIDs. If it is true, the caller may specify a list of InitiatorPortIDs or may pass in null. If LUNames is not null, the caller shall specify the DeviceAccess for each logical unit. If the provider's ProtocolControllerMaskingCapabilities ClientSelectableDeviceNumbers property is TRUE then the client shall

either provide a list of device numbers (LUNs) to use for the paths to be created or pass in NULL. If is false, the client shall pass in NULL for this parameter.

The LUNames, DeviceNumbers, and DeviceAccesses parameters are mutually indexed arrays - any element in DeviceNumbers or DeviceAccesses will set a property relative to the LogicalDevice instance named in the corresponding element of LUNames. LUNames and DeviceAccesses shall have the same number of elements. DeviceNumbers shall be null (asking the instrumentation to assign numbers) or have the same number of elements as LUNames. If these conditions are not met, the instrumentation shall return a 'Invalid Parameter' status.

For modifying an SPC, there are three specific use cases identified. The instrumentation shall support these use cases. Other permutations are allowed, but are vendor-specific. The use cases are: Add LUs to a view, Add initiator IDs to a view, and Add target port IDs to a view.

Add LUs to a view requires that the LUNames parameter not be null and that the InitiatorIDs and TargetPortIDs parameters be null. DeviceNumbers may be null if ClientSelectableDeviceNumbers is false. DeviceAccess shall be specified.

Add initiator IDs to a view requires that the LUNames parameter be null, that the InitiatorIDs not be null, and that the TargetPortIDs parameters be null. DeviceNumbers and DeviceAccess shall be null.

Add target port IDs to a view requires that the LUNames and InitiatorPortIDs parameters be null and is only possible if PortsPerView is 'Multiple Ports Per View'. DeviceNumbers and DeviceAccess shall also be null.

If a client calls ExposePaths specifying logical units already associated to the SPC and specifies different DeviceNumber or DeviceAccess values, the instrumentation shall change these properties in the appropriate ProtocolControllerForUnit instance(s).

When calling ExposePaths where an entry (e.g., LogicalDevice) does not exist, then ExposePaths shall fail and report an error.

There are four valid use cases for ExposePaths - create plus the three modify use cases above. These four use cases and the requirements for parameters are summarized in Table 236.

Table 236: ExposePath Use Cases

parameters /use cases	LUNames	InitiatorPortIDs	TargetPortIDs	DeviceNumbers	DeviceAccesses	ProtocolControllers (on input)
Create a new view	See 1)	See 1)	See 1) See 2)	See 3)	Mandatory, see 4)	NULL
Add LUs to a view	Mandatory	NULL	NULL	See 3)	Mandatory, see 4)	contains a single SPC ref
Add initiator IDs to a view (see 5)	NULL	Mandatory	NULL	NULL	NULL	contains a single SPC ref
Add target port IDs to a view (see 6)	NULL	NULL	Mandatory	NULL	NULL	contains a single SPC ref
Vendor-specific	As long as all the previous use cases are implemented, the instrumentation may support other vendor-specific combinations of parameters.					

Table 236: ExposePath Use Cases (Continued)

parameters /use cases	LUNames	InitiatorPortIDs	TargetPortIDs	DeviceNumbers	DeviceAccesses	ProtocolControllers (on input)
1. Dependent on values of new SPCAllowsNo* capability properties described below 2. If PortsPerView is "All ports share same view", TargetPortIDs parameter shall be null. 3. If ClientSelectableDeviceNumbers is true, shall either be null or have same number of elements as LUNames. If ClientSelectableDeviceNumbers is false, shall be null. 4. Shall have same number of elements as LUNames 5. Only valid if OneHardwareIDPerView is false 6. Only valid if PortsPerView is "Multiple Ports per View"						

The relevant rules of SCSI semantics are:

- an SPC shall not be exposed through a particular host/target port pair that is in use by another SPC. (In other words, an SPC and its associated logical units and ports together correspond to the logical unit inventory provided by SCSI REPORT LUNS and INQUIRY commands)

- each LogicalDevice associated to an SPC shall have a unique ProtocolControllerForUnit DeviceNumber (logical unit number)

The instrumentation shall report an error if the client request would violate one of these rules.

If the instrumentation provides PrivilegeManagementService, the results of setting DeviceAccesses shall be synchronized with PrivilegeManagementService as described in the ProtocolControllerForUnit DeviceAccess description.

18.5.1.1 Uint32 ExposePaths

OUT CIM_ConcreteJob REF Job

Reference to the job (may be null if no job started)

IN string LUNames[]

An array of IDs of logical unit instances. The LU instances need to already exist. The members of this array shall match the Name property of LogicalDevice instances that represent SCSI logical units. See the method description for conditions where this parameter may be null.

IN string InitiatorPortIDs[]

IDs of initiator ports. If existing StorageHardwareID instances exist, they shall be used. If no StorageHardwareID instance matches, then one is implicitly created. See the method description for conditions where this may be null.

IN string TargetPortIDs[]

IDs of target ports. See the method description for conditions where this may be null.

IN string DeviceNumber[]

A list of logical unit numbers to assign to the corresponding logical unit in the LUNames parameter. (within the context of the elements specified in the other parameters). If the LUNames parameter is null, then this parameter shall be null. Otherwise, if this parameter is null, all LU numbers are assigned by the hardware or instrumentation. This shall be formatted as unseparated uppercase hexadecimal digits, with no leading "0x".

IN uint16 DeviceAccess[]

A list of permissions to assign to the corresponding logical unit in the LUNames parameter. This specifies the permission to assign within the context of the elements specified in the other parameters. Setting this to 'No Access' assigns the DeviceNumber for the associated initiators, but does not grant read or write access. If the LUNames parameter is not null then this parameter shall be specified.

IN/OUT CIM_SCSIProtocolController REF ProtocolControllers[]

An array of references to SCSIProtocolControllers (SPCs). On input, this can be null, or contain exactly one element; if null on input, the instrumentation will create one or more new SPC instances.

On output, this will be either null (if a job was created) or the set of SPCs affected (those created or modified). or those having some part of the 'view' modified, e.g. such as association being created or an AuthorizedPrivilege being created). If a job was started, references to the SPCs affected will be found by following the AffectedJobElement association from the job.

18.5.2 HidePaths

HidePaths is used in place of the HideAccess and DetachDevice methods used in 1.0. The problem with these methods is the same as AssignAccess and AttachDevice, in that they required the clients to perform multiple transactions to achieve a valid view. This forced providers to maintain invalid view states while waiting for the client to complete a sequence of transactions. This also created non-interoperability when the providers only supported transactions in a certain order, and when a second client looked at the model before a sequence of transactions was completed.

HidePaths is the inverse of ExposePaths. It hides a list of SCSI logical units (such as RAID volumes or tape drives) from a list of initiators through a list of target ports, through one or more SCSIProtocolControllers (SPCs). Support the 1.0 equivalent functionality is available by passing in an existing SCSIProtocolController.

When hiding logical units, there are three specific use cases identified. The instrumentation shall support these use cases. Other permutations are allowed, but are vendor-specific. The use cases are: Remove LUs from a view, Remove initiator IDs from a view, and Remove target port IDs from a view.

Remove LUs from a view requires that the LUNames parameter not be null and that the InitiatorIDs and TargetPortIDs parameters be null.

Remove initiator IDs from a view requires that the LUNames parameter be null, that the InitiatorIDs not be null, and that the TargetPortIDs parameters be null.

Remove target port IDs from a view requires that the LUNames and InitiatorPortIDs parameters be null.

The disposition of the SPC when the last logical unit, initiator ID, or target port ID is removed depends upon the ProtocolControllerMaskingCapabilites SPCAllowsNo* properties. If SPCAllowsNoLUs is false, then the SPC is automatically deleted when the last logical unit is removed. If SPCAllowsNoTargets is false, then the SPC is automatically deleted when the last target port ID is removed. If SPCAllowsNoInitiators is false, then the SPC is automatically deleted when the last initiator port ID is removed. In all other cases, the SPC needs to be explicitly deleted via the DeleteInstance intrinsic function or via the DeleteProtocolController method. The use cases for HidePaths() are summarized in Table 237.

Table 237: HidePaths Use Cases

Parameters/use cases	LUNames	InitiatorPortIDs	TargetPortIDs	ProtocolController (on input) see 1
Remove LUs from a view	Mandatory	NULL	NULL	contains a single SPC ref
Remove initiator IDs from a view	NULL	Mandatory	NULL	contains a single SPC ref

Table 237: HidePaths Use Cases (Continued)

Parameters/use cases	LUNames	InitiatorPortIDs	TargetPortIDs	ProtocolController (on input) see 1
Remove target ports from a view (see 2)	NULL	NULL	Mandatory	contains a single SPC ref
Hide full paths from a view	Mandatory	Mandatory	Mandatory	contains a single SPC ref
Vendor-specific	As long as all the previous use cases are implemented, the instrumentation may support other vendor-specific combinations of parameters.			
1. On output, the provider returns a list of refs to SPCs that have been affected (those created or modified or those having some part of the 'view' modified, e.g. such as association being created or deleted an AuthorizedPrivilege being created or deleted).				
2. Only valid if PortsPerView is "Multiple Ports per View"				

When calling HidePaths where the Port, SPC, StorageHardwareID, or StorageVolume exist, but the association(s) that are being modified don't exist (e.g. calling HidePaths for a volume that is not currently exposed), then HidePaths may return success. The rationale for returning success is the net result of the operation is the same whether or not the association exists, so it is not necessarily considered an error

However, when calling HidePaths where an entry (e.g. Port) does not exist, then HidePaths shall return an error. The difference between this and the above case is that the above has just a connection between instances missing, while this case has an actual instance missing. The net result of the HidePaths operation would be different because HidePaths does not delete the instance (with the exception of the AuthorizedPrivilege), just the association between instances.

18.5.2.1 uint32 HidePaths

OUT CIM_ConcreteJob REF Job

Reference to the job (may be null if no job started)

IN string LUNames[]

An array of IDs of logical unit instances. The LU instances need to already exist. See the method description for conditions where this parameter may be null.

IN string InitiatorPortIDs[]

IDs of initiator ports. See the method description for conditions where this may be null.

IN string TargetPortIDs[]

IDs of target ports. See the method description for conditions where this may be null.

IN/OUT CIM_SCSIProtocolController REF ProtocolControllers[]

An array of references to SCSIProtocolControllers (SPCs). On input, this can be null, or contain exactly one element. The instrumentation will attempt to remove associations (LUNames, InitiatorPortIDs, or TargetPortIDs) from this SPC. Depending upon the specific implementation, the instrumentation may need to create new SPCs with a subset of the remaining associations.

On output, this will be either null (if a job was created) or the set of SPCs affected (those created or modified). If a job was started, references to the SPCs affected will be found by following the AffectedJobElement association from the job.

18.5.3 ExposeDefaultLUs

ExposeDefaultLUs is similar to ExposePaths, except ExposeDefaultLUs works with 'default view' SPCs. The 'default view' SPC exposes logical units to all initiators. This SPC is identified by an association to a StorageHardwareID with Name property set to the empty string. ExposeDefaultLUs exposes a list of SCSI logical units (such as RAID volumes or tape drives) through a 'default view' SCSIProtocolController (SPC) through a list of target ports.

As with ExposePaths, there are two modes of operation, create and modify. If a NULL value is passed in for the SPC, then the instrumentation will attempt to create a new default view. If PortsPerView is 'All Ports share the same view', then there is at most one default view SPC. If PortsPerView is not 'All Ports share the same view', then there may be multiple default view SPCs as long as different ports are associated with each. If an SPC is passed in, then the instrumentation adds the new paths to the existing SPC. The instrumentation may return an error if honoring this request would violate SCSI semantics.

For creating a default view SPC, the parameters that need to be specified are dependent upon the SPCAllows* properties in ProtocolControllerMaskingCapabilities. If SPCAllowsNoLUs is false, the caller shall specify a list of LUNames. If it is true, the caller may specify a list of LUNames or may pass in null. If SPCAllowsNoTargets is false, the caller shall specify a list of TargetPortIDs. If it is true, the caller may specify a list of TargetPortIDs or may pass in null. If LUNames is not null, the caller shall specify the DeviceAccess for each logical unit. If the provider's ProtocolControllerMaskingCapabilities ClientSelectableDeviceNumbers property is TRUE then the client shall either provide a list of device numbers (LUNs) to use for the paths to be created or pass in NULL. If it is false, the client shall pass in NULL for this parameter.

The LUNames, DeviceNumbers, and DeviceAccesses parameters are mutually indexed arrays - any element in DeviceNumbers or DeviceAccesses will set a property relative to the LogicalDevice instance named in the corresponding element of LUNames. LUNames and DeviceAccesses shall have the same number of elements. DeviceNumbers shall be null (asking the instrumentation to assign numbers) or have the same number of elements as LUNames. If these conditions are not met, the instrumentation shall return a 'Invalid Parameter' status.

For modifying an SPC, there are two specific use cases identified. The instrumentation shall support one and the other is required depending on how a property is set. Other permutations are allowed, but are vendor-specific.

The required use case is - Add LUs to a default view. Add LUs to a default view requires that the LUNames parameter not be null and that the TargetPortIDs parameters be null. DeviceNumbers may be null if ClientSelectableDeviceNumbers is false. DeviceAccess shall be specified.

Add target port IDs to a default view is only valid if PortsPerView is set to 'Multiple Ports per View'. It requires that the LUNames, DeviceNumbers, and DeviceAccess shall also be null. The use cases for ExposeDefaultLUs() are summarized in Table 238.

Table 238: Use Cases for ExposeDefaultLUs

Parameters /use cases	LUNames	TargetPort IDs	Device-Numbers	Device-Accesses	ProtocolControllers (on input)
Create a new default view (see 1)	See 2)	See 2)	See 3)	Mandatory, see 4)	Shall be null
Add LUs to a view	Mandatory	Shall be null	See 3)	Mandatory, see 4)	Shall contain a single SPC ref
Add target port IDs to a view (see 5)	Shall be null	Mandatory	Shall be null	Shall be null	Shall contain a single SPC ref
Vendor-Specific	As long as all the previous use cases are implemented, the instrumentation may support other vendor-specific combinations of parameters.				
<div>1. Only valid if PortsPerView is not "All Ports share the same View"</div> <div>2. Dependent on values of SPCAllows* capability properties described above</div> <div>3. If ClientSelectableDeviceNumbers is true, shall either be null or have same number of elements as LUNames. If ClientSelectableDeviceNumbers is false, shall be null.</div> <div>4. Shall have same number of elements as LUNames</div> <div>5. Only valid if PortsPerView is "Multiple Ports per View"</div>					

The relevant rules of SCSI semantics are:

- an SPC shall be exposed through a particular host/target port pair that is in use by another SPC. (In other words, an SPC and its associated logical units and ports together correspond to the logical unit inventory provided by SCSI REPORT LUNS and INQUIRY commands)
- each LogicalDevice associated to an SPC shall have a unique ProtocolControllerForUnit DeviceNumber (logical unit number)

The instrumentation shall report an error if the client request would violate one of these rules.

If the instrumentation provides PrivilegeManagementService, the results of setting DeviceAccesses shall be synchronized with PrivilegeManagementService as described in the ProtocolControllerForUnit DeviceAccess description.

If the instrumentation supports ExposeDefaultLUs then it shall also support HideDefaultLUs.

18.5.3.1 uint32 ExposeDefaultLUs

OUT CIM_ConcreteJob REF Job

Reference to the job (may be null if no job started)

IN string LUNames[]

An array of IDs of logical unit instances. The LU instances shall already exist. The members of this array shall match the Name property of LogicalDevice instances that represent SCSI logical units. See the method description for conditions where this parameter may be null.

IN string TargetPortIDs[]

IDs of target ports. See the method description for conditions where this may be null.

IN string DeviceNumber[]

A list of logical unit numbers to assign to the corresponding logical unit in the LUNames parameter. (within the context of the elements specified in the other parameters). If the LUNames parameter is null, then this parameter shall be null. Otherwise, if this parameter is null, all LU numbers are assigned by the hardware or instrumentation. Each element shall be formatted as unseparated uppercase hexadecimal digits, with no leading "0x".

IN uint16 DeviceAccess[]

A list of permissions to assign to the corresponding logical unit in the LUNames parameter. This specifies the permission to assign within the context of the elements specified in the other parameters. Setting this to 'No Access' assigns the DeviceNumber for the associated initiators, but does not grant read or write access. If the LUNames parameter is not null then this parameter shall be specified.

IN/OUT CIM_SCSIProtocolController REF ProtocolControllers[]

An array of references to SCSIProtocolControllers (SPCs). On input, this can be null, or contain exactly one element; there may be multiple references on output. If null on input, the instrumentation will create one or more new SPC instances.

On output, this will be either null (if a job was created) or the set of SPCs affected (those created or modified). If a job was started, references to the SPCs affected will be found by following the AffectedJobElement association from the job.

18.5.4 HideDefaultLUs

HideDefaultLUs is similar to HidePaths, except HideDefaultLUs works with 'default view' SPCs. The 'default view' SPC exposes logical units to all initiators. This SPC is identified by an association to a StorageHardwareID with Name property set to the empty string. HideDefaultLUs hides a list of SCSI logical units (such as RAID volumes or tape drives) through a 'default view' SCSIProtocolController (SPC) through a list of target ports.

HideDefaultLUs is the inverse of ExposeDefaultLUs. It hides a list of SCSI logical units (such as RAID volumes or tape drives) from a list of initiators through a list of target ports, through one or more SCSIProtocolControllers (SPCs).

When hiding logical units, there are two specific use cases identified. The use cases are: Remove LUs from a default view and Remove target port IDs from a default view. Remove LUs from a default view requires that the LUNames parameter not be null and that the TargetPortIDs parameter be null. Remove target port IDs from a default view is required if PortsPerView is Multiple Ports per view. It requires that the LUNames parameter be null.

The instrumentation shall support the Remove LUs case and shall support the remove target port IDs if PortsPerView is set to 'Multiple Ports per View'. Other permutations are allowed, but are vendor-specific.

If both LUNames and TargetIDs parameters are non-null and ProtocolControllerMaskingCapabilities.MaximumMapCount is 0, then the instrumentation shall create new SPCs and change associations as necessary to meet the client request and maintain the relevant rules of SCSI in the ExposeDefaultLUs description. If both LUNames and TargetIDs parameters are non-null and ProtocolControllerMaskingCapabilities.MaximumMapCount is greater than 0, then any client that cannot be

honored by changing associations to the specified SPC shall receive a 'Maximum Map Count Error' response. The use cases for HideDefaultLUs are summarized in Table 239

Table 239: Use Cases for HideDefaultLUs

parameters/ use cases	LUNames	TargetPortIDs	ProtocolController (on input)
Remove LUs from a default view	Mandatory	Shall be null	Mandatory
Remove target ports from a view (see 1)	Shall be null	Mandatory	Mandatory
Vendor-specific	As long as all the previous usecases are implemented, the instrumentation may support other vendor-specific combinations of parameters.		
1. Only valid if PortsPerView is "Multiple Ports per View"			

The disposition of the SPC when the last logical unit or target port ID is removed depends upon the ProtocolControllerMaskingCapabilites SPCAllows* properties. If SPCAllowsNoLUs is false, then the SPC is automatically deleted when the last logical unit is removed. If SPCAllowsNoTargets is false, then the SPC is automatically deleted when the last target port ID is removed. In all other cases, the SPC shall be explicitly deleted via the DeleteInstance intrinsic function.

If the instrumentation supports HideDefaultLUs then it shall also support ExposeDefaultLUs.

18.5.4.1 uint32 HideDefaultLUs

OUT CIM_ConcreteJob REF Job

Reference to the job (may be null if no job started)

IN string LUNames[]

An array of IDs of logical unit instances. The LU instances shall already exist. See the method description for conditions where this parameter may be null.

IN string TargetPortIDs[]

IDs of target ports. See the method description for conditions where this may be null.

IN/OUT CIM_SCSIProtocolController REF ProtocolControllers[]

An array of references to SCSIProtocolControllers (SPCs). On input, this shall contain exactly one element. The instrumentation will attempt to remove associations (LUNames or TargetPortIDs) from this SPC. Depending upon the specific implementation, the instrumentation may need to create new SPCs with a subset of the remaining associations.

On output, this will be either null (if a job was created) or the set of SPCs affected (those created or modified). If a job was started, references to the SPCs affected will be found by following the AffectedJobElement association from the job.

18.5.5 CreateStorageHardwareID

CreateStorageHardwareID creates a StorageHardwareID and the ConcreteDependency association between this service and the new StorageHardwareID.

18.5.5.1 Uint32 CreateStorageHardwareID(

IN string ElementName

The ElementName of the new StorageHardwareID instance.

IN string StorageID

StorageID is the value used by the SecurityService to represent identity - in this case, a hardware worldwide unique name.

IN Uint16 IDType

The type of the StorageID property.

IN string OtherIDType

The type of the storage ID, when IDType is 'Other'.

IN CIM_StorageClientSettingData REF Setting

REF to the StorageClientSettingData containing the OSType appropriate for this initiator. If left NULL, the instrumentation assumes a standard OSType - i.e., that no OS-specific behavior for this initiator is defined.

IN CIM_StorageHardwareID REF HardwareID

REF to the new StorageHardwareID instance.

18.5.6 DeleteStorageHardwareID

DeleteStorageHardwareID deletes a StorageHardwareID and the ConcreteDependency association between the ID and the service. If the StorageHardwareID still has associations to AuthorizedPrivilege instances (and thus to ProtocolControllers), then this method shall return an error. The reason is that deleting it without deleting the associations would cause an invalid model. Deleting the Association and AuthorizedPrivilege and SPC would be a very unexpected side effect. The client shall call HidePaths() first to delete these associations.

18.5.6.1 Uint32 DeleteStorageHardwareID

IN CIM_StorageHardwareID REF HardwareID

REF to the StorageHardwareID to delete

18.5.7 CreateHardwareIDCollection

Create a group of StorageHardwareIDs as a new instance of SystemSpecificCollection. This is useful to define a set of authorized subjects that can access volumes in a disk array. This method allows the client to make a request of a specific Service instance to create the collection and provide the appropriate class name. When these capabilities are standardized in CIM/WBEM, this method can be deprecated and intrinsic methods used. In addition to creating the collection, this method causes the creation of the HostedCollection association (to this service's scoping system) and MemberOfCollection association to members of the IDs parameter.

18.5.7.1 uint32 CreateHardwareIDCollection

IN string ElementName

The ElementName to be assigned to the created collection.

IN string HardwareIDs[]

Array of strings containing representations of references to StorageHardwareID instances that will become members of the new collection.

IN CIM_SystemSpecificCollection REF Collection

The new instance of SystemSpecificCollection that is created.

18.5.8 AddHardwareIDsToCollection

Create MemberOfCollection instances between the specified Collection and the StorageHardwareIDs. This method allows the client to make a request of a specific Service instance to create the associations. When these capabilities are standardized in CIM/WBEM, this method can be deprecated and intrinsic methods used.

18.5.8.1 uint32 AddHardwareIDsToCollection

IN string HardwareIDs[]

Array of strings containing representations of references to StorageHardwareID instances that will become members of the collection.

IN CIM_SystemSpecificCollection REF Collection

The Collection which groups the StorageHardwareIDs.

EXPERIMENTAL

18.5.9 DeleteProtocolController

DeleteProtocolController deletes the ProtocolController and all associations connected directly to this ProtocolController. It shall also delete any AuthorizedPrivilege instances associated to this ProtocolController as otherwise they would be left dangling. Since this subprofile does not have the notion of child ProtocolControllers, the DeleteChildrenProtocolControllers parameter shall be false. If the DeleteLogicalUnits parameter is True, the provider also deletes LogicalDevice instances associated via ProtocolControllerForUnit to this ProtocolController. LogicalDevice instances shall only be deleted when they are not part of any other ProtocolControllerForUnit associations. Whether or not the volumes may be deleted shall be determined by the instrumentation's support for the ReturnToStoragePool method in Block Services.

18.5.9.1 Uint32 DeleteProtocolController(

IN CIM_ProtocolController REF ProtocolController

ProtocolController to be deleted.

IN boolean DeleteChildrenProtocolControllers

If true, the management instrumentation provider will also delete 'child' ProtocolControllers (i.e., those defined as Dependent references in instances of AssociatedProtocolController where this ProtocolController is the Antecedent reference). Also, all direct associations involving the 'child' ProtocolControllers will be removed.

IN boolean DeleteUnits

If true, the management instrumentation provider will also delete LogicalDevice instances associated via ProtocolControllerForUnit, to this ProtocolController and its children. (Note that 'child' controllers will only be affected if the DeleteChildrenProtocolControllers input parameter is TRUE). LogicalDevice instances are only deleted if there are NO remaining ProtocolControllerForUnit associations, to other ProtocolControllers.

EXPERIMENTAL

18.6 Client Considerations and Recipes

18.6.1 Expose and Hide LUNs

```
// DESCRIPTION:
//
// Test the accuracy of the Masking and Mapping
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. A reference to a storage element, a Storage Volume or Logical Disk
//    is defined in the $StorageElement-> variable
//    This storage element must not already be masked to any initiator
// 2. The WWN of two different Initiator Ports to be masked to is defined in the
//    #InitiatorWWN1 and #InitiatorWWN2 variables.
// 3. The value of
//    CIM_ProtocolControllerMaskingCapabilities.ClientSelectableDeviceNumbers
//    is stored in #ClientSelectableDeviceNumbers
// 4. If #ClientSelectableDeviceNumbers is TRUE, the device number to be used
//    for mapping is defined in #DeviceNumber.
// 5. The value of CIM_ProtocolControllerMaskingCapabilities.PortsPerView is
//    stored in #PortsPerView
// 6. If #PortsPerView != 4 (All ports share the same view), the target port WWN
//    is contained in the #TargetPortWWN variable.
// 7. The ControllerConfigurationService has been found and the object path
//    value is stored in $ControllerConfigService->

// 8. The value of CIM_ProtocolControllerMaskingCapabilities.OneHardwareIDPerView
//    is
//
//    stored in #OneHardwareIDPerView

// Determine if there is a job created by method
// and wait for the job to complete
// Input:
//   #ReturnCode : The return code of the method
//   $ConcreteJob-> :The output parameter that may have a ConcreteJob REF.
// This method will return control if the recipe was not exited because of error
sub void WaitForJob(#ReturnCode, $ConcreteJob->) {
    if (4096 == #ReturnCode) {
        if ($ConcreteJob-> != null) {
            /*Wait until the completion of the job using $ConcreteJob-> as
            a filter Verify that the OperationalStatus contains 2 ("OK"),
            or 17 ("Completed") */
            $JobInstance = GetInstance($ConcreteJob->,
                false, false, false, null)
            if ($JobInstance.JobState != 7) { // 7 - Completed
                <ERROR! Job failed! >
            }
        }
    }
}
```

Masking and Mapping Subprofile

```
    } else {
        <ERROR! Missing Job reference>
    }
}
}

// Step 1. Subscribe for indications on the Job

// Job success -- Status is '17' ("Completed") and '2' ("OK")

#Filter1 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete' and 'OK'">

// Determine if the Indication already exists

// If it doesn't, create it


// Job failure -- Status is '17' ("Completed") and '6' ("Error")

#Filter2 = <the filter from Job Control subprofile with the description
"Modification of Operational Status for a Concrete Job to 'Complete' and
'Error'">

// Determine if the Indication already exists

// If it doesn't, create it


// Step 2. Expose a new LUN to an initiator
$StorageElement = GetInstance($StorageElement->,
    false, false, false, {"Name"})
%InputArguments["LUNames"] = {$StorageElement.Name}
%InputArguments["InitiatorPortIDs"] = {#InitiatorWWN1}

if (#PortsPerView != 4) { // 4 = All ports share the same view
    %InputArguments["TargetPortIDs"] = {#TargetPortWWN}
}

if (#ClientSelectableDeviceNumbers == TRUE) {
    %InputArguments["DeviceNumbers"] = {#DeviceNumber}
    %InputArguments["DeviceAccesses"] = {2} // Read-Write
}
else {
    %InputArguments["DeviceNumbers"] = NULL
    %InputArguments["DeviceAccesses"] = NULL
}
```

```

}

#ReturnCode = InvokeMethod($ControllerConfigService->,
    "ExposePaths",
    %InputArguments,
    %OutputArguments)
// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

$MMJob-> = %OutputArguments["Job"]
if ($MMJob-> == null) {
    $CreatedOrModifiedSPCs->[] = %OutputArguments["ProtocolControllers"]
}
else {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $MMJob->)

    // Now get the SPCs
    $CreatedOrModifiedSPCs->[] = Associators(
        $MMJob->,
        "CIM_AffectedJobElement",
        "CIM_ProtocolController",
        "AffectingElement",
        "AffectedElement",
        false, false, null)
}

// Verify results
if ($CreatedOrModifiedSPCs->[].length == 0) {
    <ERROR! There must be one or more SPC created or modified>
}

#Found = false
for #i in $CreatedOrModifiedSPCs->[] {
    $CheckSPCForUnits[] = References($CreatedOrModifiedSPCs->[#i],
        "CIM_ProtocolControllerForUnit",
        "Antecedent",
        false, false, null)
    for #u in $CheckSPCForUnits[] {
        if (#ClientSelectableDeviceNumbers == TRUE) {
            if ($CheckSPCForUnits[#u].DeviceNumber != #DeviceNumber ||
                $CheckSPCForUnits[#u].DeviceAccess != 2) {
                // no match found try next one (if any)
                continue
            }
        }
    }
}

```

```

    }

    // Validate Initiator ID
    $CheckAuthTargets->[] =
        AssociatorNames($CheckSPCForUnits[#u].Antecedent,
            "CIM_AuthorizedTarget",
            "CIM_AuthorizedPrivilege",
            null, null)

    for #k in $CheckAuthTargets->[] {
        $StorageHWIDs[] = Associators($CheckAuthTargets->[#k],
            "CIM_AuthorizedSubject",
            "CIM_StorageHardwareID",
            null, null, false, false, null)
        for #j in $StorageHWIDs[] {
            if ($StorageHWIDs[#j].StorageID == #InitiatorWWN1) {
                #Found = true
                break
            }
        }
        if (#Found == true) {
            break
        }
    }
    // Validate StorageElement
    if (#Found == true) { // If we didn't find initiator then don't bother
        $CheckStorageElement = GetInstance($CheckSPCForUnits[#u].Dependent,
            false, false, false, null)
        if ($StorageElement.Name != $CheckStorageElement.Name) {
            <ERROR! Masked and Mapped Storage Element not found>
        }
    }
}

if (#Found == false) {
    <ERROR! Created mapping and masking was not found>
}

// Note: since we created one SPC, there should only be one entry here
$AllCreatedOrModifiedSPCs->[] = $CreatedOrModifiedSPCs->[]

// Step 3. Expose a currently exposed LUN to a different initiator

if (#OneHardwareIDPerView == FALSE) {
    %InputArguments["LUNames"] = NULL
    %InputArguments["InitiatorPortIDs"] = {#InitiatorWWN2}
    %InputArguments["TargetPortIDs"] = NULL
    %InputArguments["DeviceAccesses"] = NULL
    // Note: ExposePaths on a modify operation takes an array containing

```



```

// one and only one SPC, which is what we have here
%InputArguments["ProtocolControllers"] = { $CreatedOrModifiedSPCs->[0]}

#ReturnCode = InvokeMethod($ControllerConfigService->,
                           "ExposePaths",
                           %InputArguments,
                           %OutputArguments)
// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

$MMJob-> = %OutputArguments["Job"]
if ($MMJob-> == null) {
    $CreatedOrModifiedSPCs->[] = %OutputArguments["ProtocolControllers"]
}
else {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $MMJob->)

    // Now get the SPCs
    $CreatedOrModifiedSPCs->[] = Associators($MMJob->,
        "CIM_AffectedJobElement",
        "CIM_ProtocolController",
        "AffectingElement",
        "AffectedElement",
        false, false, null)
}

// Verify results
if ($CreatedOrModifiedSPCs->[].length == 0) {
    <ERROR! There must be one or more SPC created or modified>
}

#Found = false
for #i in $CreatedOrModifiedSPCs->[] {
    $CheckSPCForUnits[] = References($CreatedOrModifiedSPCs->[#i],
        "CIM_ProtocolControllerForUnit",
        "Antecedent",
        false, false, null)
    for #u in $CheckSPCForUnits[] {
        // Validate Initiator ID
        $CheckAuthTargets->[] =
            AssociatorNames($CheckSPCForUnits[#u].Antecedent,
                "CIM_AuthorizedTarget",

```

Masking and Mapping Subprofile

```

        "CIM_AuthorizedPrivilege",
        null, null)
    for #k in $CheckAuthTargets->[] {
        $StorageHWIDs[] = Associators($CheckAuthTargets->[#k],
            "CIM_AuthorizedSubject",
            "CIM_StorageHardwareID",
            null, null, false, false, null)
        for #j in $StorageHWIDs[] {
            if ($StorageHWIDs[#j].StorageID == #InitiatorWWN2) {
                #Found = true
                break
            }
        }
        if (#Found == true) {
            break
        }
    }
    // Validate StorageElement
    if (#Found == true) { // If we didn't find initiator then don't bother
        $CheckStorageElement =
            GetInstance($CheckSPCForUnits[#u].Dependent,
                false, false, false, null)
        if ($StorageElement.Name != $CheckStorageElement.Name) {
            <ERROR! Masked and Mapped Storage Element not found>
        }
    }
}

if (#Found == false) {
    <ERROR! Created mapping and masking was not found>
}

$AllCreatedOrModifiedSPCs->[] = $AllCreatedOrModifiedSPCs->[] +
    $CreatedOrModifiedSPCs->[]
/* Current contents of $AllCreatedOrModifiedSPCs->[] array
   plus any new, unique SPC REFs */
} // if #OneHardwareIDPerView == FALSE

// Step 4. Hide the paths previously exposed
// Since we can only pass in one SPC to HidePaths, we need to loop

// through the SPCs and call HidePaths for each one

$ModifiedSPCs->[] = null

```

```

for #spc in $AllCreatedOrModifiedSPCs->[] {
    $StorageElement = GetInstance($StorageElement->,
                                false, false, false, {"Name"})
    %InputArguments2["LUNames"] = {$StorageElement.Name}

    if (#OneHardwareIDPerView == FALSE) {
        %InputArguments2["InitiatorPortIDs"] = {#InitiatorWWN1,#InitiatorWWN2}
    }

    else {

        %InputArguments2["InitiatorPortIDs"] = {#InitiatorWWN1}

    }

    if (#PortsPerView != 4) { // All ports share the same view
        %InputArguments["TargetPortIDs"] = {#TargetPortWWN}
    }

    %InputArguments2["ProtocolControllers"] = {$AllCreatedOrModifiedSPCs->[#spc]}

    #ReturnCode = InvokeMethod($ControllerConfigService->,

                                "HidePaths",

                                %InputArguments2, %OutputArguments2)
    // 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
    if(#ReturnCode != 0 || #ReturnCode != 4096) {
        <ERROR! Method failure>
    }

    // Save any SPCs returned for later validation

    $MMJob-> = %OutputArguments["Job"]
    if ($MMJob == null) {

        $ModifiedSPCs->[] = %OutputArguments["ProtocolControllers"]

    }
    else {
        // Wait until job is finished

```

Masking and Mapping Subprofile

```
&WaitForJob(#ReturnCode, $MMJob->)

// Now get the SPCs

$CreatedOrModifiedSPCs->[] = Associators(

    $MMJob->,

    "CIM_AffectedJobElement",

    "CIM_ProtocolController",

    "AffectingElement",

    "AffectedElement",

    false,

    false,

    null)

$ModifiedSPCs->[] = $ModifiedSPCs->[] + $CreatedOrModifiedSPCs->[]

/* Current contents of $ModifiedSPCs->[] array
   plus any new, unique SPC REFs from $CreatedOrModifiedSPCs->[]

   this list may be null */
}

}

// Verify results
#Found = false

// See if the storage element is still associated to one of the SPCs
$CheckSPCs->[] = AssociatorNames($StorageElement->,
    "CIM_ProtocolControllerForUnit",
    "CIM_ProtocolController",

    // Assumes StorageElement LogicalDevice
    null, null)

for #x in $CheckSPCs->[] {
    for #i in $ModifiedSPCs->[] {
```

```

        if($CheckSPCs->[#x].DeviceID == $ModifiedSPCs->[#i].DeviceID) {
            #Found = true

            break
        }
    }

    if (#Found == true) {

        <ERROR! Element still mapped>
    }
}

// See if the Initiator WWNs are still associated to one of the SPCs

for #i in $ModifiedSPCs->[] {
    $CheckAuthPrivilege->[] = AssociatorNames($ModifiedSPCs->[#i],
        "CIM_AuthorizedTarget",
        "CIM_AuthorizedPrivilege",
        null, null)

    for #k in $CheckAuthPrivilege->[] {
        $StorageHWIDs[] = Associators($CheckAuthPrivilege->[#k],
            "CIM_AuthorizedSubject",
            "CIM_StorageHardwareID",
            null, null, false, false, { "StorageID" })

        for #j in $StorageHWIDs[] {
            if($StorageHWIDs[#j].StorageID == #InitiatorWWN1 ||

                $StorageHWIDs[#j].StorageID == #InitiatorWWN2 ) {
                #Found = true
                break
            }
        }
    }

    if(#Found == true) {

        break // CheckAuthTargets loop
    }
}

```

```

    }
    if(#Found == true) {

        <ERROR! Element still masked>
    }
}

```

EXPERIMENTAL

18.6.2 Set Host Mode for a Port

```

// DESCRIPTION:
//
// Associate a ElementSettingData to a Port
// In this use case, the client wishes to set the FCPort to a specific
// OS-type.
// 1. Find a StorageClientSettingData instance to uses by enumerating
//    all instances of StorageClientSettingData scoped to that
//    ComputerSystem
// 2. Identify the Port to use
// 3. Find the existing ElementSettingData association (if any),
// 4. Delete it via DeleteInstance
// 5. Create a new one from the Port to the
//    StorageClientSettingData via CreateInstance

// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The StorageClientSettingData instance to use has been identified
//    and its reference stored in $ClientSettingData->
// 2. A Port has been identified and the reference stored in
//    $Port->

// Step 1. Delete any existing ElementSettingData association

$ExistingAssocs[] = Associators(
    $Port->,
    "CIM_ElementSettingData",
    "CIM_StorageClientSettingData",
    "ManagedElement",
    "SettingData",
    false, false, null)
for #i in $ExistingAssocs[] {
    $ObjectPath-> = $ExistingAssocs[#i].getObjectPath()
    #Result = DeleteInstance($ObjectPath->)
}

// Step 2. Associate the Port to the new setting

```

```

$instance = newInstance("CIM_ElementSettingData")
$instance.ManagedElement = $Port->
$instance.SettingData     = $ClientSettingData->

$CreatedInst-> = CreateInstance($instance)

```

18.6.3 Set Host Mode for a ProtocolController

```

// DESCRIPTION:
//
// Associate a ElementSettingData to a ProtocolController
// In this use case, the client wishes to set the ProtocolController
// to a specific OS-type.
// 1. Find a StorageClientSettingData instance to uses by enumerating
//    all instances of StorageClientSettingData scoped to that
//    ComputerSystem
// 2. Identify the ProtocolController to use
// 3. Find the existing ElementSettingData association (if any),
// 4. Delete it via DeleteInstance
// 5. Create a new one from the ProtocolController to the
//    StorageClientSettingData via CreateInstance

// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The StorageClientSettingData instance to use has been identified
//    and its reference stored in $ClientSettingData->
// 2. A ProtocolController has been identified and the reference stored in
//    $SPC->

// Step 1. Delete any existing ElementSettingData association

$ExistingAssocs[] = Associators(
    $SPC->,
    "CIM_ElementSettingData",
    "CIM_StorageClientSettingData",
    "ManagedElement",
    "SettingData",
    false, false, null)
for #i in $ExistingAssocs[] {
    $ObjectPath-> = $ExistingAssocs[#i].getObjectPath()
    #Result = DeleteInstance($ObjectPath->)
}

// Step 2. Associate the ProtocolController to the new setting

$instance = newInstance("CIM_ElementSettingData")
$instance.ManagedElement = $SPC->

```

```
$instance.SettingData = $ClientSettingData->
```

```
$CreatedInst-> = CreateInstance($instance)
```

EXPERIMENTAL

18.7 Registered Name and Version

Masking and Mapping version 1.2.0

18.8 CIM Elements

Table 240: CIM Elements for Masking and Mapping

Element Name	Requirement	Description
CIM_AuthorizedPrivilege (18.8.1)	Mandatory	
CIM_AuthorizedSubject (18.8.2)	Mandatory	
CIM_AuthorizedTarget (18.8.3)	Mandatory	
CIM_ConcreteDependency (Associates ControllerConfigurationService and ProtocolController) (18.8.4)	Mandatory	
CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and StorageHardwareID) (18.8.5)	Mandatory	
CIM_ConcreteDependency (Associates PrivilegeManagementService and AuthorizedPrivilege) (18.8.6)	Mandatory	
CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and SystemSpecificCollection) (18.8.7)	Conditional	Conditional requirement: Implementation support for collections of StorageHardwareIDs.
CIM_ControllerConfigurationService (18.8.8)	Mandatory	
CIM_ElementCapabilities (18.8.9)	Mandatory	
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareIDManagementService) (18.8.10)	Optional	Associates EnabledLogicalElementCapabilities with StorageHardwareIDManagementService.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ControllerConfigurationService) (18.8.11)	Optional	Associates EnabledLogicalElementCapabilities with ControllerConfigurationService.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareID) (18.8.12)	Optional	Associates EnabledLogicalElementCapabilities to StorageHardwareID
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to SystemSpecificCollection) (18.8.13)	Conditional	Conditional requirement: Implementation support for collections of StorageHardwareIDs. Associates EnabledLogicalElementCapabilities and SystemSpecificCollection.
CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ProtocolController) (18.8.14)	Optional	Expressed the ability for the element to be named or have its state changed.
CIM_ElementSettingData (Associates ComputerSystem and StorageClientSettingData) (18.8.15)	Mandatory	

Table 240: CIM Elements for Masking and Mapping

Element Name	Requirement	Description
CIM_ElementSettingData (Associates StorageHardwareID and StorageClientSettingData) (18.8.16)	Optional	
CIM_ElementSettingData (Associates ProtocolController and StorageClientSettingData) (18.8.17)	Optional	
CIM_ElementSettingData (Associates Port and StorageClientSettingData) (18.8.18)	Optional	
CIM_EnabledLogicalElementCapabilities (18.8.19)	Optional	This class is used to express the naming and possible requested state change possibilities for storage elements.
CIM_HostedCollection (18.8.20)	Conditional	Conditional requirement: Implementation support for collections of StorageHardwareIDs.
CIM_HostedService (Associates ComputerSystem and ControllerConfigurationService) (18.8.21)	Mandatory	
CIM_HostedService (Associates ComputerSystem and PrivilegeManagementService) (18.8.22)	Mandatory	
CIM_HostedService (Associates ComputerSystem and StorageHardwareIDManagementService) (18.8.23)	Mandatory	
CIM_MemberOfCollection (18.8.24)	Conditional	Conditional requirement: Implementation support for collections of StorageHardwareIDs.
CIM_PrivilegeManagementService (18.8.25)	Mandatory	
CIM_ProtocolController (18.8.26)	Mandatory	
CIM_ProtocolControllerForUnit (18.8.27)	Mandatory	
CIM_ProtocolControllerMaskingCapabilities (18.8.28)	Mandatory	
SNIA_ProtocolControllerMaskingCapabilities (18.8.29)	Optional	An experimental subclass of CIM_ProtocolControllerMaskingCapabilities.
CIM_SAPAvailableForElement (18.8.30)	Mandatory	
CIM_StorageClientSettingData (18.8.31)	Mandatory	
CIM_StorageHardwareID (18.8.32)	Mandatory	
CIM_StorageHardwareIDManagementService (18.8.33)	Mandatory	

Table 240: CIM Elements for Masking and Mapping

Element Name	Requirement	Description
CIM_SystemSpecificCollection (18.8.34)	Conditional	Conditional requirement: Implementation support for collections of StorageHardwareIDs.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ProtocolController	Mandatory	Creation of a ProtocolController
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ProtocolController	Mandatory	Deletion of a ProtocolController
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Mandatory	Creation of a ProtocolControllerForUnit association
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Mandatory	Deletion of a ProtocolControllerForUnit association
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Mandatory	Modification of a ProtocolControllerForUnit association (e.g. changing DeviceNumber)
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_AuthorizedSubject	Mandatory	Creation of an AuthorizedSubject association
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_AuthorizedSubject	Mandatory	Deletion of an AuthorizedSubject association

18.8.1 CIM_AuthorizedPrivilege

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Class Mandatory: Mandatory

Table 241 describes class CIM_AuthorizedPrivilege.

Table 241: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque and unique identifier
ElementName		Optional	User friendly name
PrivilegeGranted		Mandatory	Indicates if the privilege is granted or not

Table 241: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Properties	Flags	Requirement	Description & Notes
Activities		Mandatory	For SMI-S, shall be 5,6 ('Read' and Write').

18.8.2 CIM_AuthorizedSubject

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Class Mandatory: Mandatory

Table 242 describes class CIM_AuthorizedSubject.

Table 242: SMI Referenced Properties/Methods for CIM_AuthorizedSubject

Properties	Flags	Requirement	Description & Notes
PrivilegedElement		Mandatory	The Subject for which Privileges are granted or denied
Privilege		Mandatory	The Privilege either granted or denied to an Identity or group of Identities collected by a Role.

18.8.3 CIM_AuthorizedTarget

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Class Mandatory: Mandatory

Table 243 describes class CIM_AuthorizedTarget.

Table 243: SMI Referenced Properties/Methods for CIM_AuthorizedTarget

Properties	Flags	Requirement	Description & Notes
TargetElement		Mandatory	The target set of resources to which the Privilege applies
Privilege		Mandatory	The Privilege affecting the target resource

18.8.4 CIM_ConcreteDependency (Associates ControllerConfigurationService and ProtocolController)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 244 describes class CIM_ConcreteDependency (Associates ControllerConfigurationService and ProtocolController).

Table 244: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates ControllerConfigurationService and ProtocolController)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

18.8.5 CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and StorageHardwareID)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 245 describes class CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and StorageHardwareID).

Table 245: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and StorageHardwareID)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.6 CIM_ConcreteDependency (Associates PrivilegeManagementService and AuthorizedPrivilege)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 246 describes class CIM_ConcreteDependency (Associates PrivilegeManagementService and AuthorizedPrivilege).

Table 246: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates PrivilegeManagementService and AuthorizedPrivilege)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.7 CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and SystemSpecificCollection)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ProtocolControllerSupportsCollections

Table 247 describes class CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and SystemSpecificCollection).

Table 247: SMI Referenced Properties/Methods for CIM_ConcreteDependency (Associates StorageHardwareIDManagementService and SystemSpecificCollection)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.8 CIM_ControllerConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 248 describes class CIM_ControllerConfigurationService.

Table 248: SMI Referenced Properties/Methods for CIM_ControllerConfigurationService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The scoping System CreationClassName
SystemName		Mandatory	The scoping System Name
CreationClassName		Mandatory	The name of the concrete subclass
Name		Mandatory	Unique identifier for the Service
ExposePaths()		Mandatory	
HidePaths()		Mandatory	
ExposeDefaultLUs()		Optional	
HideDefaultLUs()		Optional	
DeleteProtocolController()		Optional	

18.8.9 CIM_ElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 249 describes class CIM_ElementCapabilities.

Table 249: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
Capabilities		Mandatory	

18.8.10 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareIDManagementService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 250 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareIDManagementService).

Table 250: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareIDManagementService)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

18.8.11 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ControllerConfigurationService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 251 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ControllerConfigurationService).

Table 251: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ControllerConfigurationService)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	The managed element

18.8.12 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareID)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 252 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareID).

Table 252: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to StorageHardwareID)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	

18.8.13 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to SystemSpecificCollection)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: ProtocolControllerSupportsCollections

Table 253 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to SystemSpecificCollection).

Table 253: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to SystemSpecificCollection)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	

18.8.14 CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ProtocolController)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 254 describes class CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ProtocolController).

Table 254: SMI Referenced Properties/Methods for CIM_ElementCapabilities (EnabledLogicalElementCapabilities to ProtocolController)

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	The capabilities object associated with the element.
ManagedElement		Mandatory	

18.8.15 CIM_ElementSettingData (Associates ComputerSystem and StorageClientSettingData)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 255 describes class CIM_ElementSettingData (Associates ComputerSystem and StorageClientSettingData).

Table 255: SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates ComputerSystem and StorageClientSettingData)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
SettingData		Mandatory	

18.8.16 CIM_ElementSettingData (Associates StorageHardwareID and StorageClientSettingData)

Created By: Extrinsic: CIM_StorageHardwareIDManagementService.CreateStorageHardwareID

Modified By: Static

Deleted By: Extrinsic: CIM_StorageHardwareIDManagementService.DeleteStorageHardwareID

Class Mandatory: Optional

Table 256 describes class CIM_ElementSettingData (Associates StorageHardwareID and StorageClientSettingData).

Table 256: SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates StorageHardwareID and StorageClientSettingData)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
SettingData		Mandatory	

18.8.17 CIM_ElementSettingData (Associates ProtocolController and StorageClientSettingData)

Created By: CreateInstance

Modified By: Static

Deleted By: DeleteInstance

Class Mandatory: Optional

Table 257 describes class CIM_ElementSettingData (Associates ProtocolController and StorageClientSettingData).

Table 257: SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates ProtocolController and StorageClientSettingData)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
SettingData		Mandatory	

18.8.18 CIM_ElementSettingData (Associates Port and StorageClientSettingData)

Created By: CreateInstance

Modified By: Static

Deleted By: DeleteInstance

Class Mandatory: Optional

Table 258 describes class CIM_ElementSettingData (Associates Port and StorageClientSettingData).

Table 258: SMI Referenced Properties/Methods for CIM_ElementSettingData (Associates Port and StorageClientSettingData)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
SettingData		Mandatory	

18.8.19 CIM_EnabledLogicalElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 259 describes class CIM_EnabledLogicalElementCapabilities.

Table 259: SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	The moniker for the instance.

Table 259: SMI Referenced Properties/Methods for CIM_EnabledLogicalElementCapabilities

Properties	Flags	Requirement	Description & Notes
ElementNameEditSupported		Mandatory	Denotes whether an storage element can be named
MaxElementNameLength		Mandatory	Specifies the maximum length in glyphs (letters) for the name. See MOF for details.
ElementNameMask		Mandatory	The regular expression that specifies the possible content and format for the element name. See MOF for details
RequestedStatesSupported		Optional	Expresses the states to which this element may be changed using the RequestStateChange method. If this property, it may be assumed that the state may not be changed.

18.8.20 CIM_HostedCollection

Created By: Extrinsic: CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection

Modified By: Static

Deleted By: Static

Class Mandatory: ProtocolControllerSupportsCollections

Table 260 describes class CIM_HostedCollection.

Table 260: SMI Referenced Properties/Methods for CIM_HostedCollection

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.21 CIM_HostedService (Associates ComputerSystem and ControllerConfigurationService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 261 describes class CIM_HostedService (Associates ComputerSystem and ControllerConfigurationService).

Table 261: SMI Referenced Properties/Methods for CIM_HostedService (Associates Computer-System and ControllerConfigurationService)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.22 CIM_HostedService (Associates ComputerSystem and PrivilegeManagementService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 262 describes class CIM_HostedService (Associates ComputerSystem and PrivilegeManagementService).

Table 262: SMI Referenced Properties/Methods for CIM_HostedService (Associates Computer-System and PrivilegeManagementService)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.23 CIM_HostedService (Associates ComputerSystem and StorageHardwareIDManagementService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 263 describes class CIM_HostedService (Associates ComputerSystem and StorageHardwareIDManagementService).

Table 263: SMI Referenced Properties/Methods for CIM_HostedService (Associates Computer-System and StorageHardwareIDManagementService)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

18.8.24 CIM_MemberOfCollection

Created By: Extrinsic: CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection,
CIM_StorageHardwareIDManagementService.AddHardwareIDsToCollection

Modified By: Static

Deleted By: Static

Class Mandatory: ProtocolControllerSupportsCollections

Table 264 describes class CIM_MemberOfCollection.

Table 264: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	
Member		Mandatory	

18.8.25 CIM_PrivilegeManagementService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 265 describes class CIM_PrivilegeManagementService.

Table 265: SMI Referenced Properties/Methods for CIM_PrivilegeManagementService

Properties	Flags	Requirement	Description & Notes
SystemCreationClass sName		Mandatory	The scoping System CreationClassName
CreationClassName		Mandatory	The name of the concrete subclass
SystemName		Mandatory	The scoping System Name
Name		Mandatory	Uniquely identifies the Service
ElementName		Mandatory	User friendly name
AssignAccess()		Mandatory	
RemoveAccess()		Mandatory	

18.8.26 CIM_ProtocolController

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths

Class Mandatory: Mandatory

Table 266 describes class CIM_ProtocolController.

Table 266: SMI Referenced Properties/Methods for CIM_ProtocolController

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The scoping System CreationClassName
CreationClassName		Mandatory	The name of the concrete subclass
SystemName		Mandatory	The scoping System's Name
DeviceID		Mandatory	Unique name for the ProtocolController

18.8.27 CIM_ProtocolControllerForUnit

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs,
CIM_ControllerConfigurationService.HideDefaultLUs

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs,
CIM_ControllerConfigurationService.HideDefaultLUs

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs,
CIM_ControllerConfigurationService.HideDefaultLUs

Class Mandatory: Mandatory

Table 267 describes class CIM_ProtocolControllerForUnit.

Table 267: SMI Referenced Properties/Methods for CIM_ProtocolControllerForUnit

Properties	Flags	Requirement	Description & Notes
DeviceNumber		Mandatory	Address (e.g. LUN) of the associated Device. Shall be formatted as unseparated uppercase hexadecimal digits, with no leading 0x.
DeviceAccess		Mandatory	The access rights granted to the referenced logical unit as exposed through referenced ProtocolController

Table 267: SMI Referenced Properties/Methods for CIM_ProtocolControllerForUnit

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	A reference to the SCSI logical unit (for example, a Block Services StorageVolume)

18.8.28 CIM_ProtocolControllerMaskingCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 268 describes class CIM_ProtocolControllerMaskingCapabilities.

Table 268: SMI Referenced Properties/Methods for CIM_ProtocolControllerMaskingCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque and unique identifier
ElementName		Mandatory	User-friendly name
ValidHardwareIDTypes		Mandatory	A list of the valid values for StorageHardwareID.IDType
PortsPerView		Mandatory	Indicates the way that ports per view (ProtocolController) are handled
ClientSelectableDeviceNumbers		Mandatory	Indicates whether the client can specify the DeviceNumbers parameter when calling ControllerConfigurationService.ExposePaths().
OneHardwareIDPerView		Mandatory	Set to true if this storage system limits configurations to a single subject hardware ID per view.
PrivilegeDeniedSupported		Mandatory	Set to true if this storage system allows a client to create a Privilege instance with PrivilegeGranted set to FALSE.
UniqueUnitNumbersPerPort		Mandatory	Indicates whether different ProtocolControllers attached to a SCSIProtocolEndpoint can expose the same unit numbers (e.g. multiple LUN 0s) or if the numbers must be unique
ProtocolControllerSupportsCollections		Optional	Indicates the storage system supports SystemSpecificCollections of StorageHardwareIDs
OtherValidHardwareIDTypes		Conditional	Conditional requirement: Properties required when ValidHardwareIDTypes includes 1 (Other).. An array of strings describing types for valid StorageHardwareID.IDType. Used when the ValidHardwareIDTypes includes Other

Table 268: SMI Referenced Properties/Methods for CIM_ProtocolControllerMaskingCapabilities

Properties	Flags	Requirement	Description & Notes
MaximumMapCount		Mandatory	The maximum number of ProtocolControllerForUnit associations that can be associated with a single LogicalDevice (for example, StorageVolume). Zero indicates there is no limit
SPCAllowsNoLUs		Mandatory	Set to true if a client can create an SPC with no LogicalDevices
SPCAllowsNoTargets		Mandatory	Set to true if a client can create an SPC with no target SCSIProtocolEndpoints
SPCAllowsNoInitiators		Mandatory	Set to true if a client can create an SPC with no StorageHardwareIDs
SPCSupportsDefault Views		Mandatory	Set to true if it the instrumentation supports default view SPCs that exposes logical units to all initiators

18.8.29 SNIA_ProtocolControllerMaskingCapabilities

An experimental subclass of CIM_ProtocolControllerMaskingCapabilities that adds two properties. This may be used rather than SNIA_ProtocolControllerMaskingCapabilities. All the properties of CIM_ProtocolControllerMaskingCapabilities apply in addition to the properties described here.

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 269 describes class SNIA_ProtocolControllerMaskingCapabilities.

Table 269: SMI Referenced Properties/Methods for SNIA_ProtocolControllerMaskingCapabilities

Properties	Flags	Requirement	Description & Notes
SupportedAsynchronousActions		Mandatory	Indicates which operations will result in a Job being created
SupportedSynchronousActions		Mandatory	Indicates which operations will execute without a Job being created

18.8.30 CIM_SAPAvailableForElement

Created By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths, CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs, CIM_ControllerConfigurationService.HideDefaultLUs

Modified By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths, CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs, CIM_ControllerConfigurationService.HideDefaultLUs

Deleted By: Extrinsic: CIM_ControllerConfigurationService.ExposePaths,
 CIM_ControllerConfigurationService.HidePaths, CIM_ControllerConfigurationService.ExposeDefaultLUs,
 CIM_ControllerConfigurationService.HideDefaultLUs

Class Mandatory: Mandatory

Table 270 describes class CIM_SAPAvailableForElement.

Table 270: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Properties	Flags	Requirement	Description & Notes
AvailableSAP		Mandatory	
ManagedElement		Mandatory	

18.8.31 CIM_StorageClientSettingData

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 271 describes class CIM_StorageClientSettingData.

Table 271: SMI Referenced Properties/Methods for CIM_StorageClientSettingData

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque and unique identifier
ElementName		Mandatory	A user-friendly name
ClientTypes		Mandatory	Array of OS names

18.8.32 CIM_StorageHardwareID

Created By: Extrinsic: CIM_StorageHardwareIDManagementService.CreateStorageHardwareID,
 CIM_ControllerConfigurationService.ExposePaths

Modified By: Static

Deleted By: Extrinsic: CIM_StorageHardwareIDManagementService.DeleteStorageHardwareID

Class Mandatory: Mandatory

Table 272 describes class CIM_StorageHardwareID.

Table 272: SMI Referenced Properties/Methods for CIM_StorageHardwareID

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque and unique identifier
StorageID	N	Mandatory	The worldwide unique ID
IDType		Mandatory	StorageID type. Values may be 1 2 3 4 5 (Other or PortWWN or NodeWWN or Hostname or iSCSI Name).

18.8.33 CIM_StorageHardwareIDManagementService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 273 describes class CIM_StorageHardwareIDManagementService.

Table 273: SMI Referenced Properties/Methods for CIM_StorageHardwareIDManagementService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The scoping System CreationClassName
SystemName		Mandatory	The scoping System Name
CreationClassName		Mandatory	The name of the concrete subclass
Name		Mandatory	Uniquely identifies the Service
CreateStorageHardwareID()		Mandatory	
DeleteStorageHardwareID()		Mandatory	
CreateHardwareIDCollection()		Optional	
AddHardwareIDsToCollection()		Optional	

18.8.34 CIM_SystemSpecificCollection

Created By: Extrinsic: CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection

Modified By: Static

Deleted By: Static

Class Mandatory: ProtocolControllerSupportsCollections

Table 274 describes class CIM_SystemSpecificCollection.

Table 274: SMI Referenced Properties/Methods for CIM_SystemSpecificCollection

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque and unique identifier
ElementName		Mandatory	A user-friendly name

STABLE

EXPERIMENTAL

Clause 19: Pool Management Policy Subprofile

19.1 Description

The Pool Management Policy subprofile would be deployed by any Array Profile implementation that provides Policy management capability. The Profile that supports the Pool Management Policy subprofile may be referred to as a “**Policy based**” implementation. The Pool Management Policy implementation may support the Rules, Conditions and Actions via static instances of the same. For more information refer to the Policy Subprofile.

The idea of the Pool Management Policy Subprofile is to support the automated filling of a pool that is nearing exhaustion. This eases the administrative burden of pool management and provides pools space when more volumes are required. This policy could also be combined with other policies that create or expand volumes based on filesystem free space, add extents to the primordial pool, etc.

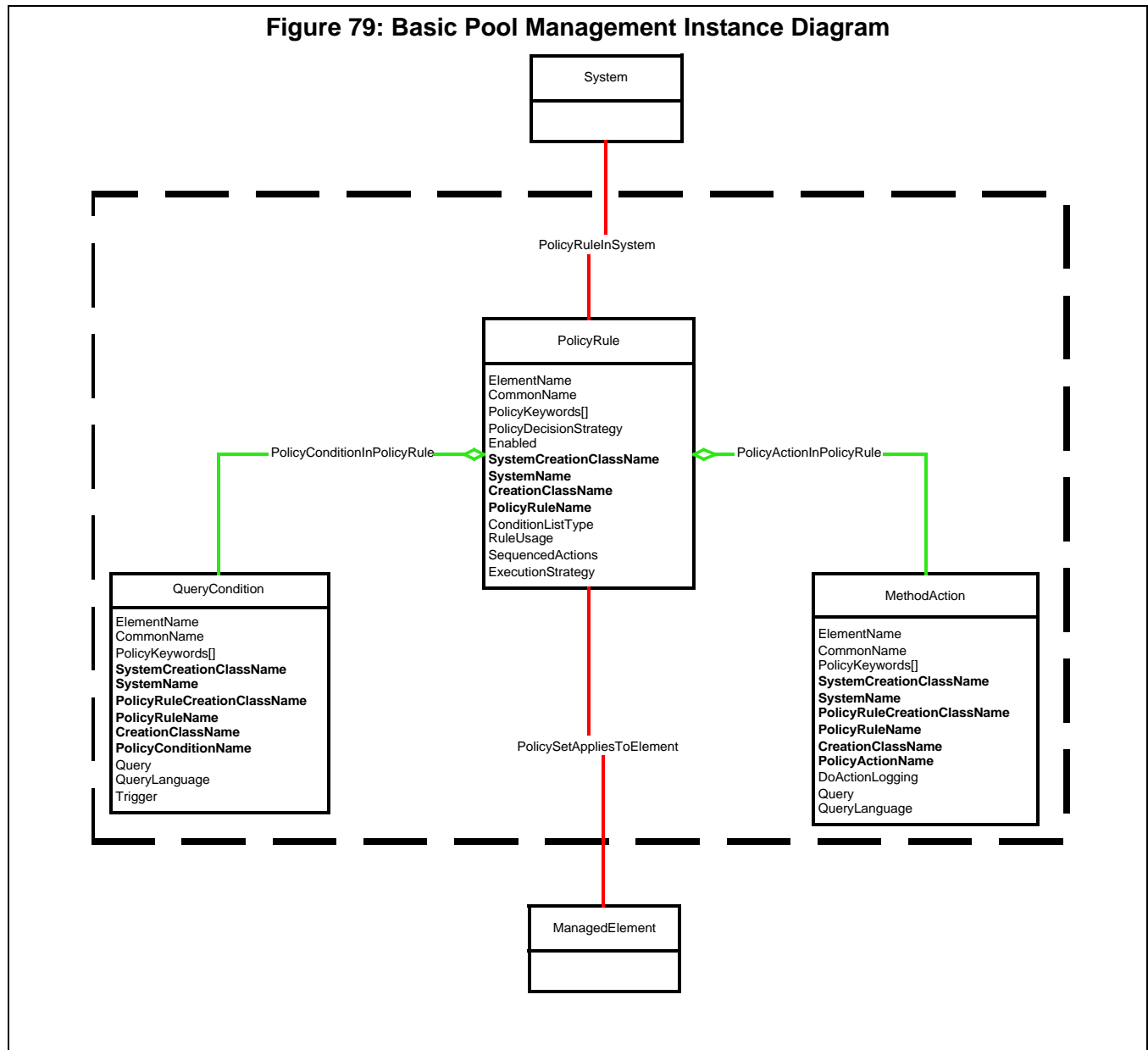
The basic idea is to draw space from the primordial pool into a concrete pool whenever the concrete pool is in danger of being exhausted. Normally, the volume allocation would fail, then the administrator would need to add more space to the concrete pool via a manual set of operations. This policy eliminates that step and simplifies administration as a result.

19.1.1 Instance Diagrams

Support for the Pool Management Policy subprofile entails support for a single QueryCondition, single MethodAction and a single PolicyRule.

A client (Policy Client) would be able to enumerate these single instances if they are static, and create them if dynamic support is provided. Additionally, the client will create an instance of PolicySetAppliesToElement for each pool that this policy applies to. The Policy then takes over and makes sure that the designated pools always have sufficient free space.

The basic constructs used by the Pool Management Policy Subprofile are illustrated in Figure 79



The Managed Element in this case is the concrete **StoragePool** that is desired to be managed.

19.1.2 Query Condition

The QueryCondition Query string that defines the conditions under which to add space to the concrete pool is shown below:

```

/*
Pool Exhausted Policy Condition
Preexisting conditions
- 25% of the size of the found concrete Pool remains
  in associated primordial Pool
- The policy set exists and is enabled

```

```

- There is at least one concrete Pool associated to the PolicySet
*/
// Continuously evaluated QueryCondition that 'triggers' policy
select
    OBJECTPATH(primordial) AS POBJ,
    OBJECTPATH(concrete) AS COBJ,
    OBJECTPATH(service) AS SOBJ,
    concrete.TotalManagedSpace * .25 AS AmountToIncrease
from
    CIM_PolicyAppliesToElement applies,
    CIM_StoragePool concrete,
    CIM_StoragePool primordial.
    CIM_AllocatedFromStoragePool alloc,
    CIM_PolicySet policy,
    CIM_HostedService hosted,
    CIM_HostedStoragePool hostedpool,
    CIM_ComputerSystem, system,
    CIM_StorageConfigurationService service
where (concrete.RemainingManagedSpace/p.TotalManagedSpace * 100) < 75
    and concrete.Primordial = false
// Join Primordial Pool with Concrete Pools
    and OBJECTPATH(primordial) = alloc.Antecedent
    and OBJECTPATH(concrete) = alloc.Dependent
// Determine what concrete Pools the PolicySet applies to
    and policy.CommonName = "Pool Exhausting Policy Condition"
    and OBJECTPATH(policy) = element.PolicySet
    and OBJECTPATH(concrete) = element.ManagedElement
// Join found primordial Pool with Service
    and OBJECTPATH(primordial) = hostedpool.PartComponent
    and OBJECTPATH(system) = hostedpool.GroupComponent
    and OBJECTPATH(system) = hosted.Antecedent
    and OBJECTPATH(service) = hosted.Dependent
    and service ISA "CIM_StorageConfigurationService"

```

19.1.3 Method Action

The MethodAction that invokes the StorageConfigurationService to add more space is shown below:

```

// Method Action
select
    SOBJ,      // Service object path
    'CreateOrModifyStoragePool',
    NULL,      // ElementName parameter
    NULL,      // Goal parameter, take default Setting
    AmountToIncrease, // Size parameter
    POBJ,      // InPools parameter
    NULL,      // InExtents parameter
    COBJ       // Pool parameter
from

```

```

CIM_QueryCondition condition,
CIM_QueryResult result,
CIM_PolicySet policy,
CIM_PolicyConditionInPolicyRule inpolicyset
where  policy.CommonName = "Pool Exhausting Policy Condition"
and OBJECTPATH(policy) = inpolicyset.GroupComponent
and OBJECTPATH(condition) = inpolicyset.PartComponent
and CLASSNAME(result) = QueryResult.QueryResultSubclassName

```

19.2 Health and Fault Management Considerations

Not defined in this standard.

19.3 Cascading Considerations

Not defined in this standard.

19.4 Supported Subprofiles and Packages

Table 275 list the subprofiles available to this subprofile.

Table 275: Pool Management Policy Subprofiles

Dependency Name	Type	Mandatory	Notes
Policy Subprofile	subprofile	Yes	

19.5 Methods of the Profile

19.5.1 Extrinsic Methods of the Profile

None.

19.5.2 Intrinsic Methods of the Profile

Table 276 identifies how Policy constructs get created, deleted or modified. Any class not listed is assumed to be pre-existing (e.g., canned) or manipulated through another profile or subprofile.

Table 276: Static Policy Instance Methods

Method	Created Instances	Deleted Instances	Modified Instances
CreateInstance	PolicySetAppliesToElement	N/A	N/A
DeleteInstance	N/A	PolicySetAppliesToElement	N/A
SetProperty	N/A	N/A	PolicyRule (Enabled)

Table 277 identifies how Policy constructs get created, deleted or modified. Any class not listed is assumed to be pre-existing (e.g., canned) or manipulated through another profile or subprofile.

Table 277: Instance Methods of Dynamic Rules and Static Conditions and Actions

Method	Created Instances	Deleted Instances	Modified Instances
CreateInstance	PolicyRule	N/A	N/A
CreateInstance	PolicyConditionInPolicyRule	N/A	N/A
CreateInstance	PolicyActionInPolicyRule	N/A	N/A
DeleteInstance	N/A	PolicyRule	N/A
DeleteInstance	N/A	PolicyConditionInPolicyRule	N/A
DeleteInstance	N/A	PolicyActionInPolicyRule	N/A
SetProperty	N/A	N/A	PolicyRule (Enabled)
SetProperty	N/A	N/A	QueryCondition (Trigger)

Table 278 identifies how Policy constructs get created, deleted or modified. Any class not listed is assumed to be pre-existing (e.g., canned) or manipulated through another profile or subprofile.

Table 278: Dynamic Policy Instance Methods

Method	Created Instances	Deleted Instances	Modified Instances
CreateInstance	PolicyRule	N/A	N/A
CreateInstance	QueryCondition	N/A	N/A
CreateInstance	PolicyConditionInPolicyRule	N/A	N/A
CreateInstance	PolicyConditionIn PolicyCondition	N/A	N/A
CreateInstance	ReusablePolicyComponent	N/A	N/A
	ReusablePolicyContainer	N/A	N/A
CreateInstance	MethodAction	N/A	N/A
CreateInstance	PolicyActionInPolicyRule	N/A	N/A
CreateInstance	PolicyActionInPolicyAction	N/A	N/A
DeleteInstance	N/A	PolicyRule	N/A
DeleteInstance	N/A	QueryCondition	N/A
DeleteInstance	N/A	MethodAction	N/A

Table 278: Dynamic Policy Instance Methods (Continued)

DeleteInstance	N/A	PolicyConditionIn PolicyRule	N/A
DeleteInstance	N/A	PolicyActionInPolicyRule	N/A
DeleteInstance	N/A	PolicyConditionIn PolicyCondition	N/A
DeleteInstance	N/A	ReusablePolicyComponent	N/A
DeleteInstance	N/A	ReusablePolicyContainer	N/A
DeleteInstance	N/A	PolicyActionInPolicyAction	N/A
SetProperty	N/A	N/A	PolicyRule (Enabled)
SetProperty	N/A	N/A	QueryCondition (Trigger)
ModifyInstance	N/A	N/A	QueryCondition
ModifyInstance	N/A	N/A	MethodAction
ModifyInstance	N/A	N/A	PolicyConditionIn PolicyCondition
ModifyInstance	N/A	N/A	PolicyActionIn PolicyAction

19.6 Client Considerations and Recipes

See recipe in the Policy Subprofile for usage of PolicySetAppliesToElement.

19.7 Required CIM Elements

Table 279: Table 8: Pool Management Policy Subprofile Required Classes, Associations, Methods and Indications

Profile Classes &Associations	Notes
MethodAction	Defines a Method to be executed as part of a PolicyRule
MethodActionResult	
PolicyActionInPolicyRule	Associates a MethodAction to the PolicyRules that it is part of.
PolicyCapabilities (EXPERIMENTAL)	Defines the capabilities of the Policy Subprofile.
PolicyConditionInPolicyRule	Associates a QueryCondition to the PolicyRules that it is part of.
PolicyRule	Defines a PolicyRule that is either a Template (with Static Conditions or Actions) or a PolicyRule to be effected.
PolicyRuleInSystem	Associates PolicyRules to the System that hosts them.
PolicySetAppliesToElement	An association that may be referenced in QueryConditions or MethodActions to constrain the application of a PolicyRule. It associates the PolicyRule to ManagedElements.
QueryCapabilities	Defines the Query execution capabilities of the Profile or CIMOM.
QueryCondition	A Query that is used as a condition of a PolicyRule. A QueryCondition where Trigger=TRUE serves as an indication to drive evaluation of other QueryConditions in the PolicyRule.
QueryConditionResult (QueryResult)	
QueryResult	
Profile Indications	Notes

Table 280: Table 9: Instance Creation, Deletion or Modification for Pool Management Policy Sub-profile Classes

Class / Association	Creation	Deletion	Modification
CompoundPolicyAction			
CompoundPolicyCondition			
MethodAction	PROVIDER SUPPLIED or CreateInstance	N/A or DeleteInstance	N/A
MethodActionResult			
PolicyCapabilities (EXPERIMENTAL)	PROVIDER SUPPLIED	N/A	N/A
PolicyCondition (Policy)			
PolicyRule (PolicySet)			
PolicyRuleInSystem (PolicySetInSystem)			
PolicySetAppliesToElement	CreateInstance	DeleteInstance or deletion of either end	N/A
PolicyTrigger			
QueryCapabilities	PROVIDER SUPPLIED	N/A	N/A
QueryCondition (PolicyCondition)			
QueryResult			

19.8 Classes Used in the Profile

The *flags* columns of the tables below employ the values detailed in Table 281:

Table 281: Valid Flag Values

Flag	Long Name	Meaning
R	Required	Property is mandatory
C	Correlatable	An ID (often derived from hardware) that can be correlated between software components
D	Durable	An ID that tends to not change
F	Format	A property that describes the format of another property, the referenced property should be mentioned in Notes

Table 281: Valid Flag Values (Continued)

M	Modifiable	The property can be modified by ModifyInstance
N	Null Okay	Client may set this property to null

19.8.1 MethodAction Class

MethodAction is a PolicyAction that invokes an action defined by a query. The action is defined by a method of an ObjectName, which may be an intrinsic method of a CIM Namespace or an extrinsic method of a CIM_ManagedElement in this case, the Method invoked is. The input parameters to the method are defined by the query and may be fixed values defined by literals or may be defined by reference to one or more properties of QueryConditionResult, MethodActionResult, or other instances.

MethodAction is subclassed from PolicyAction. Its properties are summarized in Table 282.

Created by: A MethodAction can be “Static” (predefined by the provider)

Deleted by: “Static” MethodActions shall not be deleted. However, client defined MethodActions may be deleted using the DeleteInstance intrinsic method.

Modified by: “Static” MethodActions shall not be modified. However, client defined MethodActions may be modified.

ConditionalRequirements: none

Table 282: Properties for MethodAction

Property	Type	Flags	Description / Notes
ElementName	string		Another user-friendly name
CommonName	string		User-friendly name of policy object
PolicyKeywords[]	string		FCAPS strings
SystemCreationClassName	string	R	The name of the class or the subclass used in the creation of the System object in whose scope this PolicyAction is defined.
SystemName	string	R	The name of the System object in whose scope this PolicyAction is defined.
PolicyRuleCreationClassName	string	R	For a rule-specific PolicyAction, the CreationClassName of the PolicyRule object with which this Action is associated. For a reusable PolicyAction, a special value, 'NO RULE', should be used.
PolicyRuleName	string	R	For a rule-specific PolicyAction, the name of the PolicyRule object with which this Action is associated. For a reusable PolicyAction, a special value, 'NO RULE', should be used.
CreationClassName	string	R	The name of the class or the subclass used in the creation of an instance.

Table 282: Properties for MethodAction (Continued)

PolicyActionName	string	R	A user-friendly name of this policy (method) action
DoActionLogging	Boolean		
Query	string	R	The query that defines the method and the input parameters to that method. See 19.1.2 Query Condition for the actual query string text.
QueryLanguage	Uint16	R	This defines the query language being used and shall be set to « 2 » (CQL).

19.8.2 MethodActionResult Class

MethodActionResult is an InstCIMMethodCall class used to communicate the results of one execution of a method invoked by evaluation of the Query in the referenced MethodAction. MethodActionResult instances are dynamically instantiated and have a lifecycle that begins and ends in the context of a single PolicyRule evaluation. The explicit property MethodActionPath identifies the specific MethodAction instance that produced this result.

MethodActionResult is subclassed from InstCIMMethodCall.

Created by: This is created as a side effect of invocation of a MethodAction.

Deleted by: This is implicitly deleted after execution of a PolicyRule.

Modified by: N/A

Conditional Requirements: none

This shall be supported for all Policy Subprofile implementations, except for Policy Subprofiles that only support “type 1” Static Policy Rules.

Table 283: Properties for MethodActionResult

Property	Type	Flags	Description / Notes
IndicationIdentifier	string		An identifier for the Indication. This property is similar to a key value in that it can be used for identification.
CorrelatedIndications[]	string		A list of IndicationIdentifiers whose notifications are correlated with (related to) this one.
IndicationTime	datetime		The time and date of creation of the Method call.
SourceInstance	string	R	A copy of the instance that changed to generate the Indication. SourceInstance contains the current values of the properties selected by the MethodAction
MethodName	string	R	The name of the method invoked.
MethodParameters	string		The parameters of the method, formatted as an EmbeddedObject (with a predefined class name of \"__MethodParameters\").

Table 283: Properties for MethodActionResult (Continued)

ReturnValue	string		When PreCall is FALSE, ReturnValue contains a string representation of the method's return value.
PreCall	Boolean	R	Boolean indicating whether the Indication is sent before the method begins executing (TRUE) or when the method completes (FALSE).
MethodActionPath	string	R	This shall be a fully qualified, WBEM URI, (DSP0207), based Instance Path to the MethodAction instance that produced this result.

19.8.3 PolicyActionInPolicyRule Class

A PolicyRule aggregates zero or more instances of the PolicyAction class, via the PolicyActionInPolicyRule association. A Rule that aggregates zero Actions is not valid--it may, however, be in the process of being entered into a PolicyRepository or being defined for a System. Alternately, the actions of the policy may be explicit in the definition of the PolicyRule. Note that a PolicyRule should have no effect until it is valid.

The Actions associated with a PolicyRule may be given a required order, a recommended order, or no order at all. For Actions represented as separate objects, the PolicyActionInPolicyRule aggregation can be used to express an order.

This aggregation does not indicate whether a specified action order is required, recommended, or of no significance; the property SequencedActions in the aggregating instance of PolicyRule provides this indication.

PolicyActionInPolicyRule is subclassed from PolicyActionStructure. Its properties are summarized in Table 284.

Created by: PolicyAction can be "Static" (predefined by the provider).

Deleted by: "Static" PolicyActions shall not be deleted.

Modified by: "Static" PolicyActions shall not be modified.

Conditional Requirements: none

Table 284: Properties for PolicyActionInPolicyRule

Property	Type	Flags	Description / Notes
ActionOrder	Uint16		ActionOrder is an unsigned integer 'n' that indicates the relative position of a PolicyAction in the sequence of actions associated with a PolicyRule or CompoundPolicyAction.
GroupComponent	REF	R	This property represents the PolicyRule that contains one or more PolicyActions.
PartComponent	REF	R	This property holds the name of a PolicyAction contained by one or more PolicyRules.

19.8.4 PolicyCapabilities Class

PolicyCapabilities specifies the various aspects of the Policy implementation of the Subprofile. PolicyCapabilities are scoped by the Profile or subprofile. If the PolicyCapabilities are scoped to the Profile, then the capabilities would apply to all subprofiles. If the PolicyCapabilities are scoped to a subprofile, then the capabilities would apply to the Subprofile (and override capabilities specified at the Profile level).

PolicyCapabilities are subclassed from Capabilities. Its properties are summarized in Table 285.

Created by: This is implicitly created by the provider.

Deleted by: They are never deleted.

Modified by: They are never modified.

Conditional Requirements: none

For the Pool Management Policy Subprofile, there shall be at least one PolicyCapabilities for at least one Profile or Subprofile that supports the Policy Subprofile. That is, if the Profile and no subprofile is covered by the Policy Subprofile, then the Policy Subprofile shall not be identified as a RegisteredSubprofileForProfile.

Table 285: Properties for PolicyCapabilities

Property	Type	Flags	Description / Notes
InstanceID	String	R	
ElementName	String	R	
PolicyLevelsSupported[]	string	R	Values {}
CQLLevelsSupported[]	string	R	Values {}
FCAPSSupported[]	string	R	Values {}
SynchronousMethodsSupported[]	string	R	Values { "" }
AsynchronousMethodsSupported[]	string	R	Values { "" }

19.8.5 PolicyConditionInPolicyRule Class

A PolicyRule aggregates zero or more instances of the PolicyCondition class, via the PolicyConditionInPolicyRule association. A Rule that aggregates zero Conditions is not valid; it may, however, be in the process of being defined. Note that a PolicyRule should have no effect until it is valid.

PolicyConditionInPolicyRule is subclassed from PolicyConditionStructure. Its properties are summarized in Table 286.

Created by: PolicyAction can be "Static" (predefined by the provider).

Deleted by: "Static" PolicyActions shall not be deleted.

Modified by: "Static" PolicyActions shall not be modified.

Conditional Requirements: none

Table 286: Properties for PolicyConditionInPolicyRule

Property	Type	Flags	Description / Notes
----------	------	-------	---------------------

Table 286: Properties for PolicyConditionInPolicyRule (Continued)

GroupNumber	Uint16		Unsigned integer indicating the group to which the contained PolicyCondition belongs. This integer segments the Conditions into the ANDed sets (when the ConditionListType is \"DNF\") or, similarly, into the ORed sets (when the ConditionListType is \"CNF\").
ConditionNegated	Boolean		Indication of whether the contained PolicyCondition is negated. TRUE indicates that the PolicyCondition IS negated, FALSE indicates that it IS not negated.
GroupComponent	REF	R	This property represents the PolicyRule that contains one or more PolicyConditions.
PartComponent	REF	R	This property holds the name of a PolicyCondition contained by one or more PolicyRules.

19.8.6 PolicyRule Class

PolicyRule is subclassed from PolicySet. It is The central class used for representing the 'If Condition then Action' semantics of a policy rule. Its properties are summarized in Table 288.

Created by: implicitly by provider.

Deleted by: intrinsic DeleteInstance and/or implicitly. Indicate whether locking is needed.

Modified by: intrinsic SetInstance and/or implicitly. Indicate whether locking is needed.

Conditional Requirements: none

Table 287: Properties for PolicyRule

Property	Type	Flags	Description / Notes
ElementName	string		Another user-friendly name
CommonName	string		user-friendly name of policy object
PolicyKeywords[]	string		FCAPS strings
PolicyDecisionStrategy	uint16		PolicyDecisionStrategy defines the evaluation method used for policies contained in the PolicySet. FirstMatching enforces the actions of the first rule that evaluates to TRUE. It is the only value currently defined. Values { "First Matching" }

Table 287: Properties for PolicyRule

Enabled	uint16	R	Indicates whether this PolicySet is administratively enabled, administratively disabled, or enabled for debug. The "\"EnabledForDebug\" property value is deprecated and, when it or any value not understood by the receiver is specified, the receiving enforcement point treats the PolicySet as \"Disabled\". To determine if a PolicySet is \"Enabled\", the containment hierarchy specified by the PolicySetComponent aggregation is examined and the Enabled property values of the hierarchy are ANDed together. Thus, for example, everything aggregated by a PolicyGroup may be disabled by setting the Enabled property in the PolicyGroup instance to \"Disabled\" without changing the Enabled property values of any of the aggregated instances. The default value is 1 (\"Enabled\"). Values { "Enabled", "Disabled", "Enabled For Debug" }
SystemCreationClassName	string	R	The scoping System's CreationClassName.
SystemName	string	R	The scoping System's Name.
CreationClassName	string	R	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyRuleName	string	R	A user-friendly name of this PolicyRule.
ConditionListType	uint16		Indicates whether the list of PolicyConditions associated with this PolicyRule is in disjunctive normal form (DNF), conjunctive normal form (CNF), or has no conditions (i.e., is an UnconditionalRule) and is automatically evaluated to \"True.\" The default value is 1 (\"DNF\"). Values { "Unconditional Rule", "DNF", "CNF" }
RuleUsage	string		A free-form string that can be used to provide guidelines on how this PolicyRule should be used.
SequencedActions	uint16		This property gives a policy administrator a way of specifying how the ordering of the PolicyActions associated with this PolicyRule is to be interpreted. Three values are supported: - mandatory(1): Do the actions in the indicated order, or don't do them at all. - recommended(2): Do the actions in the indicated order if you can, but if you can't do them in this order, do them in another order if you can. - dontCare(3): Do them -- don't care about the order. The default value is 3 (\"DontCare\"). Values { "Mandatory", "Recommended", "Dont Care" }

Table 287: Properties for PolicyRule

ExecutionStrategy	uint16		<p>ExecutionStrategy defines the strategy to be used in executing the sequenced actions aggregated by this PolicyRule. There are three execution strategies:</p> <p>Do Until Success - execute actions according to predefined order, until successful execution of a single action.</p> <p>Do All - execute ALL actions which are part of the modeled set, according to their predefined order. Continue doing this, even if one or more of the actions fails.</p> <p>Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance.</p> <p>Values { "Do Until Success", "Do All", "Do Until Failure" }</p>
-------------------	--------	--	---

19.8.7 PolicyRuleInSystem Class

An association that links a PolicyRule to the System in whose scope the Rule is defined. It represents a relationship between a System and a PolicyRule used in the administrative scope of that system (e.g., AdminDomain, ComputerSystem). The Priority property is used to assign a relative priority to a PolicyRule within the administrative scope in contexts where it is not a component of another PolicySet.

PolicyRuleInSystem is subclassed from PolicySetInSystem.

Created by: implicitly by provider. Indicate whether locking is needed.

Deleted by: intrinsic DeleteInstance and/or implicitly. Indicate whether locking is needed.

Modified by: intrinsic SetInstance and/or implicitly. Indicate whether locking is needed.

Conditional Requirements: none

Table 288: Properties for PolicyRuleInSystem

Property	Type	Flags	Description / Notes
Priority	UInt16		The Priority property is used to specify the relative priority of the referenced PolicySet (PolicyRule) when there are more than one PolicySet instances applied to a managed resource that are not PolicySetComponents and, therefore, have no other relative priority defined. The priority is a non-negative integer; a larger value indicates a higher priority.
Antecedent	REF	R	The System in whose scope a PolicyRule is defined.
Dependent	REF	R	A PolicyRule named within the scope of a System.

19.8.8 PolicySetAppliesToElement Class

PolicySetAppliesToElement makes explicit which PolicySets (i.e., policy rules and groups of rules) are currently applied to a particular Element. This association indicates that the PolicySets that are appropriate for a ManagedElement (specified using the PolicyRoleCollection aggregation) have actually been deployed in the policy management infrastructure. One or more QueryCondition or MethodAction instances may reference the PolicySetAppliesToElement association as part of its query. PolicySetAppliesToElement shall not be used if the associated PolicySet, (collectively though its rules, conditions, and actions), does not make use of the association. Note that if the named Element refers to a Collection, then the PolicySet is assumed to be applied to all the members of the Collection.

For the Pool Management Policy subprofile, PolicySetAppliesToElement shall point to a valid concrete StoragePool.

PolicySetAppliesToElement is not subclassed from anything.

Created by: The CreateInstance intrinsic method.

Deleted by: The DeleteInstance intrinsic method (or implicitly on the deletion of either end of the association).

Modified by: N/A

Conditional Requirements: none

Table 289: Properties for PolicySetAppliesToElement

Property	Type	Flags	Description / Notes
PolicySet	REF	R	The PolicyRules and/or groups of rules that are currently applied to an Element.
ManagedElement	REF	R	The concrete StoragePool to which the PolicySet applies.

19.8.9 QueryCondition Class

QueryCondition defines the criteria for generating a set of QueryConditionResult instances that result from the contained query.

QueryCondition is subclassed from PolicyCondition.

Created by: implicitly by provider. Indicate whether locking is needed.

Deleted by: intrinsic DeleteInstance and/or implicitly. Indicate whether locking is needed.

Modified by: intrinsic SetInstance and/or implicitly. Indicate whether locking is needed.

Conditional Requirements: none

Table 290: Properties for QueryCondition

Property	Type	Flags	Description / Notes
ElementName	string		Another user-friendly name
CommonName	string		User-friendly name of policy object
PolicyKeywords[]	string		FCAPS strings

Table 290: Properties for QueryCondition

SystemCreationClassName	String	R	The name of the class or the subclass used in the creation of the System object in whose scope this PolicyCondition is defined.
SystemName	String	R	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName	String	R	For a rule-specific PolicyCondition, the CreationClassName of the PolicyRule object with which this Condition is associated. For a reusable Policy Condition, a special value, 'NO RULE', should be used to indicate that this Condition is reusable and not associated with a single PolicyRule.
PolicyRuleName	String	R	For a rule-specific PolicyCondition, the name of the PolicyRule object with which this Condition is associated. For a reusable PolicyCondition, a special value, 'NO RULE', should be used to indicate that this Condition is reusable and not associated with a single PolicyRule.
CreationClassName	String	R	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyConditionName	String	R	A user-friendly name of this PolicyCondition.
Query	String	R	A query expression that defines the condition(s) under which QueryConditionResult instances will be generated. For the EXACT query string that shall be supported see the above QueryCondition section.
QueryLanguage	Uint16	R	The language in which the query is expressed. SMI-S only recognizes "CQL". Other query languages may be encoded for vendor specific support, but only CQL is supported for SMI-S interoperability. Values {"CQL", "DMTF Reserved", "Vendor Reserved"}
Trigger	Boolean	R	If Trigger = true, and with the exception of any PolicyTimePeriodConditions, PolicyConditions of this PolicyRule are not evaluated until this 'triggering' condition query is true. There shall be no more than one QueryCondition with Trigger = true associated with a particular PolicyRule.

19.8.10 QueryConditionResult Class

QueryConditionResult is a QueryResult class used to communicate the results of evaluation of the query specified in the referenced QueryCondition.

QueryConditionResult is subclassed from QueryResult.

Created by: This is created as a side effect of evaluation of a QueryCondition.

Deleted by: This is implicitly deleted after execution of a PolicyRule.

Modified by: N/A.

Conditional Requirements: none.

This shall be supported for all Policy Subprofile implementations, except for Policy Subprofiles that only support "type 1" Static Policy Rules.

Table 291: Properties for QueryConditionResult

Property	Type	Flags	Description / Notes
IndicationIdentifier	string		An identifier for the Indication. This property is similar to a key value in that it can be used for identification,
CorrelatedIndications[]	string		A list of IndicationIdentifiers whose notifications are correlated with (related to) this one.
IndicationTime	datetime		The time and date of creation of the Method call.
SelectCriteria	string		The output of one row of a Query, formatted as an EmbeddedObject (with a predefined class name of \"CIM_QueryResultInstance\"). The embedded properties contained in this property shall match in both name and type to a corresponding select-list entry in the select-criteria portion of the Query.
QueryConditionPath	string	R	This shall be a fully qualified, WBEM URI Mapping Specification-based Object Name to the QueryCondition instance that produced this result.

19.9 Dependencies on Other Standards

Table 292: Pool Management Policy Subprofile Standard Dependencies

Standard	Version	Organization
CIM Specification	2.2	DMTF
CIM Operations over HTTP	1.1	DMTF
CIM Query Specification	1.0	DMTF
CIM Schema	2.9	DMTF

19.9.1 CIM Server Requirements

19.9.2 Functional Profiles

Table 293: Pool Management Policy Subprofile Functional Profile Requirements

Profile Required	Functional Group	Dependency
YES	Basic Read	None
YES	Basic Write	Basic Read
YES	Instance Manipulation	Basic Write
NO	Schema Manipulation	Instance Manipulation
YES	Association Traversal	Basic Read
Limited	Query Execution	Basic Read
NO	Qualifier Declaration	Schema Manipulation
YES	Indication	None

19.9.3 Extrinsic Methods

None.

19.9.4 Discovery

As a subprofile, the Pool Management Policy subprofile is not be advertised via SLP. It would, however, be identified as a RegisteredSubprofile for any advertised Profile that supports any Policy based function.

EXPERIMENTAL

DEPRECATED

Clause 20: Pool Manipulation Capabilities, and Settings Subprofile

The functionality of the LUN Creation and Pool Manipulation Capabilities, and Settings Subprofiles has been subsumed by the Clause 5: Block Services Package.

The Pool Manipulation Capabilities, and Settings Subprofile is defined in section 7.3.3.10 of SMI-S 1.0.2.

DEPRECATED

EXPERIMENTAL

Clause 21: Storage Server Asymmetry Profile

21.1 Description

21.1.1 Overview

High-availability storage servers using multiple redundant storage processors exhibit a range of interrelated behavior involving load-balancing, ports, and failover. This profile provides for management of these aspects.

Many such systems have the concept of a storage resource (either a RAID group or a storage volume) having an assignment to, or affinity for, one of the storage processors in a redundant set. This affinity may have one or more underlying architectural reasons for existing. Examples are both front-end (target) port connectivity with and between processors, cache processing, virtualization(RAID) processing, or connectivity partitioning of back end resources.

When the storage processor for which the storage resource has affinity fails, the resource is taken over by one of the other processors in the redundancy set

When both storage processors are healthy, the ports on the storage processor for which the storage resource as affinity provide full bandwidth access to the resource. The ports on the “other” storage processors provide full, limited, or standby access, depending on implementation

21.1.2 Relationship to Multiple Computer System sub-profile

This profile is a component profile (or subprofile) and extends the functionality of the Multiple Computer System subprofile, which in turn references this profile as a supported profile. This profile requires the use of the Multiple Computer System subprofile.

A separate profile was created for two purposes. Firstly, the functionality of Asymmetric Access is largely storage-related and since the MCS is a common profile, the asymmetry functions are specified separately. Secondly, although some asymmetric behavior may be modeled using provisions under the Multiple Computer System profile regarding aggregating resources to the lowest level ComputerSystem that represents availability, many implementations aggregate all resources to the top-level ComputerSystem, even though these implementations exhibit asymmetric behavior. These resources include CIM_StorageVolumes, CIM_StoragePools, CIM_ProtocolControllers, CIM_ProtocolEndpoints, and the CIM_StorageConfiguration and CIM_ControllerConfiguration services. CIM_LogicalPorts are usually aggregated to the lower level systems that represent the storage processors.

Asymmetric behavior is modeled through constructs in this profile and is independent of SystemDevice and Hosting associations in Multiple Computer System.

21.1.3 Relationship to Masking and Mapping sub-profile

The Masking and Mapping subprofile provides the means to expose storage volumes to initiators through front-end ports. In systems with asymmetric behavior, Masking and Mapping alone does not provide for determining whether the action of the ExposePaths method will result in the creation of a path that is primary, secondary, or standby from a performance standpoint.

This profile is does not formally extend Masking and Mapping but augments it's functionality by providing the model constructs to support this determination by a client. It does this with model relationships directly between groups of front-end ports (which are represented by subclasses of CIM_ProtocolEndpoint) and groups of storage resources, independent of the implementation of Mapping and Masking “View” CIM_ProtocolControllers. This is necessary because some implementations may not generate “primary” and “standby” view/mappings for the ports on each

storage processor but instead share common view controllers between storage processors, making it impossible to use the “view” CIM_ProtocolController to group ports with volumes.

21.1.4 Relationship to T10

This sub-profile supports the passive management of the functionality defined in the Target Port Group Access States clause of the T10 SPC-4 specification.

21.1.5 Behavior, Characteristics, and Capabilities

The behavioral use cases for redundant systems are used to derive asymmetry characteristics which in turn are used to distill capabilities for the profile that allow a client to interpret the asymmetric model objects.

21.1.5.1 Port Failover

The first differentiator to consider when trying to classify asymmetric behavior is target port failover behavior. Front-end ports on storage processors in a redundancy set exhibit either transparent or non-transparent behavior when the supporting storage processor fails

21.1.5.1.1 Transparent

In transparent failover, a storage processor can support multiple virtual ports, that is the ports that it normally has, and the functionality of ports from a failed storage processor in the same redundancy set. Stated another way, when a storage processor fails, its ports don't fail, they fail over to a healthy storage processor. This mode is called transparent because the host sees only a transient loss of access to the port. The port itself is still present after the failover.

21.1.5.1.2 Non-Transparent

In this type of architecture, the ports supported by a storage processor fail when the processor fails. Access to the storage volumes that were exposed through the failed ports is provided through ports on a surviving processor.

21.1.5.2 Port Asymmetry

Healthy storage servers have variant functionality with respect to access to volumes through ports on different storage processors. This may be related to the affinity of such volumes (or the pools to which they belong) to storage processors as described in Storage Resource Affinity, below. In some systems, there is “full” bandwidth access to a volume through both ports on processor A and ports on processor B. This is actually symmetric access. In other cases, access to a volume is full bandwidth access through ports on the storage processor (“this”) for which the volumes have affinity and “reduced” bandwidth access through ports on the “other” processor. The third variation is the there is no access at all, other than inquiry type commands, through ports on the “other” processor, until the processor for which the volumes have affinity fails. This functionality is reflexive in that there is full access to volumes having affinity for the “other” processor through ports on that processor, while there is reduced access or no access to volumes affinity to “other” through ports on “this”.

21.1.5.3 Storage Resource Affinity

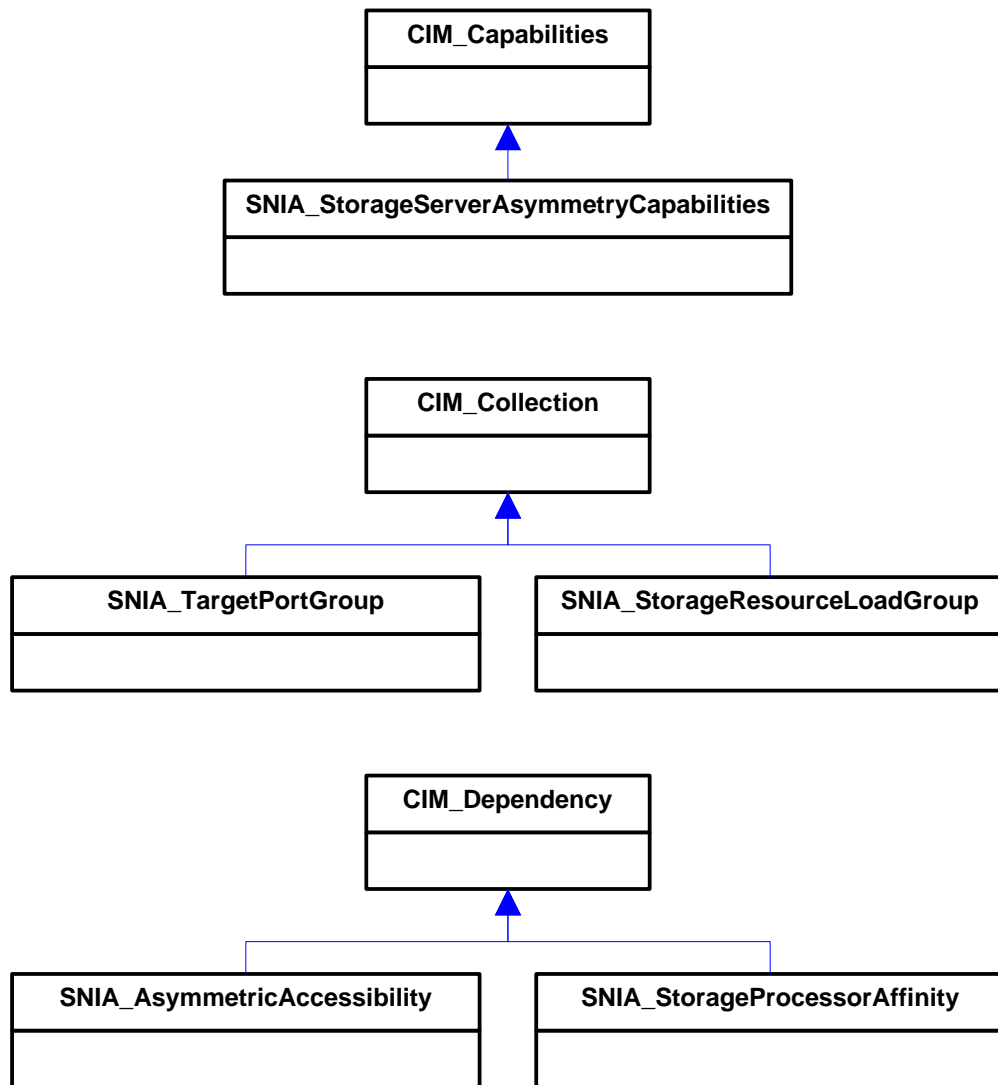
Storage resource affinity is the behavior that in many redundant servers, storage resources, either individual volumes or RAID groups (also called RAID sets or RAID ranks) and thus the volumes allocated from them, have an affinity for a given storage processor in a redundancy set. This affinity may stem from allocation of non-dual ported drives to a processor or assignment of these resources to a processor for cache or RAID processing architectural considerations. Managing this affinity is necessary on redundant systems as part of a static load balancing strategy. This is true even when the front-end ports exhibit symmetric access behavior, because assigning all resources to one storage processor may degrade the overall system throughput.

21.1.6 Model

21.1.6.1 Classes

This profile introduces five new classes. These include one capabilities class, two collections, and two associations.

Figure 80: Storage Asymmetry Class Hierarchy



21.1.6.1.1 Asymmetry Capabilities

This class contains properties that enable a client to determine the combination of asymmetry characteristics implemented by the subject storage system. More specifically, they guide the client algorithms in interpretation of the instances of the asymmetry classes and associations. The capabilities are detailed in the CIM_Elements section

21.1.6.1.2 TargetPortGroup

This sub-class of CIM_Collection aggregates the instances of CIM_ProtocolEndpoint or its subclasses that represent the ports on a storage processor (represented by CIM_ComputerSystem). The ports are aggregated because their relationship to the storage processors for failover and to the storage resources for accessibility are the same.

Whether ProtocolEndpoint is used directly or one of its subclasses is used depends on which Target Port component profile is implemented by the storage server.

21.1.6.1.2.1 Multiple Hierarchical TargetPortGroups

Some Target Port profiles, such as the ISCS Target Port profile, may have a hierarchy of ProtocolEndpoints. Each layer of ProtocolEndpoints in the hierarchy that can have affinity for a storage processor may be aggregated by a separate TargetPortGroup. This enables a client to determine which lower-level ProtocolEndpoints in the hierarchy may be used to create upper-level ProtocolEndpoints with the desired affinity. An example is the need to select TCPProtocolEndpoints with the same affinity for a storage processor when attempting to create an iSCSIProtocolEndpoint for that same processor.

21.1.6.1.3 StorageResourceLoadGroup

This sub-class of CIM_Collection aggregates either the storage volumes or storage pools that have the same affinity for a storage processor. What type of storage resource is aggregated depends on whether the pools have affinity or are common between processors and just the individual volumes have affinity. There is a capabilities property to specify this. There is one static instance of StorageResourceLoadGroup for each storage processor, with a single exception described in 21.1.6.1.3.1.

21.1.6.1.3.1 Single Volume Accessibility Override.

Some implementations allow for the normal “healthy” accessibility to a Storage Volume on the “other” storage processor through ports on “this” storage processor to be overridden. Normally in an asymmetric system this accessibility is “Standby” or “Active-NonOptimized”. This override gives Active-Optimized, or full bandwidth access to this single volume.

This is modeled by an additional instance of StorageResourceLoadGroup that collects the subject volume together with an instance of AsymmetricAccessibility that associates that special StorageResourceLoadGroup with the TargetPortGroup. The properties on AsymmetricAccessibility reflect the override. This profile does not support the action that creates or removes the override. Methods of this profile that relate to assignment of affinity operate on the default static instance of StorageResourceLoadGroup only.

21.1.6.1.4 StorageProcessorAffinity

This sub-class of CIM_Dependency associates instances of StorageResourceLoadGroup in a Redundancy Set to each instance of CIM_ComputerSystem representing a storage processor. Primary and Active properties are used to surface what the affinity is in both healthy and failed situations, and which storage processor owns the resource group which is where the Load Group will fail back to.

21.1.6.1.5 Asymmetric Accessibility

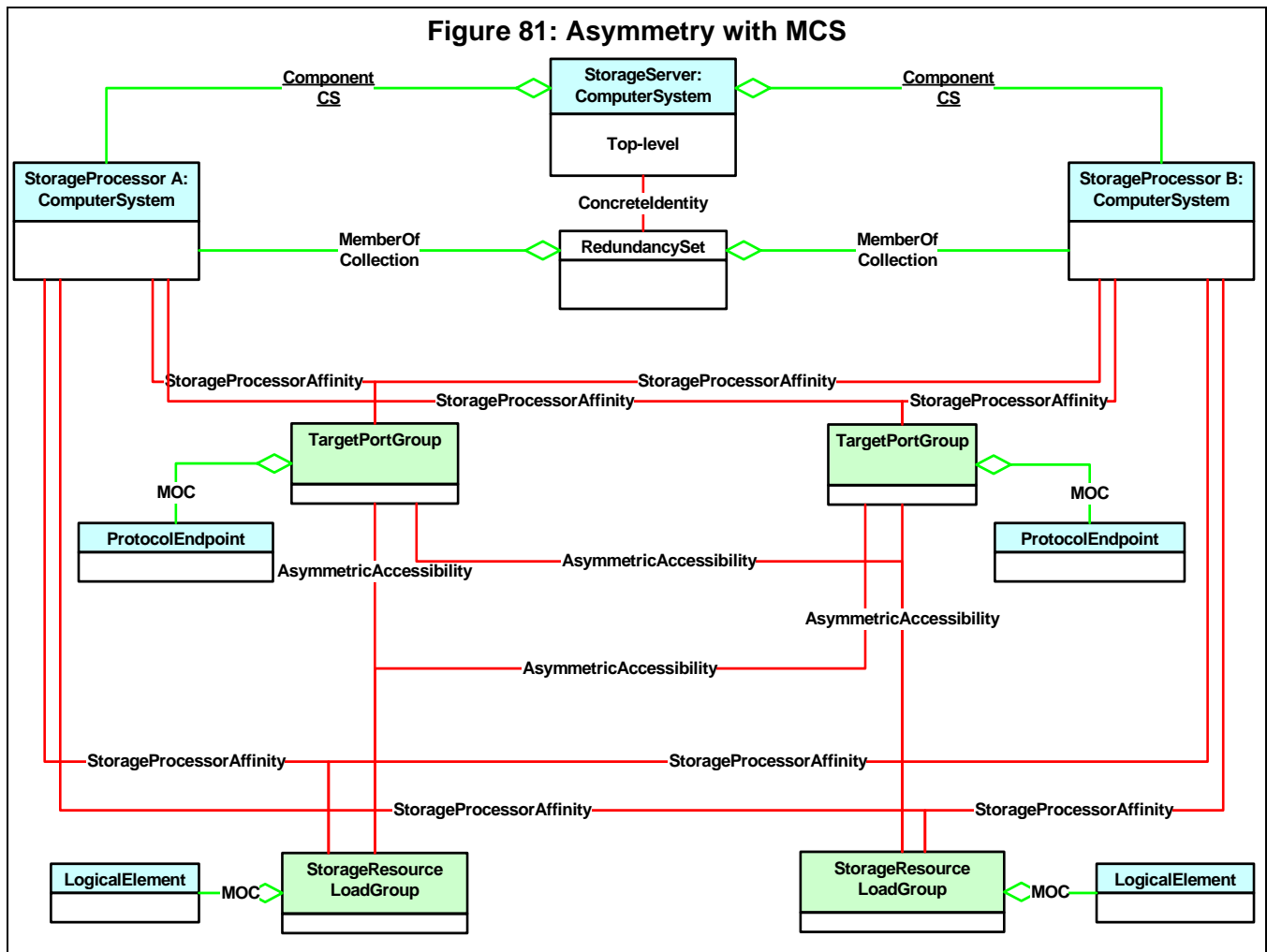
This sub-class of CIM_Dependency associates instances of StorageResourceLoadGroup in a Redundancy Set to each instance of SNIA_TargetPortGroup in the same RedundancySet. The AccessibilityState surfaces both the current and normal (healthy) accessibility of volumes in the LoadGroup from ports in the Port Group.

21.1.6.2 Instance Diagrams

The following instance diagrams provide show various asymmetry use cases. They are extensions of the MCS model, but for readability do not show Hosting and SystemDevice relationships. All instances are scoped to the top-level system.

The following general instance diagram shows the Asymmetry instances in context of the Multiple Computer System profile for a dual redundant storage server.

The subsequent instance diagrams do not show the RedundancySet-related classes.



21.1.6.2.1 Multiple Tiers of Systems

Not shown is a system that has three tiers (see Clause 32: Multiple Computer System Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles*). This type of system may aggregate storage processors into more than one redundant-failover sub-system. These subsystems are then clustered in a non-failover, but load-balancing relationship to form the top-level storage server. In this type of system, **StorageProcessorAffinity** associations would be contained within failover subsystems, but **AsymmetricAccessibility** associations may span subsystem boundaries to reflect mid-level load-balancing paths.

21.1.6.2.2 Non-Transparent Asymmetry Cases

The following pair of instance diagrams show the model for healthy and failed situations in a non-transparent port implementation. Because the ports and thus the Target Port Group do not failover, there is no need for a StorageResourceAffinity association from the Target Port Group on the storage processor to which the ports belong to the “Other” storage processor.

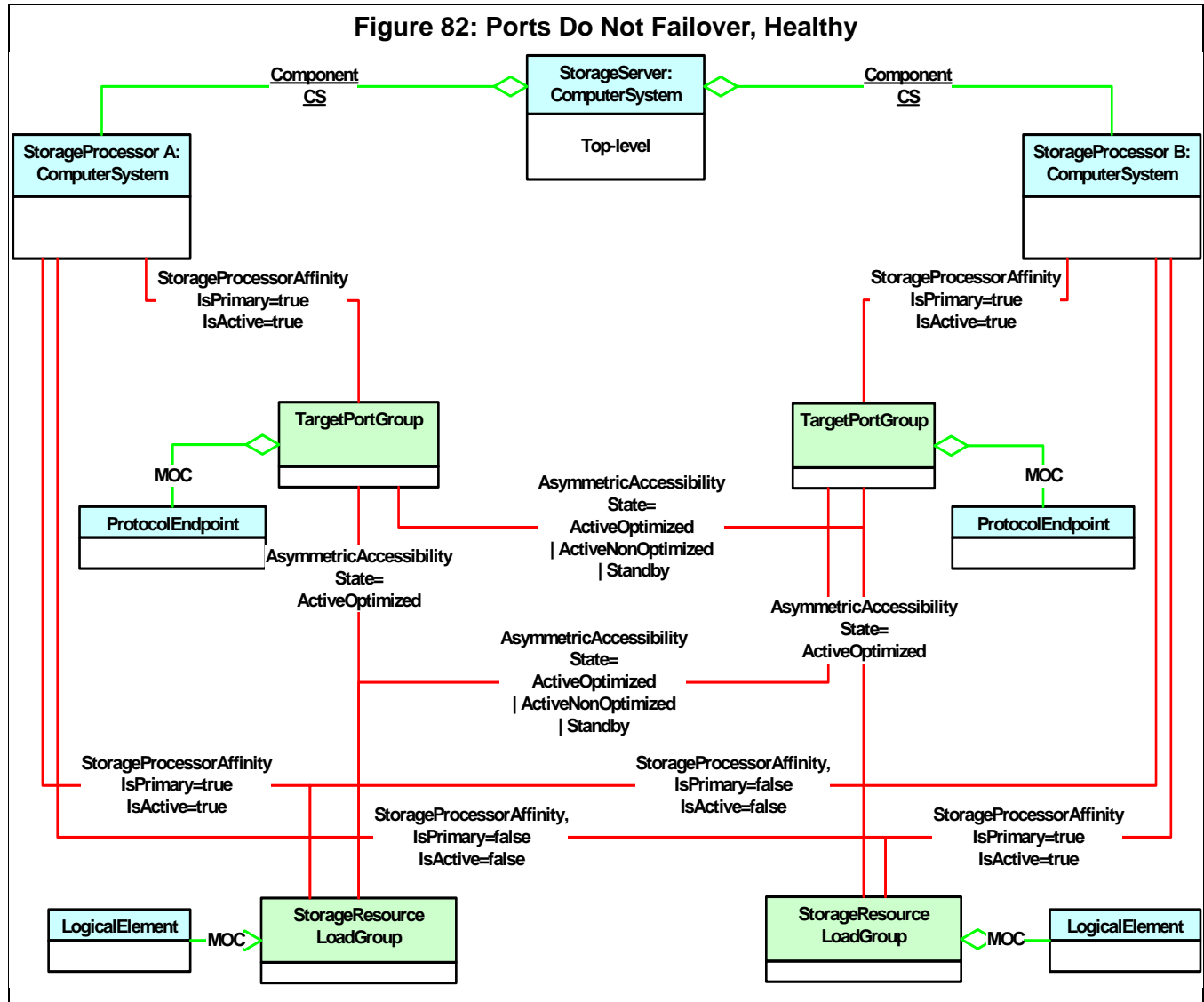
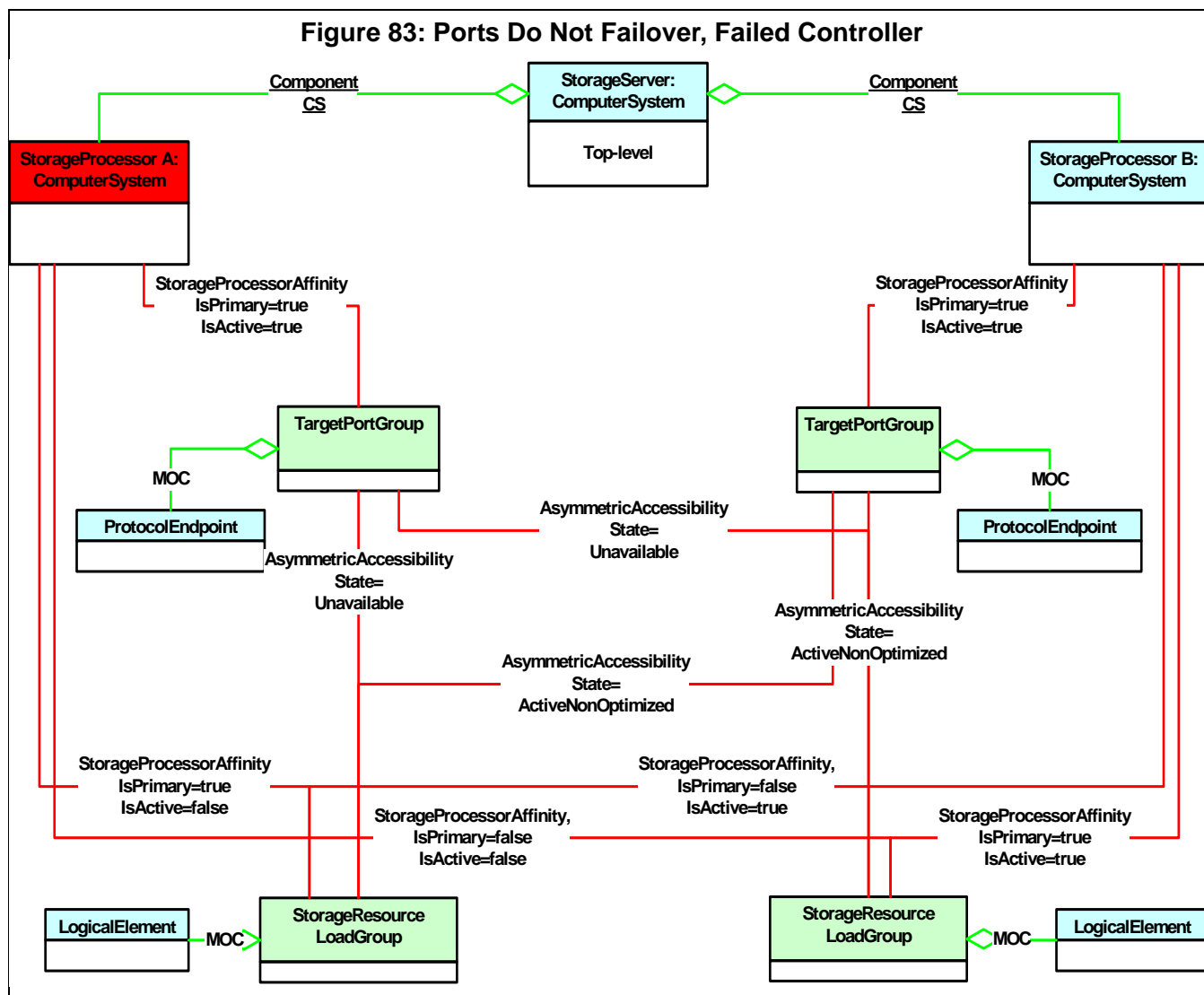
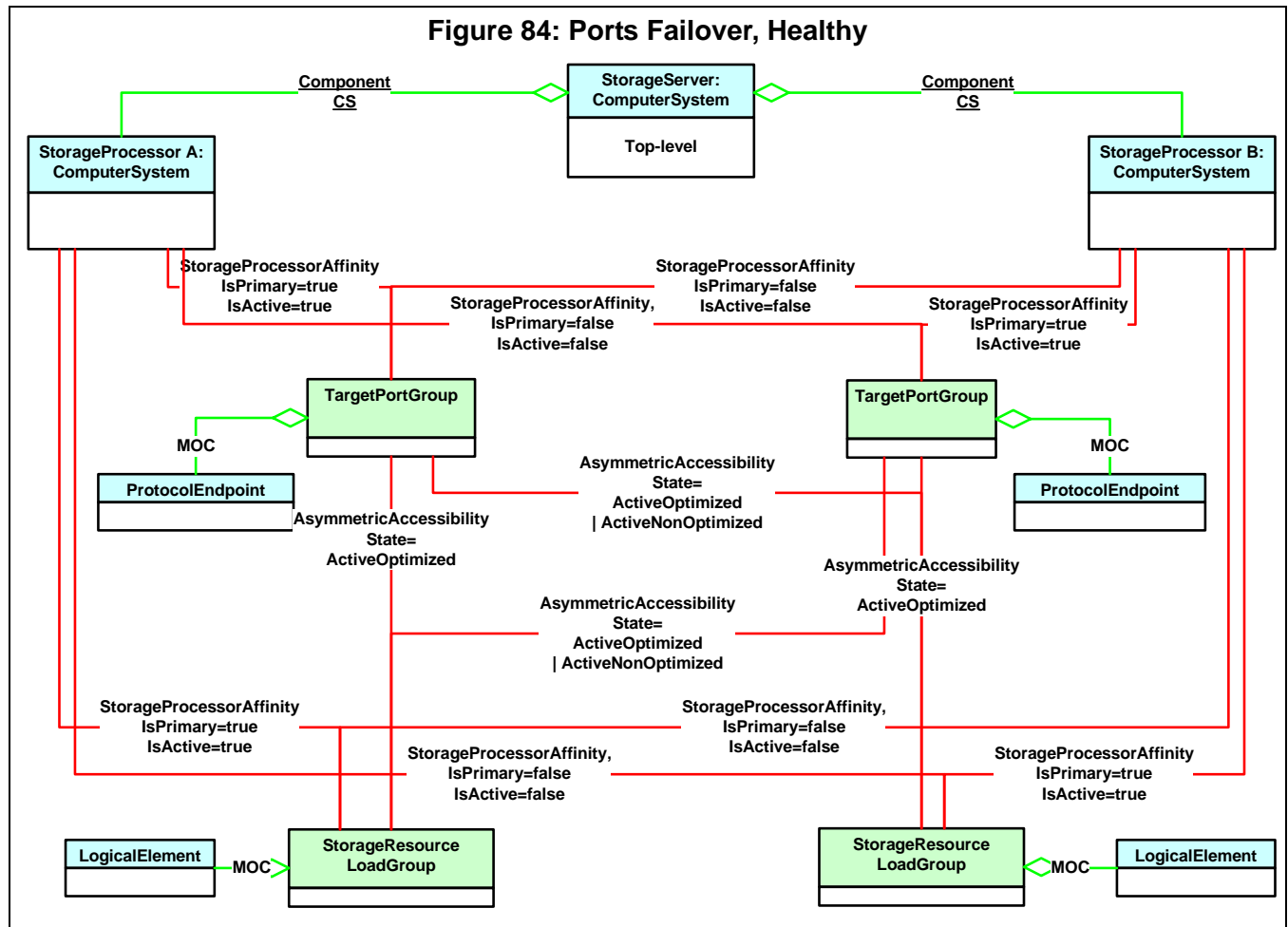


Figure 83: Ports Do Not Failover, Failed Controller



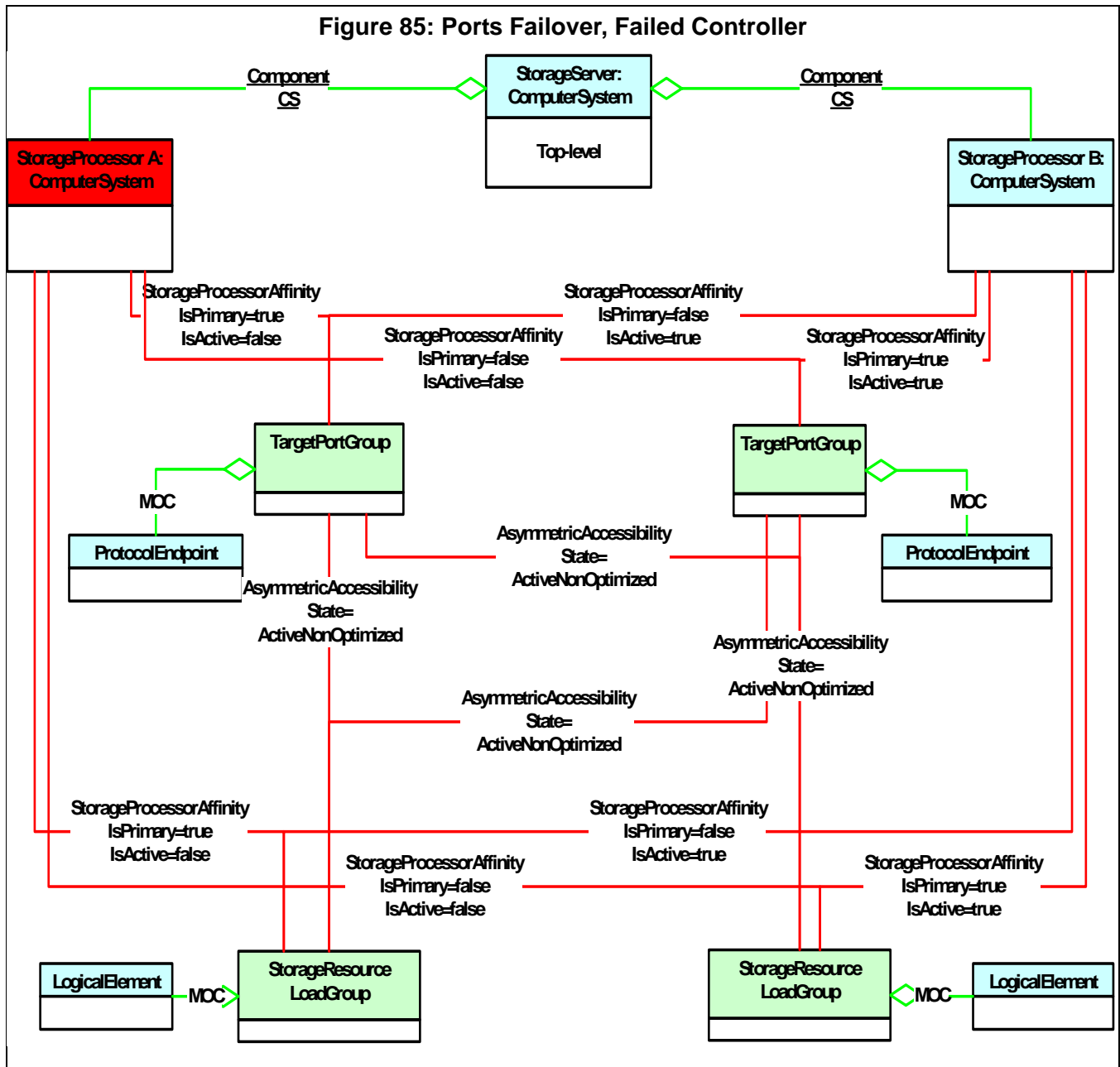
21.1.6.2.3 Transparent Asymmetry Cases

The following pair of instance diagrams show the model for healthy and failed situations in a transparent failover



port implementation.

Figure 85: Ports Failover, Failed Controller



21.2 Health and Fault Management Consideration

None

21.3 Cascading Considerations

None.

21.4 Supported Profiles, Subprofiles, and Packages

None.

21.5 Methods of the Profile

21.5.1 Assign Storage Resource Affinity

This profile specific method of CIM_StorageConfigurationService starts a job to assign affinity of a StoragePool(s) or StorageVolume(s) to a storage processor. At the conclusion of the operation, the resource will be associated by CIM_MemberOfCollection to the StorageResourceLoadGroup with the primary affinity for the specified storage processor. The existing instance of CIM_MemberOfCollection to the existing StorageResourceLoadGroup is deleted.

Support for this method is indicated by the presence of an instance of StorageServerAsymmetryCapabilities in which the property StorageResourceAffinityAssignable is 'true'. If 0 is returned, the function completed successfully and no ConcreteJob instance was required. If 4096/0x1000 is returned, a job will be started to assign the element. The Job's reference will be returned in the output parameter Job.

AssignStorageResourceAffinity

IN, string **ResourceType**

This specifies whether the resource is a StoragePool(= 2) or StorageVolume(= 3).

OUT, CIM_ConcreteJob REF **JOB**,

Reference to a job which may be created (may be null if job completed).

IN, CIM_ComputerSystem REF **StorageProcessor**

Reference to the storage processor to which to assign the resource.

IN, CIM_LogicalElement REF **StorageResources[]**

Array of references to storage resource instances to be assigned.

21.5.1.1 Return Codes

Completed with No Error - 0

Not Supported - 1

Unknown - 2

Timeout - 3

Failed - 4

Invalid Parameter - 5

In Use - 6

Method Parameters Checked - Job Started - 4096

Size Not Supported - 4097

21.6 Client Considerations and Recipes

21.6.1 Determine which ports provide full bandwidth access to a storage element.

```
//
// DESCRIPTION
// Determine which ports on a storage server provide full
// bandwidth access to a storage volume.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The Top-Level ComputerSystem representing the target system of interest
// has been previously identified and defined in the $StorageServer-> variable.
//
// 2. The CIM_StorageVolume of interest has been previously identified
// and defined in the $StorageVolume-> variable.
//
// MAIN
// Step 1. Locate the instance of SNIA_StorageServerAsymmetryCapabilities
//          associated to the
// target ComputerSystem to insure the profile is supported.
//
$StorageServerAsymmetryCapabilities[] = Associators($StorageServer->,
    "CIM_ElementCapabilities",
    "SNIA_StorageServerAsymmetryCapabilities",
    "ManagedElement",
    "Capabilities",
    false,
    false,
    {"StorageResourceSymmetryCapability"})

if ($StorageServerAsymmetryCapabilities[] == null ||
    $StorageServerAsymmetryCapabilities[0].length != 1) {
    <ERROR! The profile capabilities could not be found>
}

// Step 2. Check to see if this server has symmetric behavior.
// If so, exit here as an optimization.
//
if ( $StorageServerAsymmetryCapabilities[0].StorageResourceSymmetryCapability ==
    2 ) // Symmetric
    { <EXIT! Symmetric. All ports on the server provide full bandwidth access.> }

// Step 3. Find the Storage Resource Load Group to which this volume belongs.
//
$StorageResourceLoadGroup->[] = AssociatorNames($StorageVolume->,

```

Storage Server Asymmetry Profile

```
"CIM_MemberOfCollection",
"SNIA_StorageResourceLoadGroup",
"ManagedElement",
"Collection")
if ($StorageResourceLoadGroup[] == null || $StorageResourceLoadGroup[].length !=
    1)
    { <ERROR! Volume must be a member of one and only one Load Group > }

// Step 4. Find the Target Port groups whose member ports provide full
// bandwidth access to the subject volume, and collect the port references for
// each such port group.
//
$AsymmetricAccessibility[] = References($StorageResourceLoadGroup->[0],
    "SNIA_AsymmetricAccessibility",
    "Dependent",
    false,
    false,
    {"Antecedent", "NormalAccessState"})

#index = 0
for #i in $AsymmetricAccessibility[] {
    if ( $AsymmetricAccessibility[#i].NormalAccessState == 5 ) { // Active
        Optimized
        $Ports->[] = AssociatorNames($AsymmetricAccessibility[#i].Antecedent,
            "CIM_MemberOfCollection",
            "CIM_ProtocolEndpoint",
            "Collection",
            "ManagedElement")

        if ($Ports->[] != null ) {
            for #j in $Ports->[] {
                $FullAccessPorts->[#index] = $Ports->[#j]
                #index++
            }
        }
    }
}

<EXIT: $Ports will contain the references to the ProtocolEndpoints representing
    the ports which
    will give full bandwidth access to the volume.>
```

21.7 Registered Name and Version

Storage Server Asymmetry version 1.2.0

21.8 CIM Elements

Table 294: CIM Elements for Storage Server Asymmetry

Element Name	Requirement	Description
SNIA_StorageServerAsymmetryCapabilities (21.8.1)	Mandatory	This class defines the asymmetric characteristics and capabilities of a redundant storage server. The properties in this class guide client algorithms in the interpretation of the instances of StorageResourceLoadGroup, TargetPortGroup, StorageProcessorAffinity, and AsymmetricAccessibility, and also determining support for methods that affect assignment of storage resources to storage processors.
SNIA_StorageResourceLoadGroup (21.8.2)	Mandatory	StorageResourceLoadGroup aggregates either the StoragePools or the individual StorageVolumes that have the same affinity for a storage processor. The affinity of this group may change during failover or failback/rebind from one storage processor to another in a storage server. StorageResourceLoadGroup has a instance of the StorageProcessorAffinity association to each instance of CIM_ComputerSystem representing a storage processor that may host the StorageResourceLoadGroup in either a healthy or failover state. Each instance of StorageResourceLoadGroup in a storage server is also associated to each instance of TargetPortGroup in the server by the AsymmetricAccessibility class.

Table 294: CIM Elements for Storage Server Asymmetry

Element Name	Requirement	Description
SNIA_TargetPortGroup (21.8.3)	Mandatory	TargetPortGroup aggregates the ProtocolEndpoints representing a group of target ports in a storage server. The ProtocolEndpoints may be a subclass of CIM_ProtocolEndpoint as appropriate for the type of target port implemented by the storage server. The target ports are aggregated because they have the same affinity for an associated storage processor for failover and the same accessibility state to storage resources in a given StorageResourceLoadGroup. The TargetPortGroup may have either a fixed affinity for a storage processor within the server or an affinity that changes during failover from one storage processors to another. TargetPortGroup has a instance of the StorageProcessorAffinity association to each instance of CIM_ComputerSystem representing a storage processor that may host the TargetPortGroup in either a healthy or failover state. Each instance of TargetPortGroup in a storage server is also associated to each instance of StorageResourceLoadGroup in the server by the AsymmetricAccessibility class.
CIM_ElementCapabilities (To Top-level ComputerSystem) (21.8.4)	Mandatory	
SNIA_StorageProcessorAffinity (Target Port Group) (21.8.5)	Mandatory	Indicates a processing affinity and state between a TargetPortGroup and a ComputerSystem representing a storage processor in a redundant storage server. The processor can host the group in either a healthy or failover state. Instances of this association are static, one for each combination of StorageResourceLoadGroup and ComputerSystem in the RedundancySet.
SNIA_StorageProcessorAffinity (StorageResourceLoadGroup) (21.8.6)	Mandatory	Indicates a processing affinity and state between a TargetPortGroup and a ComputerSystem representing a storage processor in a redundant storage server. The processor can host the group in either a healthy or failover state. Instances of this association are static, one for each combination of StorageResourceLoadGroup and ComputerSystem in the RedundancySet.

Table 294: CIM Elements for Storage Server Asymmetry

Element Name	Requirement	Description
SNIA_AsymmetricAccessibility (21.8.7)	Mandatory	This association indicates the accessibility of StorageVolumes in the StorageResourceLoadGroup through ports in the associated TargetPortGroup.
CIM_MemberOfCollection (iSCSI Target Port Group) (21.8.8)	Conditional	Conditional requirement: Requires TargetPortGroup to aggregate CIM_iSCSIProtocolEndpoint. Used to aggregate iSCSI Target Ports in a Target Port Group.
CIM_MemberOfCollection (SCSI Target Port Group) (21.8.9)	Conditional	Used to aggregate DA, FC, SPI, or SAS Target Ports in a Target Port Group.
CIM_MemberOfCollection (SB Target Port Group) (21.8.10)	Conditional	Conditional requirement: Requires TargetPortGroup to aggregate SNIA_SBProtocolEndpoint. Used to aggregate SB Target Ports in a Target Port Group.
CIM_MemberOfCollection (SATA Target Port Group) (21.8.11)	Conditional	Conditional requirement: Requires TargetPortGroup to aggregate CIM_ProtocolEndpoint. Used to aggregate SATA Target Ports in a Target Port Group.
CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Volumes) (21.8.12)	Conditional	Conditional requirement: Requires StorageResourceLoadGroup to aggregate CIM_StorageVolume. Aggregates Storage Volumes in a Storage Resource Load Group.
CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools) (21.8.13)	Conditional	Conditional requirement: Requires StorageResourceLoadGroup to aggregate CIM_StoragePool. Aggregates Storage Pools in a Storage Resource Load Group.
SNIA_StorageConfigurationService (21.8.14)	Optional	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_StorageProcessorAffinity AND SourceInstance.SNIA_StorageProcessorAffinity::IsActive <> PreviousInstance.SNIA_StorageProcessorAffinity::IsActive	Optional	Experimental CQL - Change in Affinity of a StorageResourceLoadGroup
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_StorageProcessorAffinity AND SourceInstance.IsActive <> PreviousInstance.IsActive	Mandatory	Deprecated WQL - Change in Affinity of a StorageResourceLoadGroup

Table 294: CIM Elements for Storage Server Asymmetry

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_AsymmetricAccessibility AND SourceInstance.SNIA_AsymmetricAccessibilit y::CurrentAccessState <> PreviousInstance.SNIA_AsymmetricAccessibi lity::CurrentAccessState	Optional	Experimental CQL - Modification of accessibility to a storage element
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA SNIA_AsymmetricAccessibility AND SourceInstance.CurrentAccessState <> PreviousInstance.CurrentAccessState	Mandatory	Deprecated WQL - Modification of accessibility to a storage element

21.8.1 SNIA_StorageServerAsymmetryCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 295 describes class SNIA_StorageServerAsymmetryCapabilities.

Table 295: SMI Referenced Properties/Methods for SNIA_StorageServerAsymmetryCapabilities

Properties	Flags	Requirement	Description & Notes
StorageResourceSymmetryCapability		Mandatory	If this property is set to Symmetric it indicates that the StoragePools or StorageVolumes are processed in a distributed load-balanced manner between storage processors. If this property is set to Asymmetric it indicates that the StoragePools or StorageVolumes are have a primary affinity for one storage processor.
StorageResourceType		Mandatory	If this property is set to StorageVolume it indicates that the StoragePools have symmetric behavior(or no affinity) and that the Volumes have affinity for one storage processor or the other. If this property is set to StoragePool it indicates that a StoragePool as well as the Volumes allocated from it have affinity for one storage processor or the other.
StorageResourceAffinityAssignable		Mandatory	Set to true if this storage system allows the client to specify which storage processor a storage resource is assigned to, either using one of the CreateOrModify methods or the AssignStorageResourceAffinity method on StorageConfigurationService.

Table 295: SMI Referenced Properties/Methods for SNIA_StorageServerAsymmetryCapabilities

Properties	Flags	Requirement	Description & Notes
PortGroupFailoverBehavior		Mandatory	This property specifies whether a storage server supports transparent or non-transparent failover of TargetPortGroups. If this value is 2(Port Group Fails), a TargetPortGroup will have a single StorageProcessorAffinity association to the storage processor it belongs to and will fail with. If this property has a value of 3, the TargetPortGroup will have a StorageProcessorAffinity association to each storage processor that can host its function, and the properties on the association will indicate both which processor is primary and which is currently hosting the ports in the group.
TargetPortSymmetryCapability		Mandatory	This property indicates the normal(healthy) state accessibility to volumes both in the StorageResourceLoadGroup on the same storage processor as a TargetPortGroup, and to volumes in StorageResourceLoadGroups on 'other' storage processors in the redundant server. If this value is 2(Symmetric): There is equal bandwidth access to volumes on all storage processors through target ports on this storage processor. If this value is 3(Asymmetric Non-Optimized): There is full bandwidth access to volumes in the StorageResourceLoadGroup on the same storage processor as the TargetPortGroup and degraded bandwidth access to volumes in the StorageResourceLoadGroups on the 'other' storage processors. If this value is 4(Asymmetric No Access): There is full bandwidth access to volumes in the StorageResourceLoadGroup on the same storage processor as the TargetPortGroup and no access to volumes on 'other' storage processors.

21.8.2 SNIA_StorageResourceLoadGroup

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

21.8.3 SNIA_TargetPortGroup

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

21.8.4 CIM_ElementCapabilities (To Top-level ComputerSystem)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 296 describes class CIM_ElementCapabilities (To Top-level ComputerSystem).

Table 296: SMI Referenced Properties/Methods for CIM_ElementCapabilities (To Top-level ComputerSystem)

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The Top-level Storage Sever ComputerSystem.
Capabilities		Mandatory	StorageServerAsymmetryCapabilities

21.8.5 SNIA_StorageProcessorAffinity (Target Port Group)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 297 describes class SNIA_StorageProcessorAffinity (Target Port Group).

Table 297: SMI Referenced Properties/Methods for SNIA_StorageProcessorAffinity (Target Port Group)

Properties	Flags	Requirement	Description & Notes
IsPrimary		Mandatory	This property is set to true if the TargetPortGroup is hosted by the storage processor when the processor is healthy. It is set to false if the group can be hosted by the processor when the primary storage processor for the group has failed. For each StorageResourceLoadGroup, one instance of StorageProcessorAffinity will have IsPrimary=true, the rest will have IsPrimary=false.
IsActive		Mandatory	This property is set to true if the TargetPortGroup is currently being hosted by the storage processor.

Table 297: SMI Referenced Properties/Methods for SNIA_StorageProcessorAffinity (Target Port Group)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	The storage processor for which the Port Group has affinity.
Dependent		Mandatory	The Target Port Group.

21.8.6 SNIA_StorageProcessorAffinity (StorageResourceLoadGroup)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 298 describes class SNIA_StorageProcessorAffinity (StorageResourceLoadGroup).

Table 298: SMI Referenced Properties/Methods for SNIA_StorageProcessorAffinity (StorageResourceLoadGroup)

Properties	Flags	Requirement	Description & Notes
IsPrimary		Mandatory	This property is set to true if the TargetPortGroup is hosted by the storage processor when the processor is healthy. It is set to false if the group can be hosted by the processor when the primary storage processor for the group has failed. For each StorageResourceLoadGroup, one instance of StorageProcessorAffinity will have IsPrimary=true, the rest will have IsPrimary=false.
IsActive		Mandatory	This property is set to true if the StorageResourceLoadGroup is currently being hosted by the storage processor.
Antecedent		Mandatory	The storage processor for which the Storage Resource Load Group has affinity.
Dependent		Mandatory	The Storage Resource Load Group.

21.8.7 SNIA_AsymmetricAccessibility

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: Mandatory

Table 299 describes class SNIA_AsymmetricAccessibility.

Table 299: SMI Referenced Properties/Methods for SNIA_AsymmetricAccessibility

Properties	Flags	Requirement	Description & Notes
CurrentAccessState		Mandatory	This property indicates the current accessibility state of volumes in the StorageResourceLoadGroup through ports in the TargetPortGroup. With the exception of 'Unavailable', the states are those defined by the T10 SPC-4 Target Port Group Access States clause. 2(Unavailable): The volumes are not accessible in any way. 3(Standby): No data access to the volume is possible. Status and other non-data access commands are available. 4(Active Non-Optimized): Data access to the volume is available at less than full bandwidth. 5(Active Optimized): Data access to the volume is available at full bandwidth.
NormalAccessState		Mandatory	This property indicates the accessibility state of volumes in the StorageResourceLoadGroup through ports in the TargetPortGroup when the primary storage processor hosting the groups is healthy. With the exception of 'Unavailable', the states are those defined by the T10 SPC-4 Target Port Group Access States clause. 2(Unavailable): The volumes are not accessible in any way. 3(Standby): No data access to the volume is possible. Status and other non-data access commands are available. 4(Active Non-Optimized): Data access to the volume is available at less than full bandwidth. 5(Active Optimized): Data access to the volume is available at full bandwidth.
Antecedent		Mandatory	The Port Group.
Dependent		Mandatory	The Storage Resource Load Group.

21.8.8 CIM_MemberOfCollection (iSCSI Target Port Group)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: iSCSITargetPorts

Table 300 describes class CIM_MemberOfCollection (iSCSI Target Port Group).

Table 300: SMI Referenced Properties/Methods for CIM_MemberOfCollection (iSCSI Target Port Group)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Target Port Group.
Member		Mandatory	The iSCSI Target Ports.

21.8.9 CIM_MemberOfCollection (SCSI Target Port Group)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: DATargetPorts|FCTargetPorts|SPITargetPorts|SASTargetPorts

Table 301 describes class CIM_MemberOfCollection (SCSI Target Port Group).

Table 301: SMI Referenced Properties/Methods for CIM_MemberOfCollection (SCSI Target Port Group)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Target Port Group.
Member		Mandatory	The DA, FC, SPI, or SAS Target Ports.

21.8.10 CIM_MemberOfCollection (SB Target Port Group)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: SBTargetPorts

Table 302 describes class CIM_MemberOfCollection (SB Target Port Group).

Table 302: SMI Referenced Properties/Methods for CIM_MemberOfCollection (SB Target Port Group)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Target Port Group.
Member		Mandatory	The The SB Target Ports.

21.8.11 CIM_MemberOfCollection (SATA Target Port Group)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: SATATargetPorts

Table 303 describes class CIM_MemberOfCollection (SATA Target Port Group).

Table 303: SMI Referenced Properties/Methods for CIM_MemberOfCollection (SATA Target Port Group)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Target Port Group.
Member		Mandatory	The SATA Target Ports.

21.8.12 CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Volumes)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: VolumeAffinity

Table 304 describes class CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Volumes).

Table 304: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Volumes)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Storage Resource Load Group.
Member		Mandatory	The Storage Volumes.

21.8.13 CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools)

Created By: Static

Modified By: External

Deleted By: Static

Class Mandatory: PoolAffinity

Table 305 describes class CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools).

Table 305: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools)

Properties	Flags	Requirement	Description & Notes
Collection		Mandatory	The Storage Resource Load Group.

Table 305: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Storage Resource Load Group aggregating Storage Pools)

Properties	Flags	Requirement	Description & Notes
Member		Mandatory	The StoragePools.

21.8.14 SNIA_StorageConfigurationService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 306 describes class SNIA_StorageConfigurationService.

Table 306: SMI Referenced Properties/Methods for SNIA_StorageConfigurationService

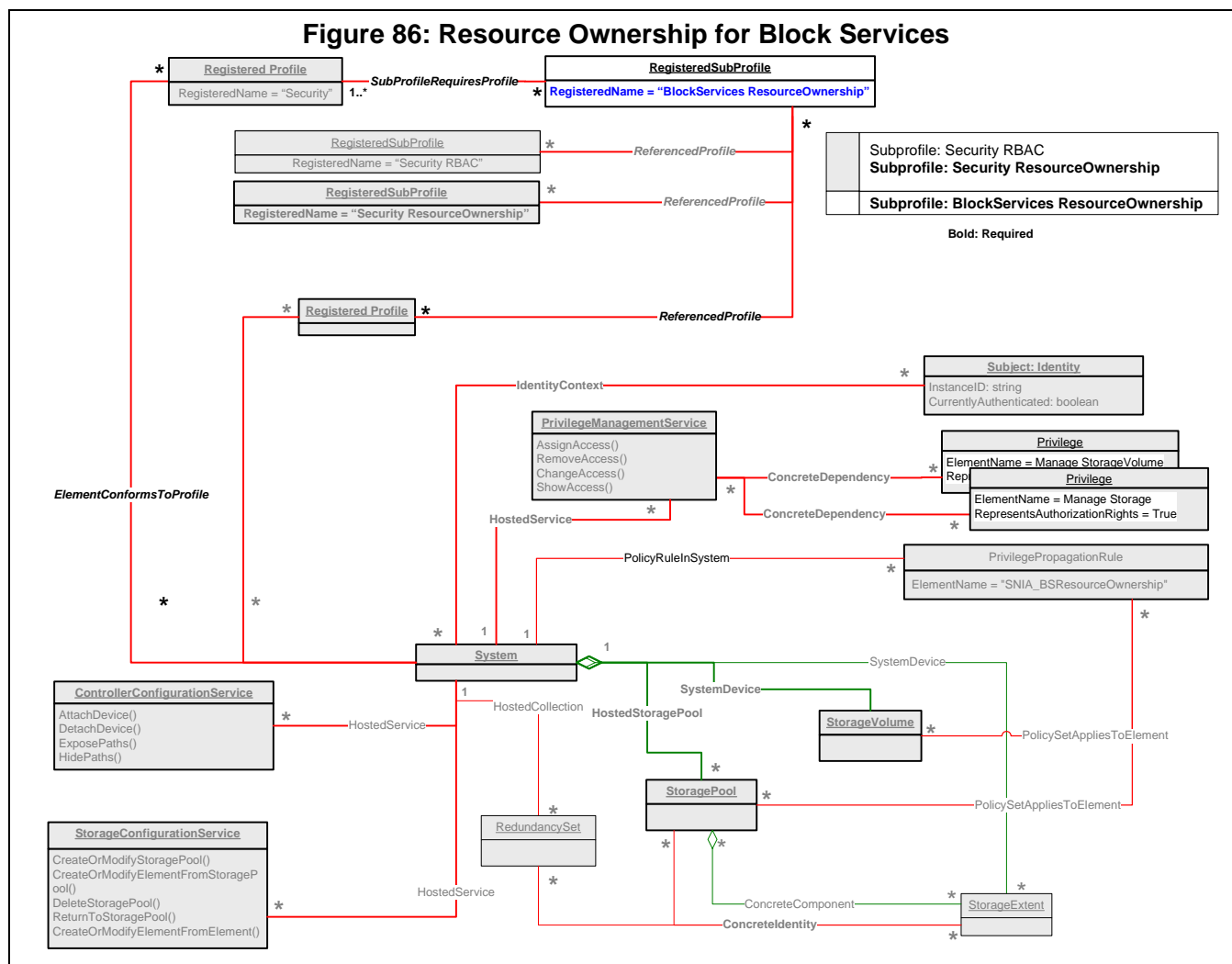
Properties	Flags	Requirement	Description & Notes
SystemCreationClass sName		Mandatory	
CreationClassName		Mandatory	
SystemName		Mandatory	
Name		Mandatory	
AssignStorageResourceAffinity()		Optional	Start a job to assign affinity of a StoragePool(s) or StorageVolume(s) to a storage processor. At the conclusion of the operation, the resource will be a member of the StorageResourceLoadGroup with the primary affinity for the specified storage processor. Support for this method is indicated by the presence of an instance of StorageServerAsymmetryCapabilities in which the property StorageResourceAffinityAssignable is 'true'. If 0 is returned, the function completed successfully and no ConcreteJob instance was required. If 4096/0x1000 is returned, a job will be started to assign the element. The Job's reference will be returned in the output parameter Job.

EXPERIMENTAL

EXPERIMENTAL

Clause 22: Resource Ownership Subprofile

22.1 Description



The Block Services Resource Ownership common subprofile¹ models control over the rights of a client to grant or deny access to block storage resources. By asserting exclusive control over these rights, one client can control which other clients may access those resources. This subprofile is intended for environments in which multiple CIM clients may not be completely aware of each other's activities, making it important that use of the resource not be disrupted by a client that is unaware of shared resource use. Specific examples include use of a volume by in-band virtualizers and NAS gateways, where attempts to manage the volume by clients not associated with this use could be seriously disruptive. An intended configuration is that a CIM client exists in the cascading device that has exclusive use of the volume. The Resource Ownership subprofile is optional.

1. The CIM Resource Ownership subprofile was formerly known as Ownership. It has been renamed to avoid confusion with the notion of file owner commonly found in filesystems.

This profile concerns itself with the existence and use of two sets of rights which may be realized as two Privilege instances that are associated via ConcreteDependency to a PrivilegeManagementService. There is one Privilege to "Manage StorageVolume" and a superset of that to "Manage Storage". Each is described in Table 307.

Table 307: Block Service Management Rights

ElementName	Property	Index	Value
Manage StorageVolume	Activities	0	Execute
	QualifiersFormats	0	<class>.method
	ActivityQualifiers	0	StorageConfigurationService. ReturnToStoragePool StorageConfigurationService. CreateorModifyElementfromElements StorageConfigurationService.AttachDevice, StorageConfigurationService.DetachDevice, StorageConfigurationService.ExposePaths, StorageConfigurationService.HidePaths, PrivilegeManagementService.AssignAccess, PrivilegeManagementService.ChangeAccess, ModifyInstance, SetProperty
Manage Storage	Activities	0	Execute
	QualifiersFormats	0	<class>.method
	ActivityQualifiers	0	StorageConfigurationService. CreateOrModifyStoragePool, StorageConfigurationService. CreateOrModifyElementFromStoragePool, StorageConfigurationService. DeleteStoragePool, StorageConfigurationService. ReturnToStoragePool, StorageConfigurationService. CreateorModifyElementfromElements, ControllerConfigurationService.AttachDevice, ControllerConfigurationService.DetachDevice, ControllerConfigurationService.ExposePaths, ControllerConfigurationService.HidePaths, PrivilegeManagementService.AssignAccess, PrivilegeManagementService.ChangeAccess, ModifyInstance, SetProperty

This profile assumes that the intrinsic CreateInstance and DeleteInstance methods are not supported for either StorageVolumes or StoragePools.

With RepresentsAuthorizationRights set to True, the ChangeAccess call may be used to assign "Manage StorageVolume" rights to a StorageVolume for a particular set of subjects, each represented by an Identity. Once this assignment is made, only members of that set of subjects are permitted to assign "Manage StorageVolume" rights to other subjects, (regardless of the setting of RepresentsAuthorizationRights. The ShowAccess call may be used to list the rights granted to a particular subject Identity and target StorageVolume or StoragePool.

To establish an “Owner” in the sense meant by this profile, only one subject is assigned the "Manage StorageVolume" privilege with RepresentsAuthorizationRights set both to True and False.

The same strategy is used to assign "Manage Storage" rights to a StoragePool.

Even though the SMI-S 1.1 ExposePaths and HidePaths extrinsics act on StorageVolumes by the LUID string parameter rather than a reference, nevertheless they are governed by authorization rights.

This profile requires that every StorageVolume allocated from a StoragePool that is governed by "Manage Storage" rights be assigned the corresponding "Manage StorageVolume" rights to the same subject. This is an implicit PrivilegePropagationRule, which need not be made explicit to be in affect. Whenever ChangeAccess, or other means, is used to modify the “Manage StorageVolume” rights of a particular subject to a StoragePool, those rights are propagated for that subject to all StorageVolumes that have an AllocatedFromStoragePool association to that StoragePool.

If an explicit PrivilegePropagationRule is used, it shall have ElementName set to “SNIA_BSResourceOwnership”.

Optionally, a QueryCondition, (not shown), may be associated to that PrivilegePropagationRule via PolicyConditionInPolicyRule, (not shown), if specified the QueryCondition instance shall have its QueryLanguage property set to “2”, meaning “CQL”, its QueryResultName set to “SNIA_BSResourceOwnershipCondition” and its Query property set to

```
“SELECT (M.SourceInstanceHost || '/' || M.SourceInstanceModelPath) AS PMSPath,
        M.MethodParameters.Subject,
        M.MethodParameters.Target,
        FROM CIM_InstMethodCall M,
        WHERE M.MethodName = 'ChangeAccess'
        AND    M.ReturnValue = 0
        AND    M.PreCall = FALSE
        AND    M.MethodParameters.Target ISA CIM_StoragePool
        AND    ANY P IN M.MethodParameters.Privileges[*]
        SATISFIES (P.ElementName = 'ManageStorage')
```

Additionally, if this optional QueryCondition is associated then a corresponding MethodAction instance, (not shown), shall also be associated to the same PrivilegePropagationRule via PolicyActionInPolicyRule, (not shown). The MethodAction instance shall have its QueryLanguage property set to “2”, meaning “CQL”, its InstMethodCallName set to “SNIA_BSResourceOwnershipAction” and its Query property set to

```
“SELECT (BS.PMSPath || '.' || 'ChangeAccess') AS MethodName,
        BS.Subject AS Subject,
        ObjectPath(SV) AS Target,
        NULL AS PropagationPolicies,
        BS.Privileges AS Privileges
        FROM SNIA_BSResourceOwnershipCondition BS,
             CIM_AllocatedFromStoragePool AFSP,
             CIM_StorageVolume SV,
             CIM_Privilege P
        WHERE ObjectPath(SV) = AFSP.Dependent
        AND    BS.Target = AFSP.Antecedent
        AND    P.ElementName = 'Manage StorageVolume'
```

If AuthorizedSubject/AuthorizedTarget associations are implemented, then these need to be created as appropriate to reflect the assigned rights. In any case, a client may use ShowAccess to determine what privileges are in force for particular subject Identity, StorageVolume or StoragePool.

If the `ChangeAccess` request to establish ownership is not permitted, then the return status shall be `CIM_ERR_ACCESS_DENIED`. This result may be because the requestor is not permitted to make the call, or the requestor does not have sufficient rights to modify ownership of the target.

Some vendors may define additional vendor-specific extrinsic operations that need to be restricted in order to realize the functionality of Resource Ownership. Execution of each such vendor-specific extrinsics shall be added to the above list of restricted activities. Clients may check for the presence of at least the above list of restricted activities, but shall not check for an exact match to the above list, as such a check may fail if there are vendor-specific extrinsics that are also restricted.

22.1.1 Design considerations

This list realizes a number of design decisions:

- For simplicity, the "Manage Storage" Privilege is a superset of the "Manage StorageVolume" Privilege. The "Manage Storage" Privilege may be used against both `StorageVolumes` and `StoragePools`. When applied to a `StorageVolume`, methods called out in that Privilege that do not affect `StorageVolumes` are simply ignored.
- The capability to own `StoragePools` is signaled by a `PrivilegeManagementService` with a `ConcreteDependency.Dependent` "Manage Storage" Privilege with `RepresentsAuthorizationRights` set to `True`.
- The "Manage StorageVolume" Privilege does not provide the ability to manage `StoragePools`.
- `DeleteProtocolController` is not restricted. The design goal is to control resource management in a fashion that keeps reasonably well-behaved clients from causing unintended problems. Control of the `StoragePool` and `StorageVolume` instances is sufficient, as a reasonably well-behaved client should at least call `DetachDevice` or `HidePaths` on the associated `StorageVolumes` before calling `DeleteProtocolController` (both `DeleteDevice` and `HidePaths` are controlled), or at the very least understand what the attached volumes are being used for before deleting the protocol controller. The `ProtocolControllerforPort` and the associated port (e.g., `FCPort`) are also not restricted for similar reasons.
- `RemoveAccess` and `ChangeAccess` are not restricted to avoid complexity. These could be restricted by creating a second type of resource ownership Privilege to control them, and the corresponding access Privileges to enforce the restrictions, but for 1.1, it seems reasonable to trust clients that don't know what they're doing to avoid invocations of `RemoveAccess` and `ChangeAccess`.
- `ServiceAffectsElement` associations are assumed between `Services` and affected elements. (See Figure 87.) This subprofile does not REQUIRE an implementation to present these associations unless there is more than one of a particular type `Service` in the profiled Namespace.
- `AuthorizedPrivilege` instances are assumed when a Privilege is granted to a subject or assigned to a target. (See Figure 88.) `AuthorizedTarget` and `AuthorizedSubject` associations are assumed between the `AuthorizedPrivilege` and the target and subject entities respectively. This subprofile does not REQUIRE the

implementation to make these instances explicit. Instead this profile relies on the ChangeAccess method to grant or deny rights and on the ShowAccess method to display rights.

Figure 87: ServiceAffectsElement Associations for ResourceOwnership

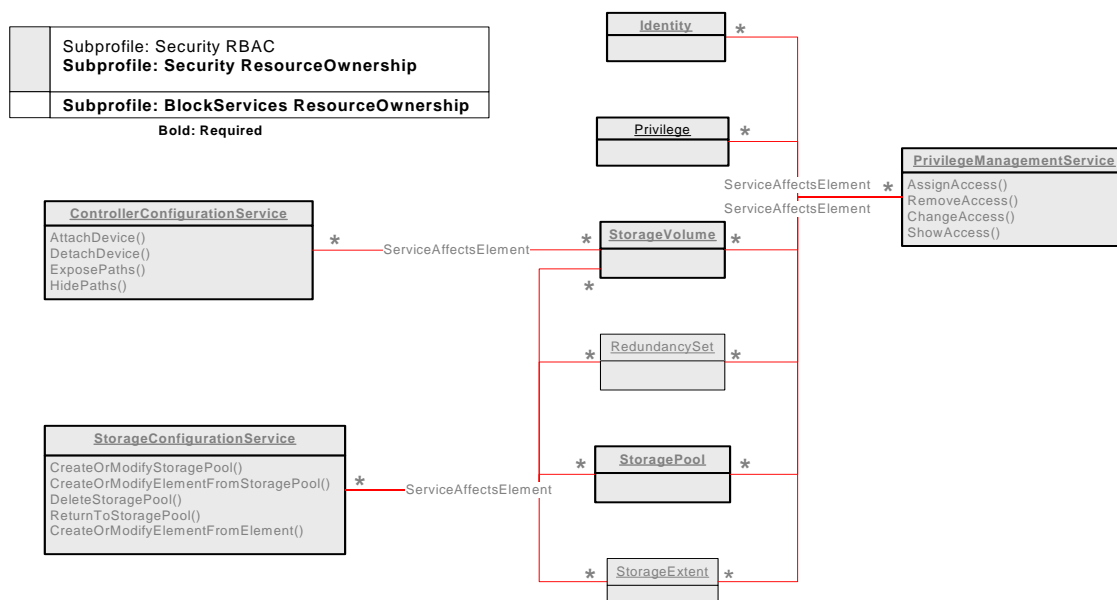
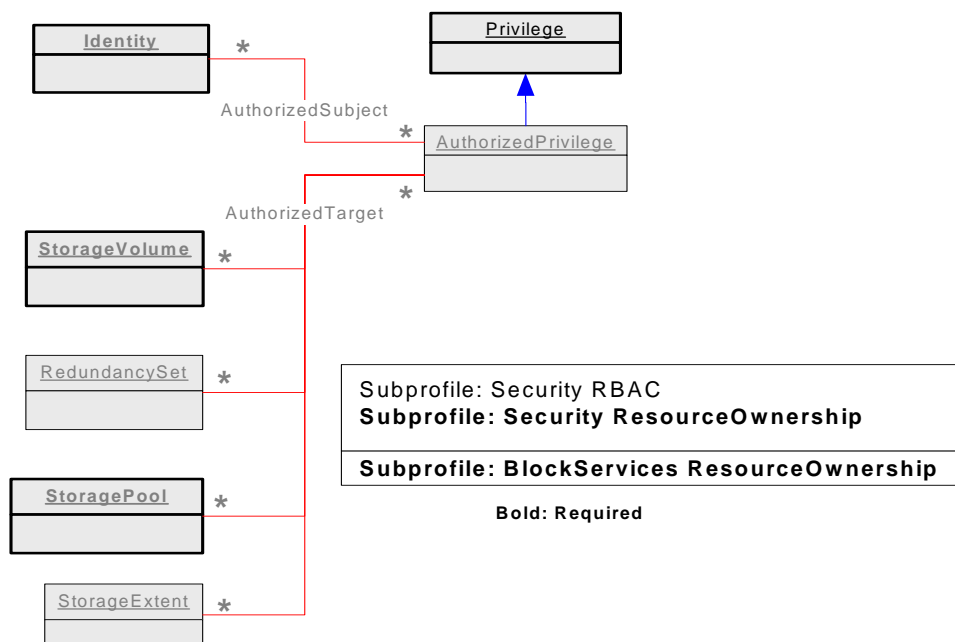


Figure 88: AuthorizedPrivilege Associations for ResourceOwnership



- PrivilegePropagationRule instances are assumed with appropriate PolicySetAppliesToElement associations to StoragePool and StorageVolume instances and a PolicyRuleInSystem association to a System instances. This subprofile does not REQUIRE either the PrivilegePropagationRule instances nor the related association instances.

22.1.2 Privilege Propagation

Propagation is a means of restricting the number of AuthorizedTarget associations for a Privilege. Propagation has two elements:

- 1) Privilege restrictions on a StoragePool propagate to ConcreteComponent.PartComponent StorageExtents.
- 2) Privilege restrictions on a StoragePool propagate across ConcretelIdentity to a StorageExtent aspect. (For instance when a Raid 5 extent is used as a StoragePool.)
- 3) Privilege restrictions on a StorageExtent propagate across ConcretelIdentity to a RedundancySet aspect. (For instance when spares are available for a Raid 5 extent.)

To place these rules in force, a PrivilegePropagationRule instance is associated via PolicySetAppliesToElement to affected StoragePools or StorageVolumes. This rule shall have its ElementName set to "BlockServices ResourceOwnership" and it shall not have any PolicyCondition or PolicyAction instances associated with it.

ShowAccess may be used to determine the resulting behavior.

22.2 Client Considerations and Recipes

Resource Ownership Privileges can be distinguished from LUN Mapping/Masking privileges as the latter contain Execute (instance of Activities[]) cdb=* (ActivityQualifiers[]) SCSI Command (QualifierFormats[]).

A cascading provider determines whether or not Resource Ownership is supported by an array by looking for Block Services Resource Ownership as a RegisteredSubprofile of the Array profile.

While this subprofile is intended to support cascading, it can be used with any CIM Client that can authenticate to the CIM Server and thereby obtain an authenticated Identity.

A client can determine whether resource ownership restrictions are enforced on a StorageVolume or StoragePool by using the ShowAccess method (preferred) or by association traversal via AuthorizedTarget to resource ownership Privileges.

When CIM Servers are cascaded, it's necessary to be able to associate the embedded CIM client (e.g., in a virtualizer or NAS head) with the Identity in the array that is the AuthorizedSubject of the privileges. Assuming shared secrets, this can be modeled and realized as follows:

- In the virtualizer or NAS Head, a CIM Service instance is associated (ServiceSAPDependency) with a RemoteServiceAccessPoint that has associated via CredentialContext to a SharedSecret credential that contains information necessary for authentication.
 - RemoteID: String by which the principal is known. This maps to Account.UserID
 - Secret: Password or other secret. This is set, but is not typically readable.
- In the array, the Identity instance is created that is authenticated by the Credential in the previous step. This Identity may be associated via ConcretelIdentity to an Account. Or, it may be associated via IdentityContext to a RemoteServiceAccessPoint that provides access to a 3rd Party Authentication service. If the latter, then the Security 3rdPartyAuthentication subprofile shall be present on the Array.
- When the CIM client uses HTTP authentication with that username and password, the authenticated Identity is assigned to the CIM client's session.

There is no requirement that the Identity and Account instances in the array be creatable or manipulable via CIM. The contents of these instances have significant security implications and hence the ability to create and change

them need to be carefully controlled. This example uses HTTP authentication, but is not meant to exclude other forms of authentication.

EXPERIMENTAL

IMPLEMENTED

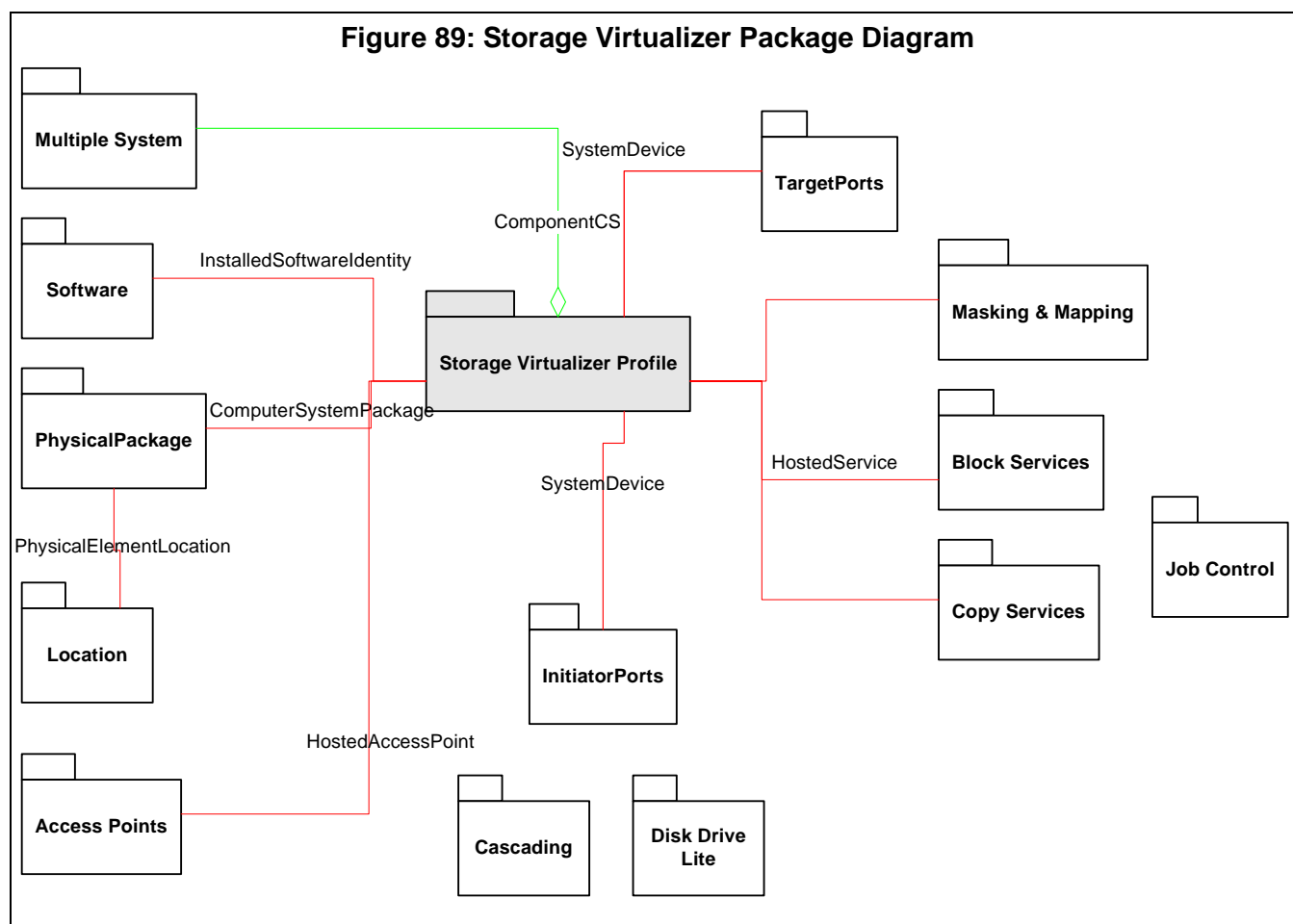
Clause 23: Storage Virtualizer Profile

23.1 Description

Storage virtualizers act like RAID arrays but can use storage provided by systems external to the storage virtualizer and local disks. A storage virtualizer system combines both remote and local storage to create a seamless pool. The virtualization system allocates volumes from the pool for host systems to use.

The basic virtualizer system profile provides a read-only view of the system. The various subprofiles indicated in Figure 89 extend this description and also enable configuration. Refer to 23.4 for more information on these optional extensions. This profile also includes the mandatory Clause 33: Physical Package Package (in *Storage Management Technical Specification, Part 2 Common Profiles*) that describes the physical layout of the system and includes product identification information. The modeling in this document is split into various sections that describe how to model particular elements of an storage virtualizer system.

Figure 89 illustrates the relationship between the packages related to the Storage Virtualizer Profile.

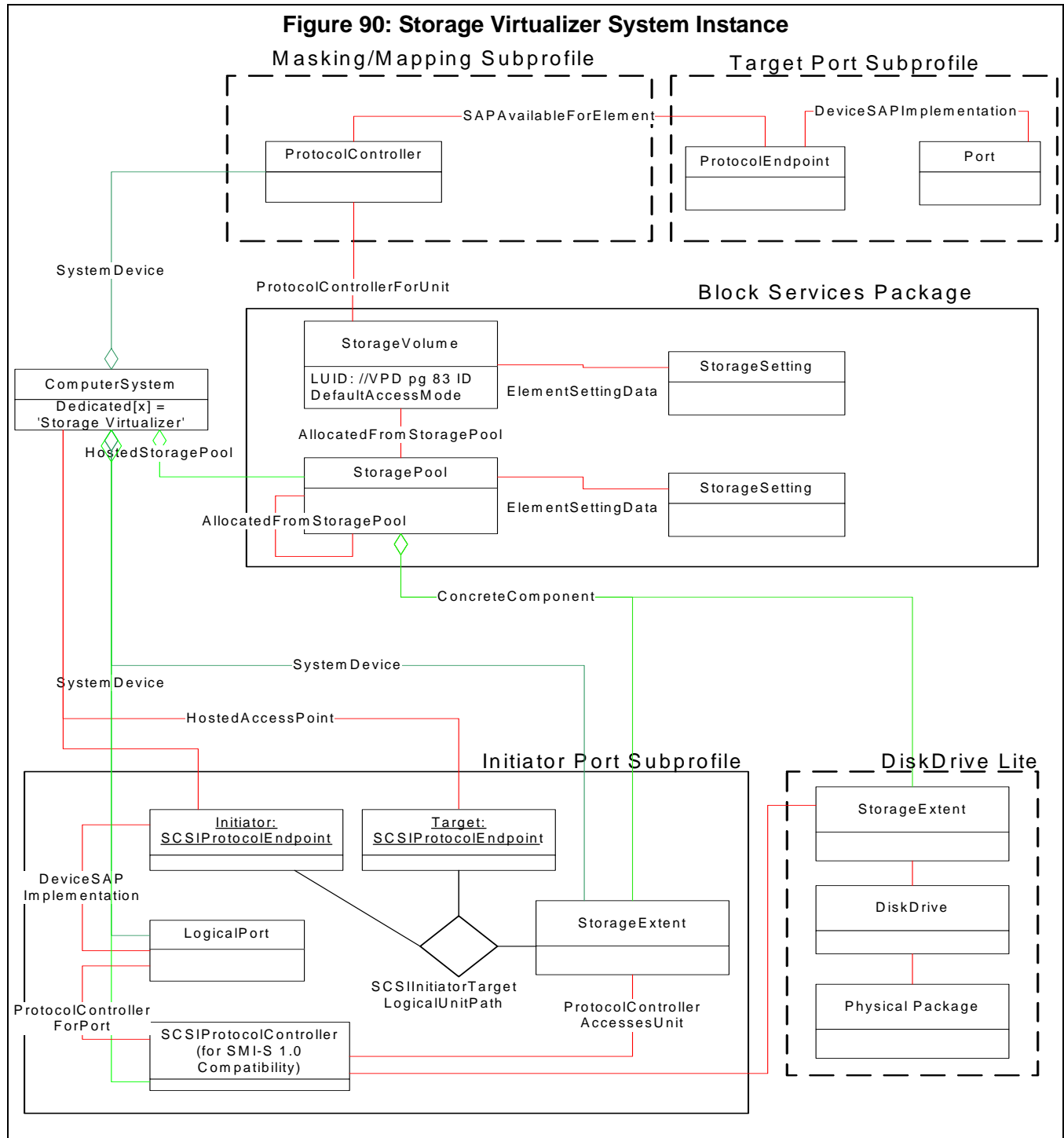


23.1.1 Instance Diagram

The diagrams used in this document are 'Instance' diagrams implying the actual classes that you implement rather than the class hierarchy diagrams often used to show CIM models. This is felt to be easier to understand. Please

refer to the DMTF MOF files for information on class inheritance information and full information on the properties and methods used.

Figure 90 is an instance diagram of a simple Storage Virtualization system.



23.1.2 Storage Virtualization System

The Virtualization system is modeled using the ComputerSystem class with the “Dedicated” properties set to ‘BlockServer’ and “StorageVirtualizer”. The model allows the system to be a cluster or contain redundant components, but the components act as a single system. The ComputerSystem class and common Multiple Computer System Subprofile model this.

The StoragePool classes in the center of the diagram represents the mapping from array storage to volumes for host access. The pool is hosted on the ComputerSystem and services to control it are host on the same controller. The StorageExtent at the bottom of the screen represents the storage from external arrays used by the mapping. These StorageExtents are connected to the pool using the ConcreteComponent association. The SCSIProtocolController with the ProtocolControllerAccessesUnit association to the StorageExtent are provided for clients convenience (and compatibility with SMI-S 1.0).

StorageVolumes at the upper right are the volumes created from the StoragePool and are accessible from hosts. The associations to the SCSIProtocolController and to the Port indicate ports the volume is mapped to. The StorageVolumes are described by the StorageSetting class connected by the ElementSettingData association.

23.1.3 Disk Drive Lite

The disk drive Lite subprofile is optional. It should be used to model storage local to the storage virtualizer system. The Disk drive lite model includes a StorageExtent instance that represents the storage of the disk drive. If the disk drive lite profile is implemented, the StorageExtent shall be associated to a primordial pool. It may share a primordial pool with external storage or it can have its own primordial pool.

23.1.4 Controller Software

Information on the installed controller software is represented by the optional Software subprofile. This is linked to the controller using an InstalledSoftwareIdentity association.

23.1.5 Device Management Access

Most devices now have a web GUI to allow device specific configuration. This is modeled using the common subprofile “Access Point”.

23.1.6 Physical Modeling

The physical aspects of the storage virtualizer ComputerSystem are represented by the Common Package “Physical Package” and the optional subprofile “Location”. See these common sections for more details.

23.1.7 Services

The system hosts services used to control the configuration of the system’s resources. These services are optional and modeled by the “Block Services” Package, and the “Copy Services”, and “Job Control” subprofiles.

23.1.8 Ports

An implementation of the storage virtualizer shall implement at least one Target Ports Subprofile and may implement one or more of the Initiator Ports Subprofiles. However, SMI-S does not specify any particular port type be supported. In either target or initiator cases, the ports could be FC or iSCSI.

The storage virtualizer ConcreteComponent StorageExtent instances shown in the Initiator Ports profile are the optional remote LogicalDevice instances from Initiator Ports. However, these StorageExtents are mandatory in the Storage Virtualizer profile.

23.2 Health and Fault Management

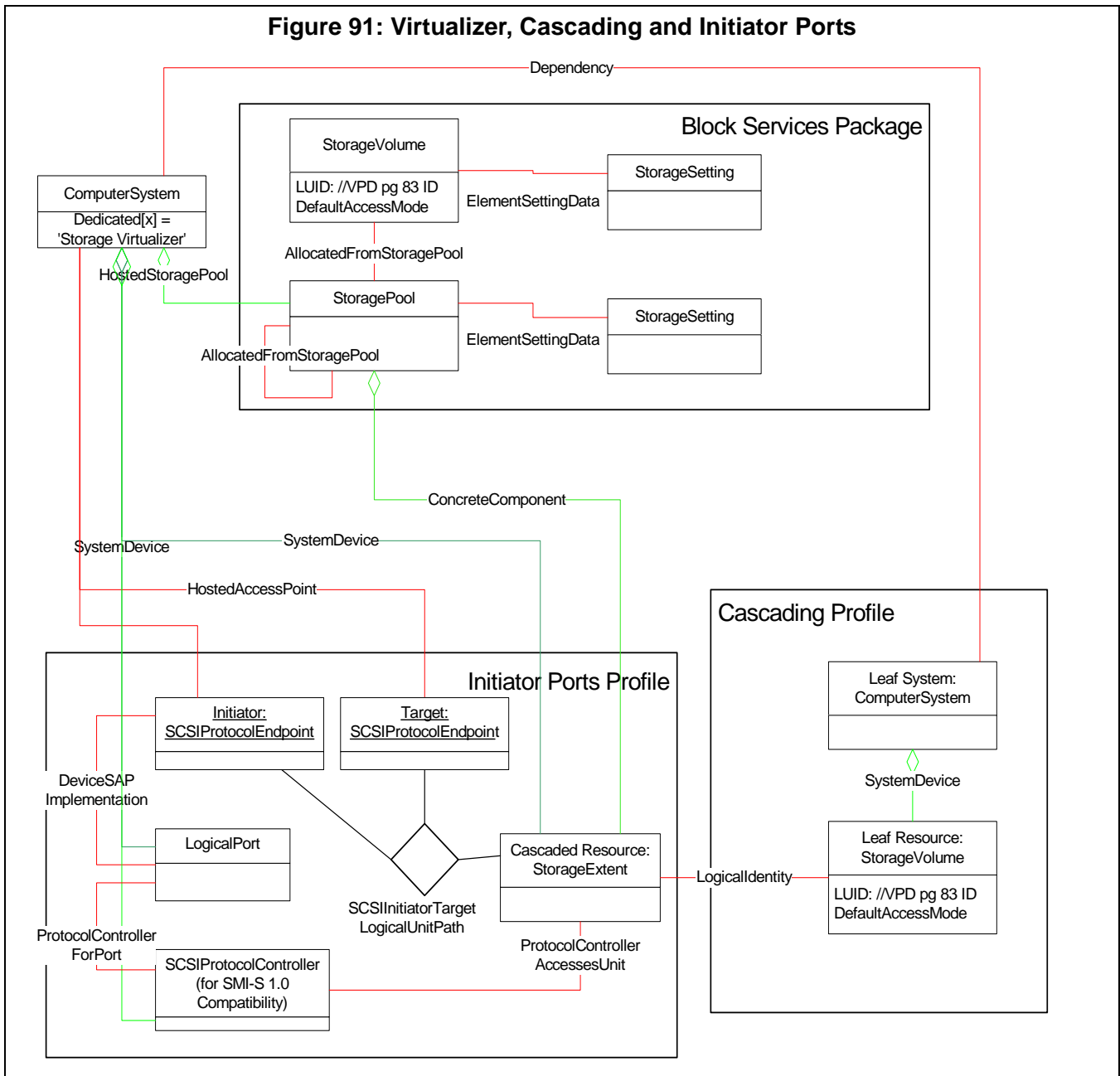
Defined in the included subprofiles.

23.3 Cascading Considerations

Figure 91 shows the relationship between the Storage Virtualizer and Cascading profiles. Cascading is required when the virtualizer imports logical units from arrays. See Clause 26: Cascading Subprofile in *Storage Management Technical Specification, Part 2 Common Profiles*.

Each imported array is modeled in the virtualizer following with a ComputerSystem; the arrays' logical units are modeled using StorageVolume instances. These are depicted in Figure 91 in the box labeled "Cascading". The Cascading profile refers to these as the leaf system and leaf resources.

Each leaf ComputerSystem (representing an array) is associated to the Virtualizer ComputerSystem using a Dependency association. StorageVolume models an Array logical unit and is associated to storage virtualizer ConcreteComponent StorageExtent via the LogicalIdentity association. The StorageExtent represents the virtualizer's view of logical units imported from arrays. The StorageExtents are local resources.(as described in the Cascading Profile). The leaf ComputerSystem and StorageVolume are documented in the Cascading profile and contain the correlatable IDs needed to map virtualizer resources to equivalent objects in an Array profile.

Figure 91: Virtualizer, Cascading and Initiator Ports

The Dependency and LogicalIdentity associations and the leaf ComputerSystem and leaf StorageVolume classes are documented in the Cascading profile.

23.4 Supported Subprofiles and Packages

Table 308 lists the supported subprofiles for this Profile.

Table 308: Supported Profiles for Storage Virtualizer

Registered Profile Names	Mandatory	Version
Access Points	No	1.2.0
Block Server Performance	No	1.2.0
Block Storage Views	No	1.2.0
CKD Block Services	No	1.2.0
Erasure	No	1.2.0
Storage Server Asymmetry	No	1.2.0
Volume Composition	No	1.2.0
Storage Element Protection	No	1.2.0
Generic Target Ports	Yes	1.2.0
Copy Services	No	1.2.0
Job Control	No	1.2.0
Location	No	1.2.0
Masking and Mapping	No	1.2.0
Software	No	1.2.0
Multiple Computer System	No	1.2.0
Generic Initiator Ports	No	1.2.0
FC Initiator Ports	No	1.2.0
iSCSI Target Ports	No	1.2.0
FC Target Ports	No	1.2.0
iSCSI Initiator Ports	No	1.2.0
Extent Composition	No	1.2.0
Cascading	Yes	1.2.0
Indications	Yes	1.2.0
Block Services	Yes	1.2.0
Physical Package	Yes	1.2.0

23.5 Methods of the Profile

None.

23.6 Client Considerations and Recipes

None.

23.7 Registered Name and Version

Storage Virtualizer version 1.2.0

23.8 CIM Elements

Table 309: CIM Elements for Storage Virtualizer

Element Name	Requirement	Description
CIM_ComputerSystem (23.8.1)	Mandatory	'Top-level' system that represents the whole virtualizer.
CIM_ConcreteComponent (23.8.2)	Mandatory	Used to associate StorageExtents that are playing the Pool Component role to a Primordial StoragePool.
CIM_StorageExtent (23.8.3)	Mandatory	Used to represent the storage imported from external arrays and used as ConcreteComponents of Primordial StoragePools.
CIM_LogicalPort (Target Port) (23.8.4)	Mandatory	Target (front end) ports for the array. Most requirements are defined in the referenced initiator port subprofiles. In this profile, this class represents a requirement for some target port - regardless of the transport.
CIM_LogicalPort (Initiator Port) (23.8.5)	Optional	Initiator (back end) ports for the array. Most requirements are defined in the referenced initiator port subprofiles. In this profile, this class represents a requirement for some initiator port - regardless of the transport.
CIM_SystemDevice (System to StorageExtent) (23.8.6)	Mandatory	This association links ConcreteComponent StorageExtent to the scoping system.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Mandatory	Creation of a ComputerSystem instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Mandatory	Deletion of a ComputerSystem instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Optional	Deprecated WQL - Modification of OperationalStatus of a Storage Volume instance, provided for backward compatibility with In-band Virtualization.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.CIM_StorageVolume::OperationalStatus <> PreviousInstance.CIM_StorageVolume::OperationalStatus	Optional	Experimental CQL - Modification of OperationalStatus of a Storage Volume instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Optional	Deprecated WQL - Modification of OperationalStatus of a FC port instance, provided for backward compatibility with In-band Virtualization.

Table 309: CIM Elements for Storage Virtualizer

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalPort AND SourceInstance.CIM_LogicalPort::OperationalStatus <> PreviousInstance.CIM_LogicalPort::OperationalStatus	Optional	Experimental CQL - Modification of OperationalStatus of a Logical (FC or Ethernet) port instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Optional	Deprecated WQL - Modification of OperationalStatus of a ComputerSystem instance, provided for backward compatibility with In-band Virtualization.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	Optional	Experimental CQL - Modification of OperationalStatus of a ComputerSystem instance

23.8.1 CIM_ComputerSystem

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 310 describes class CIM_ComputerSystem.

Table 310: SMI Referenced Properties/Methods for CIM_ComputerSystem

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
Name	C	Mandatory	Unique identifier for the storage virtualizer. Eg IP address or a FC WWN.
ElementName		Mandatory	User friendly name
OtherIdentifyingInfo	C	Mandatory	
IdentifyingDescriptions	C	Mandatory	
OperationalStatus		Mandatory	Overall status of the storage virtualizer

Table 310: SMI Referenced Properties/Methods for CIM_ComputerSystem

Properties	Flags	Requirement	Description & Notes
NameFormat		Mandatory	Format for Name property.
Dedicated		Mandatory	Indicates that this computer system is dedicated to operation as a storage virtualizer
PrimaryOwnerContact	M	Optional	Contact a details for owner
PrimaryOwnerName	M	Optional	Owner of the storage virtualizer

23.8.2 CIM_ConcreteComponent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 311 describes class CIM_ConcreteComponent.

Table 311: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Properties	Flags	Requirement	Description & Notes
GroupComponent		Mandatory	A Primordial StoragePool
PartComponent		Mandatory	The StorageExtent

23.8.3 CIM_StorageExtent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 312 describes class CIM_StorageExtent.

Table 312: SMI Referenced Properties/Methods for CIM_StorageExtent

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	

Table 312: SMI Referenced Properties/Methods for CIM_StorageExtent

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
DeviceID		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

23.8.4 CIM_LogicalPort (Target Port)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 313 describes class CIM_LogicalPort (Target Port).

Table 313: SMI Referenced Properties/Methods for CIM_LogicalPort (Target Port)

Properties	Flags	Requirement	Description & Notes
UsageRestriction		Mandatory	Shall be 2 to indicate this is a front end port only or 4 to indicate usage is not restricted.

23.8.5 CIM_LogicalPort (Initiator Port)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Optional

Table 314 describes class CIM_LogicalPort (Initiator Port).

Table 314: SMI Referenced Properties/Methods for CIM_LogicalPort (Initiator Port)

Properties	Flags	Requirement	Description & Notes
UsageRestriction		Mandatory	Shall be 3 to indicate this is a back end port only or 4 to indicate usage is not restricted.

23.8.6 CIM_SystemDevice (System to StorageExtent)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 315 describes class CIM_SystemDevice (System to StorageExtent).

Table 315: SMI Referenced Properties/Methods for CIM_SystemDevice (System to StorageExtent)

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

IMPLEMENTED

EXPERIMENTAL

Clause 24: Volume Composition Profile

24.1 Description

The Volume Composition Subprofile describes how a implementation would combine exposable storage elements into other storage elements. Storage Elements in this context are Storage Volumes or Logical Disks, although for this version of the specification, only StorageVolumes are supported. This subprofile builds on the models in the Block Services package and the Extent Composition subprofile. Block Services shows how a Storage Extents may be constructed and ultimately exposed as a storage element. Extent Composition shows the relationships between volumes and extents. This subprofile takes these models to the next logical step and shows how storage elements may be further combined into composite storage elements or storage pools.

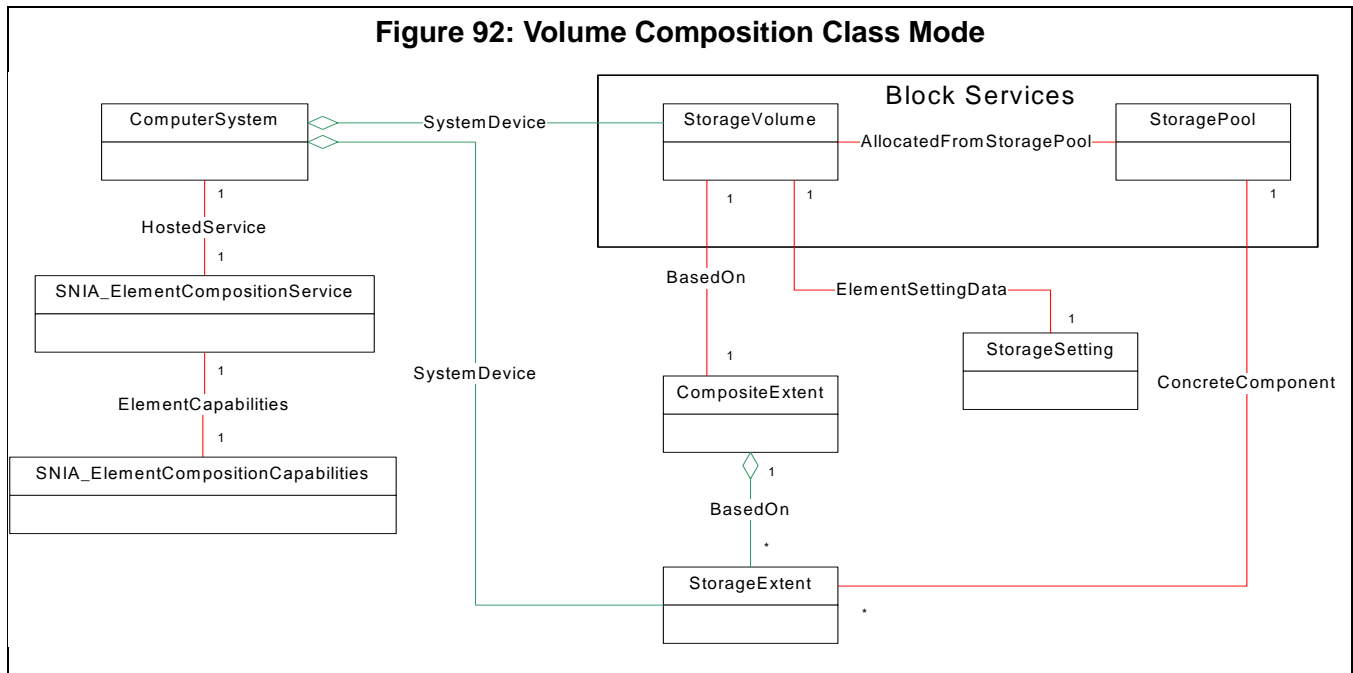
Some Arrays and Storage Virtualizers as well as Volume Managers have the ability to create composite volumes by combining together existing storage volumes to make them appear to be one, bigger, volume. This is different from the Extent Composition model because these are volumes that are already allocated from the Storage Pool and exposed. They may not necessarily be mapped to a port or masked to a host. These volumes can come from the same or different storage pools. Often the rules to determine which volumes can be combined with other volumes are quite complex and can vary even across a vendor's own product line. Once these elements are combined together, only one storage element is visible and the rest of the storage elements are hidden and cannot be exposed. When the composite storage element is dissolved, the hidden StorageVolumes reappear.

This subprofile introduces a number of new methods and capabilities. The existing methods in StoragePool and StorageConfigurationService were found to be inadequate or attached to the wrong class (i.e., StoragePool) to support the desired functionality. It was decided that it would be better to introduce new methods with specific focus, instead of extending or overloading the usage of existing methods.

24.1.1 Model

To model these composite volumes, this subprofile defines the use of CompositeExtent to represent the "composition" characteristics of the volume. A composite StorageVolume shall have a BasedOn association to the Antecedent CompositeExtent. That CompositeExtent shall have CompositeExtentBasedOn or BasedOn relationships to the underlying extents (in potentially multiple pools) that comprise the StorageVolume. These underlying extents could, in turn, be CompositeExtents. In the case where a volume is not a composite, it shall have a BasedOn relationship to an underlying StorageExtent. This may, in some implementations, be a CompositeStorageExtent.

If the volume is a composite from multiple pools, there shall be one AllocatedFromStoragePool association to each pool. SpaceConsumed shall show applicable space consumed from each pool. The general class model looks like Figure 92.



The important thing to note about the class model is that the CompositeExtent is not associated via ConcreteComponent to the StoragePool.

This version of the profile uses two SNIA classes, SNIA_ElementCompositionService and SNIA_ElementCompositionCapabilities. These classes will be replaced with the corresponding CIM classes once the appropriate change requests have gone through the DMTF.

The client can use the SNIA_ElementCompositionCapabilities to determine which features of this profile are supported. The first property to check is SupportsComposites, which will be set to true if the instrumentation supports creating and modifying composites. The client should also check MaxCompositeSize and MaxCompositeElements to determine the bounds for composite creation. Since there are a number of differences in the way vendors have implemented creation and modification, the client should check the CompositionCharacteristics array to understand which creation and modification options the instrumentation supports. The SupportedAsynchronousActions and SupportedSynchronousActions indicate which methods are supported and whether or not a job is started when the method is invoked. An entry in both arrays indicates a job may be started in some cases but not in others. SupportedStorageElements indicates the types of storage elements that may be used. For this version of the specification, only StorageVolumes are supported. The CompositionMethodsSupported indicates which of the different ways of creating a composite (simple concatenation, striping across elements, concatenate and stripe, etc.) are supported by the instrumentation. Lastly, CompositeSourcesSupported is used to indicate the source of storage elements when they are not explicitly specified in the call to CreateOrModifyCompositeElement. The instrumentation can examine the CompositionCharacteristics property to determine which options are permitted. See Table 316 for a summary of those possible values.

Table 316: CompositionCharacteristics Property

Value	Description
CompositionIsDestructive	Any data that exists on the elements will be destroyed when the composite is created
CanCompositeComposites	It is possible to use an existing composite as an element to a new composite

Table 316: CompositionCharacteristics Property

Value	Description
CanModifyComposite	An existing composite can be modified by adding or removing one or more elements
CompositeElementsMustBeSameSize	All elements used to create/modify a composite shall be the same size
CompositeElementsMustBeSameRAID/QoS	All elements used to create/modify a composite shall have the same RAID or QoS level
DecompositionDeletesElements	When the composite is dissolved, the component elements (e.g. StorageVolumes) are deleted
CompositionIsDestructive	Any data that exists on the elements will be destroyed when the composite is created

QoS Settings

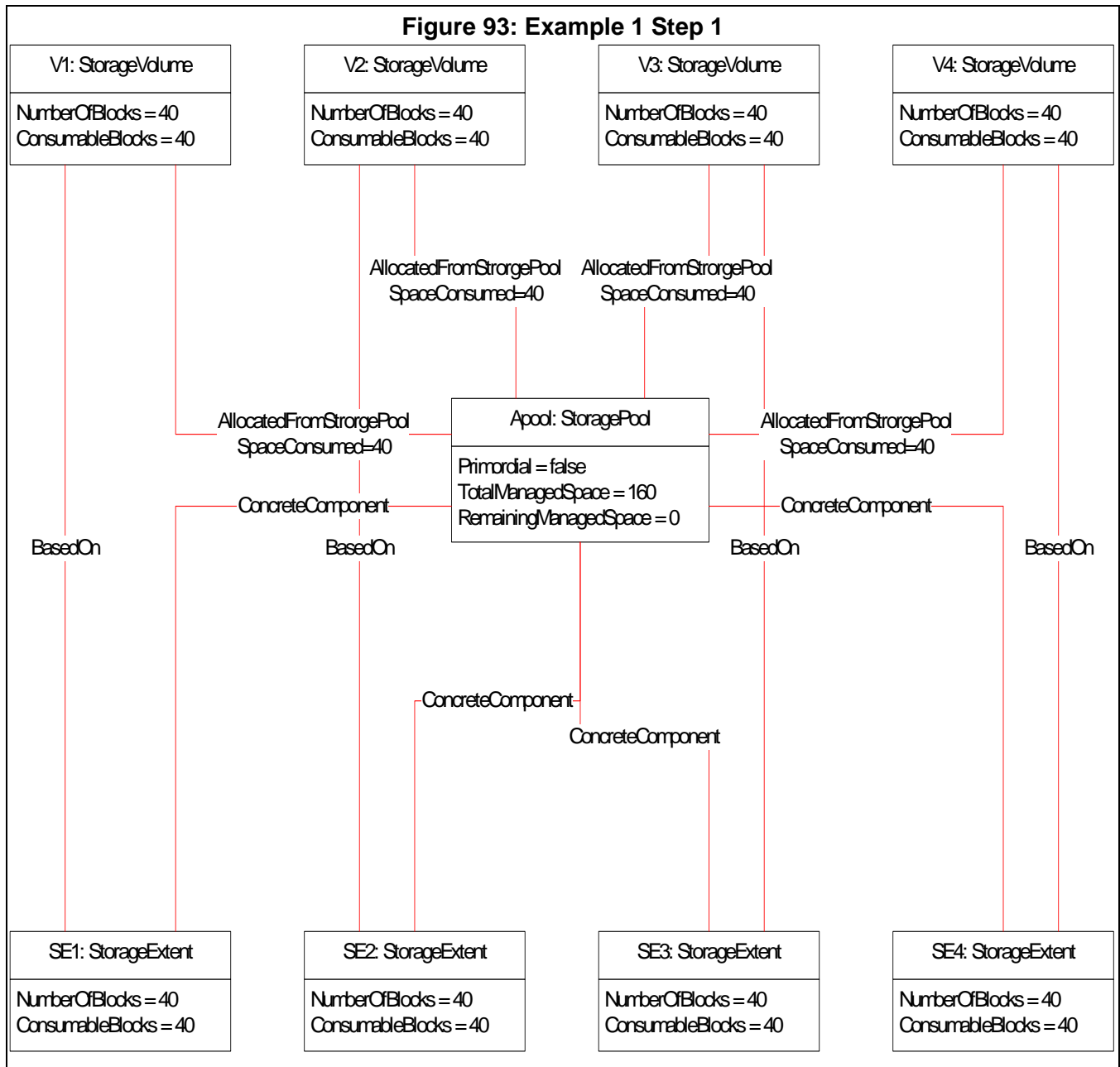
When a composite is created, it will have a StorageSetting associated to it, like regular StorageVolumes or LogicalDisks. Determining what this QoS will be will depend upon the parameters passed in to CreateOrModifyCompositeElement. If only InElements is passed in, the QoS setting shall be the lowest values of the Settings of the InElements. If Goal or RepresentativeElement is passed in, the instrumentation shall attempt to meet that QoS setting instead of InElements (if InElements is non-NULL).

24.1.2 Examples

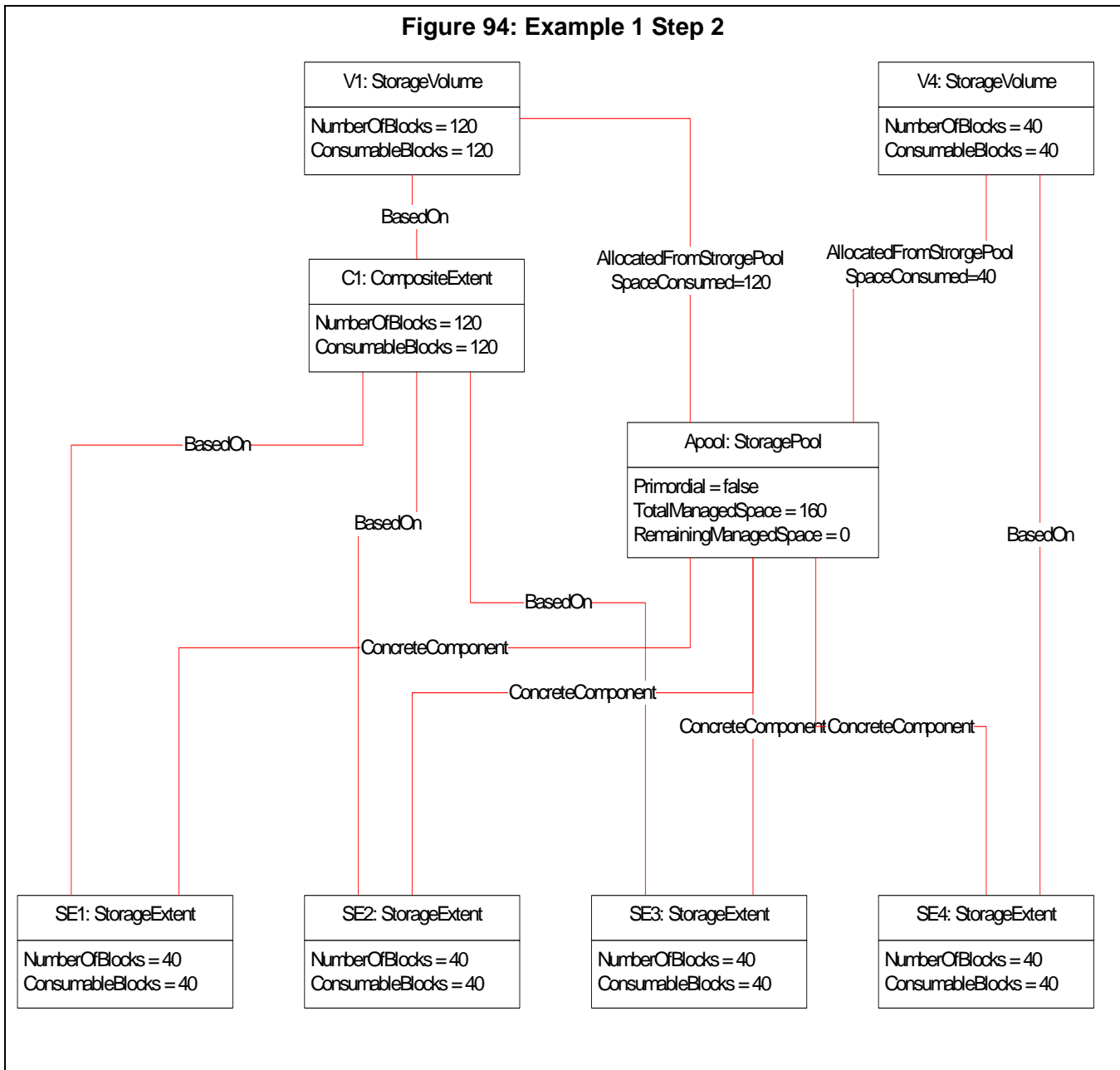
24.1.2.1 Example 1

The following example shows how a composite volume may be created. For simplification, the value of the StorageExtent.BlockSize property is 1 and the associations to the underlying primordial StoragePool have been omitted, along with the StorageSettings associated to the volumes. In some implementations, there may be intermediate extents between the volume and the ConcreteComponent StorageExtent.

In this example, we have four StorageExtents of 40 blocks each that are combined into a concrete storage pool of 160 blocks and four storage volumes allocated from the pool, each consuming 40 blocks. The remaining space in the pool is 0 blocks.

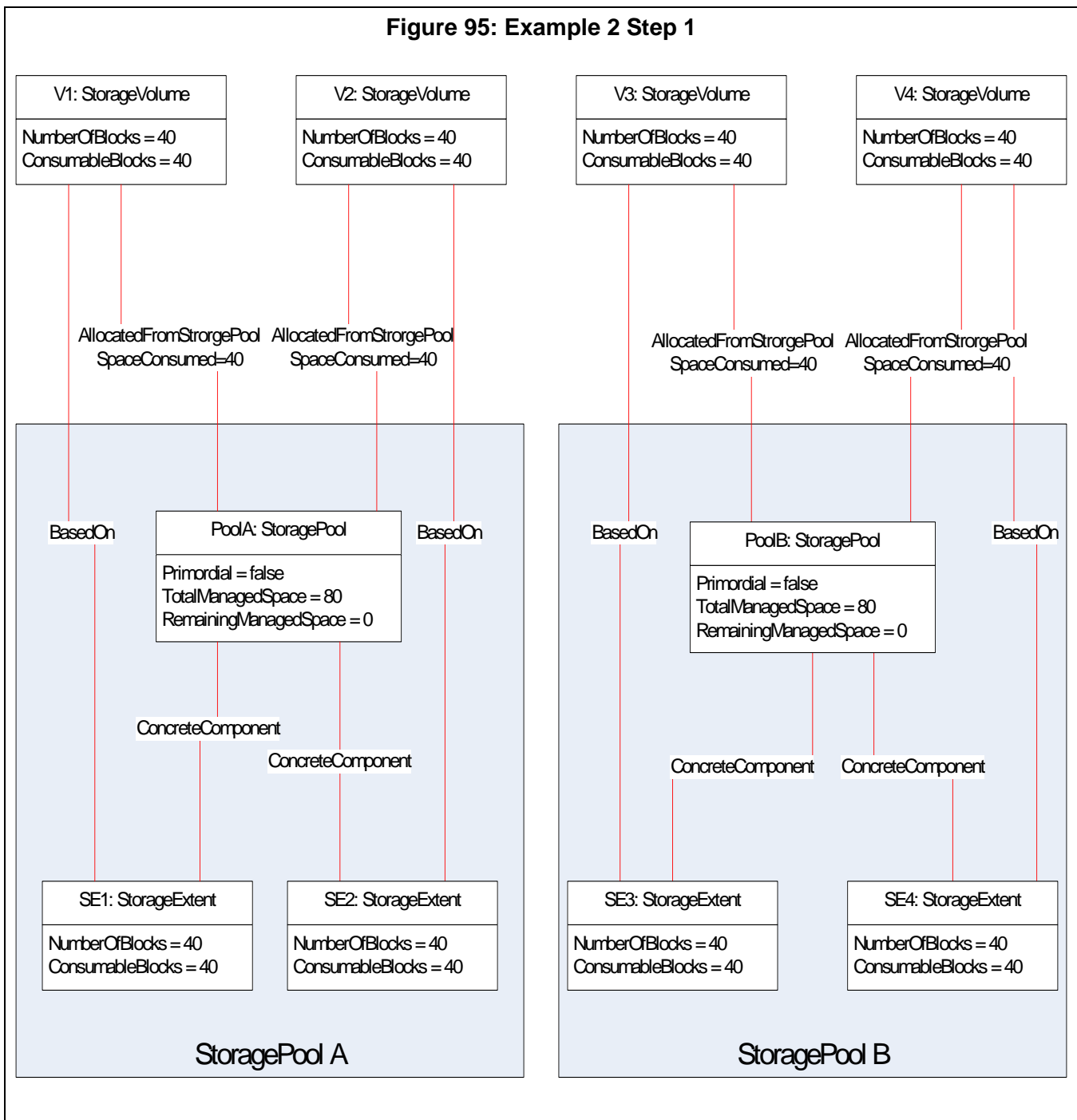


Next, a composite volume is created by calling `CreateOrModifyCompositeElement` using three of the volumes (V1, V2, and V3). The result is the creation of a composite volume with the name V1 whose size is now 120 blocks and volumes V2 and V3 are now inaccessible. The volume V4 is unchanged. A `CompositeExtent` is added and is the Antecedent of a `BasedOn` association to the `StorageVolume`. In turn, the `BasedOn` associations that were going from volumes V1, V2, and V3 from extents SE1, SE2, and SE3 are now associated from the extents to the `CompositeExtent`.

Figure 94: Example 1 Step 2**24.1.2.2 Example 2**

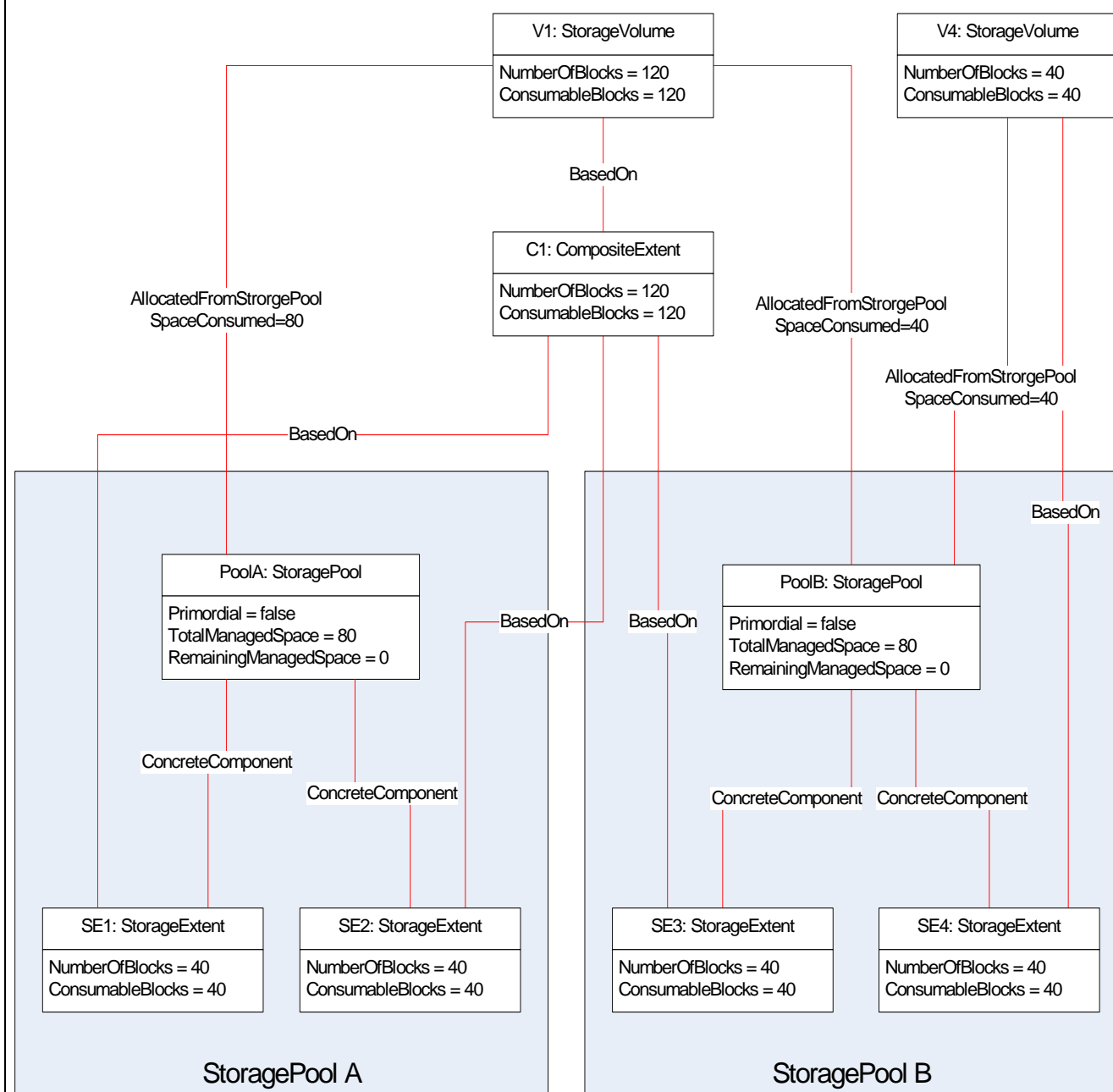
In this example, a composite volume is built from volumes from two concrete storage pools. The configuration is the same as in the first example, except now there are two concrete StoragePools. Volumes V1 and V2 and extents SE1 and SE2 are associated to StoragePool A, and volumes V3 and V4 and extents SE3 and SE4 are associated to StoragePool B.

Figure 95: Example 2 Step 1



Like Example 1, above, three volumes are combined into a composite volume, leaving one original volume. In this case, the composite volume has an `AllocatedFromStoragePool` association to each of the pools from which it was created. The `SpaceConsumed` property in the association is set to the space used from that particular pool. In this case, since two extents were consumed from StoragePool A and one from StoragePool B, the `AllocatedFromStoragePool.SpaceConsumed` for StoragePool A is 80 blocks and the `AllocatedFromStoragePool.SpaceConsumed` for StoragePool B is 40 blocks. The `CompositeExtent` has `BasedOn` associations to the underlying `StorageExtents` in each pool. Figure 96 shows the resulting model.

Figure 96: Example 2 Step 2



24.2 Health and Fault Management Consideration

Not defined in this specification.

24.3 Cascading Considerations

None

24.4 Supported Profiles, Subprofiles, and Packages

Table 317 lists supported profiles for Volume Composition.

Table 317: Supported Profiles for Volume Composition

Registered Profile Names	Mandatory	Version
Extent Composition	Yes	1.1.0
Block Services	Yes	1.1.0

24.5 Methods of the Profile

24.5.1 CreateOrModifyCompositeElement

This method is found in the `SNIA_ElementCompositionService`. It creates or modifies a composite element. Only like elements (e.g. `StorageVolumes`) can be combined. In this version of the specification, only `StorageVolumes` may be used to create composite elements.

As this method is designed to support vendors' sometimes complicated algorithms for creating and modifying composite storage elements, the operation of this method is necessarily complex. The key parameters are the `Goal`, `RepresentativeElement`, `Size`, `InElements[]`, and `TheElement`. Setting one or more of these values will influence what the other values of these key parameters may be. These combinations will be described below. Of the other parameters, they are fairly self-explanatory and are described in Table 318 below. For this version of the specification, `ElementType` shall only be "StorageVolume".

The `Goal` parameter specifies a set of generic QoS settings to use when creating the composite. The `RepresentativeElement` parameter is intended as a more detailed goal or QoS target for the composite. Because vendors have complex rules to create composites, it can be difficult to map those to the standard QoS settings that might be expressed in the usual setting properties. By passing in a representative element, the client is telling the instrumentation to use additional vendor-specific information about that storage element when trying to create a composite. This allows for better interoperability because it hides those vendor rules, while still supporting vendor needs. If `Goal` or `RepresentativeElement` is non-null, the other shall be null. `InElements[]` can also be used to deduce QoS setting to use in case neither `Goal` or `RepresentativeElement` is specified. In this case, the QoS for the composite element will be the lowest common denominator of the QoS values for the `InElements` array.

24.5.1.1 Creating a Composite

When creating a new composite storage element, there are two distinct modes of operation. Regardless of which mode is used, the following values shall apply:

The `TheElement` parameter shall be NULL. `ElementName` may be specified if the instrumentation supports naming of composite elements. `CompositeType` may be specified if the instrumentation supports the setting of this parameter. See the `CompositeTypesSupported` property in the `ElementCompositionCapabilities` class to determine if this can be set. `Job` will be non-NULL upon the method return if a `Job` was created.

The two creation use cases are the following:

- Pass in a non-empty list of extents (e.g. StorageVolumes) in InElements[]. The RepresentativeElement and Goal parameters may be NULL as the instrumentation will pick up the QoS goal from the InElements. If RepresentativeElement is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the RepresentativeElement. It shall fail if it cannot create a composite that satisfies that QoS. If Goal is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the Goal. It shall fail if it cannot create a composite that satisfies that Goal. The user may specify RepresentativeElement or Goal, but not both. If the Size parameter is NULL, the Instrumentation shall create a composite whose size is the sum of the ConsumableBlocks times BlockSize of the InElements[] entries. If Size is not NULL, then the instrumentation shall attempt to create a composite volume whose size is equal to or greater than the Size value passed in. Note that this may mean not all InExtents are consumed by the composite. The instrumentation may allow partial consumption of storage elements, in which case Extent Composition shall be used to show the relationship between the storage elements and underlying StorageExtents. The ElementSource parameter shall be NULL.
- Pass in a Size and a NULL InElements parameter. In this case, the instrumentation shall find the elements to use, based on the value of the ElementSource parameter, which may be NULL, indicating the instrumentation will determine the source of the elements. One of Goal and RepresentativeElement shall be specified. If RepresentativeElement is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the RepresentativeElement. It shall fail if it cannot create a composite that satisfies that QoS. If Goal is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the Goal. It shall fail if it cannot create a composite that satisfies that Goal. The user may specify RepresentativeElement or Goal, but not both. The size of the composite created shall be equal to or greater than the Size passed in.

24.5.1.2 Modifying a Composite

When modifying a composite, the user shall pay close attention to the supported capabilities of the instrumentation.

Modified a composite is similar to creation, with a few differences. The key difference is that TheElement shall be specified. ElementName may be specified if the instrumentation supports naming of composite elements. CompositeType may be specified if the instrumentation supports the setting of this parameter. See the CompositeTypesSupported property in the ElementCompositionCapabilities class to determine if this can be set. Job will be non-NULL upon the method return if a Job was created.

The two modification use cases are the following:

- Pass in a non-empty list of extents (e.g. StorageVolumes) in InElements[]. The RepresentativeElement and Goal parameters may be NULL as the instrumentation will pick up the QoS goal from the existing composite and the InElements. If RepresentativeElement is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the RepresentativeElement. It shall fail if it cannot modify the composite to satisfies that QoS. If Goal is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the Goal. It shall fail if it cannot modify a composite to satisfy that Goal. The user may specify RepresentativeElement or Goal, but not both. If the Size parameter is NULL, the Instrumentation shall modify the composite size to be the current size plus the sum of the ConsumableBlocks times BlockSize of the InElements[] entries. If Size is not NULL, then the instrumentation shall attempt to modify the composite to a size equal to or greater than the Size value passed in. Note that this may mean that not all InExtents are consumed by the composite or that volumes in the composite may be removed from the composite. The instrumentation may allow partial consumption of storage elements, in which case Extent Composition shall be used to show the relationship between the storage elements and underlying StorageExtents. The ElementSource parameter shall be NULL.
- Pass in a Size and a NULL InElements parameter. In this case, the instrumentation shall find the elements to use, based on the value of the ElementSource parameter, which may be NULL, indicating the instrumentation will determine the source of the elements. One or both of Goal and RepresentativeElement shall be specified. If RepresentativeElement is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the RepresentativeElement. It shall fail if it cannot modify the composite to satisfy that QoS. If Goal is not NULL, the instrumentation shall attempt to satisfy the QoS settings in the Goal. It shall fail if it cannot modify a composite to satisfy that Goal. The user may specify RepresentativeElement or Goal, but not both. The size of

the composite created shall be equal to or greater than the Size passed in. If Size is smaller than the current composite size, this may mean that volumes in the composite may remove from the composite.

Table 318: CreateOrModifyCompositeElement

Method: CreateOrModifyCompositeElement			
Return Values:			
Value		Description	
0: Success		Job completed with no error.	
1: Not Supported		Not supported	
2: Unknown		Unknown error occurred	
3: Timeout		Timeout	
4: Failed		Method failed.	
5: Invalid Parameter			
6: In Use		Element is in use and cannot be modified	
4096: Method Parameters Checked - Job started		Job was started	
4097: Size Not supported			
Parameters:			
Qualifiers	Name	Type	Description/Values
IN	ElementName	string	End-user relevant name for the element created
IN	ElementType	uint16	Type of element being created
OUT	Job	REF ConcreteJob	Reference to the job created
IN	Goal	REF ManagedElement	The QoS requirements for the composite element to maintain. This parameter may be null. Both Goal and RepresentativeElement may not be non-null
IN	RepresentativeElement	REF StorageExtent	The instrumentation will use this parameter + Size or InElements to determine the elements used to construct the composite. This parameter may be NULL

Table 318: CreateOrModifyCompositeElement

Method: CreateOrModifyCompositeElement			
IN/OUT	Size	uint64	<p>Unit: bytes</p> <p>As an input parameter Size specifies the desired size. If NULL, then InElements shall be supplied. If not NULL, this parameter will supply a new size when creating or modifying an existing element.</p> <p>As an output parameter Size specifies the size achieved.</p>
IN	InElements[]	REF StorageExtent	<p>The elements from which to create the composite element. If this parameter is NULL then Size shall be non-NULL.</p> <p>Once the elements are combined, they will be removed from the model and replaced with a single element.</p> <p>For some instrumentation, this may be one of the InElements, so in effect, all but one are removed.</p>
IN/OUT	TheElement	REF LogicalElement	<p>When used to create a composite, this shall be NULL</p> <p>Upon modification, this shall specify an existing composite element. The method will then modify the specified element. Upon completion (unless a Job is started), a reference to the resulting element shall be returned</p>
IN	CompositeType	uint16	<p>Type of composite element to create. Possible values are Concatenate, Stripe, Concatenate+Stripe, Vendor specific.</p> <p>If NULL, the instrumentation will decide</p>

Table 318: CreateOrModifyCompositeElement

Method: CreateOrModifyCompositeElement			
IN	ElementSource	uint16	<p>Tell the instrumentation where to get the elements. Only applies when Size is specified and not InElements. Otherwise it shall be NULL.</p> <p>Possible values are:</p> <ol style="list-style-type: none"> 1. Use existing elements only 2. Create new elements only 3. Can use existing or create new or both 4. Instrumentation decides <p>If NULL, the instrumentation will decide.</p>

24.5.2 ReturnElementToElements

This method is found in the SNIA_ElementCompositionService. It dissolves a composite into its constituent elements. Note that the elements returned may not match the elements that went into the composite (e.g., VPD page 83h information may not be the same).

Table 319: ReturnElementToElements

Method: ReturnElementToElements			
Return Values:			
Value		Description	
0: Success		Job completed with no error.	
1: Not Supported		Method not supported	
2: Unknown		Unknown error occurred	
3: Timeout		Operation timed out	
4: Failed		Operation failed	
5: Invalid Parameter		Invalid parameter	
6: In use		Element is in use and cannot be dissolved	
4096: Method Parameters Checked - Job started		Job was started	
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	REF ConcreteJob	Reference to the job created
IN	TheElement	REF LogicalElement	The composite element to dissolve
OUT	OutElements[]	REF StorageExtent	Elements the composite was dissolved into

24.5.3 GetAvailableElements

This method, found in the SNIA_ElementCompositionService, queries the set of pools passed in and returns a set of elements (volumes or logical disks) that can be composed together based on the specified goal and element passed in. Since there are usually complicated vendor-specific rules for creating these composite volumes, using the representative element can supply more vendor-specific information than there would be in an interoperable setting. The client can then use some or all of this list in a call to CreateOrModifyCompositeElement().

In this version of the specification, only StorageVolumes shall be supported as the ElementType.

Table 320: GetAvailableElements

Method: GetAvailableElements			
Return Values:			
Value		Description	
0: Success		Job completed with no error.	
1: Not Supported		Method not supported	
2: Unknown		Unknown error occurred	
3: Timeout		Operation timed out	
4: Failed		Operation failed	
5: Invalid Parameter		Invalid parameter	
6: In use		Element is in use and cannot be dissolved	
4096: Method Parameters Checked - Job started		Job was started	
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	REF ConcreteJob	Reference to the job created
IN	InPools[]	REF StoragePool	List of pools to look in
IN	Goal	REF StorageSetting	The QoS goal requirements for the composite element. Can be NULL. If it is NULL, then RepresentativeElement shall be non-NULL
IN	ElementType	uint16	Enumeration indicating the type of element being created or modified Values: 2: StorageVolume 3: LogicalDisk
IN	RepresentativeElement	REF StorageExtent	Serves as a guide to help the instrumentation determine which elements to return. It shall be a member f one of the pools passed in. This may be NULL, only if Goal is non-NULL

Table 320: GetAvailableElements

Method: GetAvailableElements			
OUT	Candidates[]	REF StorageExtent	The elements that can be used to create the composite element. These will be an array of references to StorageVolumes or LogicalDisks.

24.5.4 GetCompositeElements

This method is found in the SNIA_ElementCompositionService. It is used to query an existing composite element to determine the component elements that make up that composite element.

Table 321: GetCompositeElements

Method: GetCompositeElements			
Return Values:			
Value		Description	
0: Success		Job completed with no error.	
1: Not Supported		Method not supported	
2: Unknown		Unknown error occurred	
3: Timeout		Operation timed out	
4: Failed		Operation failed	
5: Invalid Parameter		Invalid parameter	
6: In use		Element is in use and cannot be accessed	
4096: Method Parameters Checked - Job started		Job was started	
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	REF ConcreteJob	Reference to the job created
IN	TheElement-Pools[]	REF StorageExtent	The element to query
OUT	OutElements[]	REF StorageExtent	The elements that comprise the composite.

24.6 Client Considerations and Recipes**24.6.1 Indications**

When storage elements are combined into a composite or a composite is dissolved, indications shall be sent. When a composite is created, the instrumentation shall send an InstDelete indication for all volumes that no longer exist as StorageVolumes. If the storage element still exists but is no longer accessible, the provider may send an InstModification indication depending upon whether or not there are any changes to the storage element itself. If the instrumentation creates a new storage element, then it shall send an InstCreation indication for the new

element. If the instrumentation modifies an existing element and it becomes the element to represent a composite, an `InstModification` indication shall be sent.

When a composite is dissolved, the instrumentation shall send an `InstCreation` indication for each storage element created. It shall send an `InstDeletion` indication if the composite element is deleted and an `InstModification` indication if the composite element is merely modified.

Indications shall not be sent for the creation, modification, or deletion of any `StorageSettings` associated to the storage elements that are created, modified, or deleted. The user is advised to check the `StorageSetting` for the storage elements they are interested in after composite creation or deletion as those settings may have changed from what they were before.

24.6.2 Recipe 1: Create Composite Volume

In this use case, all available storage is consumed in `StorageVolumes`. The client wishes to create a larger volume from volumes that are not being used. The client will call `CreateOrModifyElementFromElements()`, passing in a set storage volumes. The provider will then create the concatenated volume.

```
// DESCRIPTION:
//
// Create a composite volume
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The SNIA_ElementCompositionService has been found and the object path
//    value is stored in $CompositionService->
// 2. The list of elements (volumes) to use to create the composite has been
//    identified and the object path values are stored in $volumes->[]
// 3. The StorageSetting to use has been identified and the object path
//    values are stored in $Goal->
// 4. The type of element to create has been selected (LogicalDisk or
//    StorageVolume and it's value stored in #ElementType
// 5. A representative element (LogicalDisk or StorageVolume instance)
//    has been identified and it's object path stored in $SampleElement->
//    Note: this is allowed to be null
// 6. The SNIA_ElementCompositionCapabilities associated to the
//    SNIA_ElementCompositionService has been found and the instance
//    stored in $CompositionCapabilities
// 7. The type of composite to create has been identified and the value
//    stored in #CompositeType

// Determine if there is a job created by method
// and wait for the job to complete
// Input:
//   #ReturnCode : The return code of the method
//   $ConcreteJob-> :The output parameter that may have a ConcreteJob REF.
// This method will return control if the recipe was not exited because of error
sub void WaitForJob(#ReturnCode, $ConcreteJob->) {
    if (4096 == #ReturnCode) {
        if ($ConcreteJob-> != null) {
            /*Wait until the completion of the job using $ConcreteJob-> as
```

Volume Composition Profile

```

        a filter Verify that the OperationalStatus contains 2 ("OK"),
        or 17 ("Completed") */
        $JobInstance = GetInstance($ConcreteJob->,
            false, false, false, null)
        if ($JobInstance.JobState != 7) { // 7 - Completed
            <ERROR! Job failed! >
        }
    } else {
        <ERROR! Missing Job reference>
    }
}

// Step 1. See if creation is supported

if ($CompositionCapabilities.SupportsComposites == false) {
    <ERROR! Volume composition not supported>
}

if ((contains("CreateOrModifyCompositeElement",
    $CompositionCapabilities.SupportedAsynchronousActions[]) == FALSE)
    && (contains("CreateOrModifyCompositeElement",
    $CompositionCapabilities.SupportedSynchronousActions[]) == FALSE)) {
    <ERROR! Volume composition creation not supported>
}

// Step 2. Subscribe to indications
#Filter1 = "SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA
            CIM_StorageVolume";
#Filter2 = "SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA
            CIM_StorageVolume";
// Determine if the Indication filters already exist
// If they don't, create them

// Step 3. Create the composite
%InputArguments["ElementName"] = {"Test"}
%InputArguments["ElementType"] = #ElementType
%InputArguments["Goal"] = $Goal->
%InputArguments["InElements"] = {$volumes->}
%InputArguments["ElementType"] = #CompositeType

#ReturnCode = InvokeMethod($CompositionService->,
    "CreateOrModifyCompositeElement",
    %InputArguments,
    %OutputArguments)

// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

```

```

}

$Job-> = %OutputArguments["Job"]
if ($Job-> == null) {
    $CompositeCreated-> = %OutputArguments["TheElement"]
}
else {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $Job->)

    // Now get the composite just created
    $CompositeCreated-> = Associators($Job->,
        "CIM_AffectedJobElement",
        "CIM_StorageExtent",
        "AffectingElement",
        "AffectedElement",
        false, false, null)
}

// Step 4: Verify the volumes that went into the composite are no longer accessible
for #i in $Candidates->[] {
    if ($Candidates->[#i] == $CompositeCreated->) {
        // It's allowed for the created composite to take over the
        // identity of a volume that went into creating it
    }
    else {
        $Refs->[] = AssociatorNames(
            $Candidates->[#i],
            "CIM_SystemDevice", // AssocClass
            "CIM_ComputerSystem", // ResultClass
            "PartComponent", // Role
            null )
        if( $Refs->[].length != 0 )
        {
            <"ERROR! Composite volume component still exists">
        }
    }
}
}

```

24.6.3 Recipe 2: Delete Composite Volume

In this use case, the client wishes to return a concatenated volume to its individual component storage volumes. The client calls `ReturnElementToElements()` to dissolve the concatenated volume.

```

// DESCRIPTION:
//

```

Volume Composition Profile

```
// Delete a composite volume
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The SNIA_ElementCompositionService has been found and the object path
//    value is stored in $CompositionService->
// 2. A composite volume has been created and the object path stored in
//    $compositeVolume->
// 3. The SNIA_ElementCompositionCapabilities associated to the
//    SNIA_ElementCompositionService has been found and the instance
//    stored in $CompositionCapabilities

// Determine if there is a job created by method
// and wait for the job to complete
// Input:
//   #ReturnCode : The return code of the method
//   $ConcreteJob-> :The output parameter that may have a ConcreteJob REF.
// This method will return control if the recipe was not exited because of error
sub void WaitForJob(#ReturnCode, $ConcreteJob->) {
    if (4096 == #ReturnCode) {
        if ($ConcreteJob-> != null) {
            /*Wait until the completion of the job using $ConcreteJob-> as
            a filter Verify that the OperationalStatus contains 2 ("OK"),
            or 17 ("Completed") */
            $JobInstance = GetInstance($ConcreteJob->,
                false, false, false, null)
            if ($JobInstance.JobState != 7) { // 7 - Completed
                <ERROR! Job failed! >
            }
        } else {
            <ERROR! Missing Job reference>
        }
    }
}

// Step 1. Subscribe to indications
TBD

// Step 2. Dissolve the composite volume

if ($CompositionCapabilities.SupportsComposites == false) {
    <ERROR! Volume composition not supported>
}

%InputArguments["TheElement"] = $CompositeVolume->

#ReturnCode = InvokeMethod($CompositionService->,
```



```

        "ReturnElementToElement",
        %InputArguments,
        %OutputArguments)
// 0 is "Success" and 4096 is "Method Parameters Checked - Job Started"
if (#ReturnCode != 0 || #ReturnCode != 4096) {
    <ERROR! Method failure>
}

$Job-> = %OutputArguments["Job"]
if ($Job-> == null) {
    $Volumes->[] = %OutputArguments["OutElements"]
}
else {
    // Wait until job is finished
    &WaitForJob(#ReturnCode, $Job->)

    // Now get the SPCs
    $Volumes->[] = Associators(
        $Job->,
        "CIM_AffectedJobElement",
        "CIM_StorageExtent",
        "AffectingElement",
        "AffectedElement",
        false, false, null)
}

// Step 2. Show the elements from the former composite
for #v in $Volumes->[] {
    $AnInstance = GetInstance($Volumes->[#i],
                                false, false, false, null)
    // Display instance
}

```

24.7 Registered Name and Version

Volume Composition version 1.2.0

24.8 CIM Elements

Table 322: CIM Elements for Volume Composition

Element Name	Requirement	Description
CIM_AllocatedFromStoragePool (24.8.1)	Mandatory	
CIM_BasedOn (Volume Composition) (24.8.2)	Mandatory	
CIM_BasedOn (Extent Composition) (24.8.3)	Mandatory	
CIM_ConcreteComponent (24.8.4)	Mandatory	
CIM_CompositeExtent (24.8.5)	Mandatory	
CIM_CompositeExtentBasedOn (Volume Composition) (24.8.6)	Mandatory	
CIM_ElementCapabilities (24.8.7)	Mandatory	
CIM_HostedService (Associates ComputerSystem and the ElementCompositionService) (24.8.8)	Mandatory	
CIM_StorageExtent (24.8.9)	Mandatory	
CIM_StoragePool (24.8.10)	Mandatory	
CIM_StorageVolume (24.8.11)	Conditional	Conditional requirement: Storage Volumes used as storage elements.
SNIA_ElementCompositionCapabilities (24.8.12)	Mandatory	
SNIA_ElementCompositionService (24.8.13)	Mandatory	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Conditional requirement: Storage Volumes used as storage elements.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Conditional requirement: Storage Volumes used as storage elements.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume	Conditional	Conditional requirement: Storage Volumes used as storage elements.

24.8.1 CIM_AllocatedFromStoragePool

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 323 describes class CIM_AllocatedFromStoragePool.

Table 323: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

24.8.2 CIM_BasedOn (Volume Composition)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 324 describes class CIM_BasedOn (Volume Composition).

Table 324: SMI Referenced Properties/Methods for CIM_BasedOn (Volume Composition)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

24.8.3 CIM_BasedOn (Extent Composition)

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 325 describes class CIM_BasedOn (Extent Composition).

Table 325: SMI Referenced Properties/Methods for CIM_BasedOn (Extent Composition)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

24.8.4 CIM_ConcreteComponent

Created By: External

Modified By: External

Deleted By: External

Class Mandatory: Mandatory

Table 326 describes class CIM_ConcreteComponent.

Table 326: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

24.8.5 CIM_CompositeExtent

Created By: Extrinsic

Modified By: Extrinsic

Deleted By: Extrinsic

Class Mandatory: Mandatory

Table 327 describes class CIM_CompositeExtent.

Table 327: SMI Referenced Properties/Methods for CIM_CompositeExtent

Properties	Flags	Requirement	Description & Notes
IsConcatenated		Mandatory	Indicates data is concatenated across extents in the group
BlockSize		Mandatory	Size in bytes of the blocks which form this StorageExtent.
NumberOfBlocks		Mandatory	
ConsumableBlocks		Mandatory	The maximum number of blocks, of size BlockSize, which are available for consumption
SystemCreationClassName		Mandatory	The scoping System CreationClassName
SystemName		Mandatory	The scoping System Name
CreationClassName		Mandatory	The name of the concrete subclass
Name		Mandatory	Unique identifier for the Service

24.8.6 CIM_CompositeExtentBasedOn (Volume Composition)

Created By: Extrinsic: CreateOrModifyCompositeElement, ReturnElementToElements

Modified By: External

Deleted By: Extrinsic: CreateOrModifyCompositeElement, ReturnElementToElements

Class Mandatory: Mandatory

Table 328 describes class CIM_CompositeExtentBasedOn (Volume Composition).

Table 328: SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn (Volume Composition)

Properties	Flags	Requirement	Description & Notes
Dependent		Mandatory	
Antecedent		Mandatory	

24.8.7 CIM_ElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 329 describes class CIM_ElementCapabilities.

Table 329: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	
Capabilities		Mandatory	

24.8.8 CIM_HostedService (Associates ComputerSystem and the ElementCompositionService)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 330 describes class CIM_HostedService (Associates ComputerSystem and the ElementCompositionService).

Table 330: SMI Referenced Properties/Methods for CIM_HostedService (Associates Computer-System and the ElementCompositionService)

Properties	Flags	Requirement	Description & Notes
Antecedent		Mandatory	
Dependent		Mandatory	

24.8.9 CIM_StorageExtent

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 331 describes class CIM_StorageExtent.

Table 331: SMI Referenced Properties/Methods for CIM_StorageExtent

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks as reported by the hardware.
ConsumableBlocks		Mandatory	The number of usable blocks.
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

24.8.10 CIM_StoragePool

Created By: External

Modified By: External

Deleted By: External

Class Mandatory:

Table 332 describes class CIM_StoragePool.

Table 332: SMI Referenced Properties/Methods for CIM_StoragePool

Properties	Flags	Requirement	Description & Notes
Primordial		Mandatory	
TotalManagedSpace		Mandatory	
RemainingManagedSpace		Mandatory	

24.8.11 CIM_StorageVolume

Created By: Extrinsic: ReturnElementToElements

Modified By: External

Deleted By: Extrinsic: CreateOrModifyCompositeElement, ReturnElementToElements

Class Mandatory: StorageVolumeCondition

Table 333 describes class CIM_StorageVolume.

Table 333: SMI Referenced Properties/Methods for CIM_StorageVolume

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	The number of blocks as reported by the hardware.
ConsumableBlocks		Mandatory	The number of usable blocks.
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	

24.8.12 SNIA_ElementCompositionCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 334 describes class SNIA_ElementCompositionCapabilities.

Table 334: SMI Referenced Properties/Methods for SNIA_ElementCompositionCapabilities

Properties	Flags	Requirement	Description & Notes
ElementName		Mandatory	User friendly name for this instance of Capabilities.
InstanceID		Mandatory	Unique identifier for the instance
SupportsComposites		Mandatory	Indicates if instrumentation supports composite elements
MaxCompositeSize		Mandatory	Indicates the largest composite element that can be created in bytes
MaxCompositeElements		Mandatory	Indicates the most elements that can be combined into a composite element.
CompositionCharacteristics		Mandatory	Composition "characteristics supported by this " "system.
CompositeSourcesSupported		Mandatory	
SupportedAsynchronousActions		Mandatory	Indicates which methods are executed asynchronously
SupportedSynchronousActions		Mandatory	Indicates which methods are executed synchronously
SupportedStorageElements		Mandatory	Managed element types that can be composited. Currently only StorageVolume
SupportsCompositeNaming		Mandatory	Can the user name the composite
SupportsRepresentativeElement		Mandatory	Can the user specify the RepresentativeElement in CreateOrModifyComposite and GetAvailableElements

24.8.13 SNIA_ElementCompositionService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 335 describes class SNIA_ElementCompositionService.

Table 335: SMI Referenced Properties/Methods for SNIA_ElementCompositionService

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	The scoping System CreationClassName
SystemName		Mandatory	The scoping System Name
CreationClassName		Mandatory	The name of the concrete subclass
Name		Mandatory	Unique identifier for the Service
CreateOrModifyCompositeElement()		Mandatory	This method creates or modifies a composite element. Only like elements (e.g. StorageVolumes) can be combined
ReturnElementToElements()		Mandatory	Dissolve the composite. All elements in the composite are restored
GetAvailableElements()		Optional	This method queries the set of pools passed in and returns a set of volumes or logical disks that can be composed together based on the specified goal and element passed in
GetCompositeElements()		Optional	Returns list of volumes/logical disks that were combined into this composite volume. Since (usually) all but one of these volumes/logical disks disappear when the composite is created, this is an essential method to help the client figure out what is in the composite. Remember that a particular client may not have been the one to create the composite

EXPERIMENTAL

25.1 Description

A host volume manager is a software storage management subsystem that allows one to manage physical disks as logical devices called volumes. A volume is a logical device that appears to data management systems as a physical disk. Through support of RAID, the volume manager provides similar features as many disk arrays. Therefore, CIM administration of a volume manager is similar to that of an array. Embedded volume managers, like in a switch, should use the virtualization profile.

25.1.1 Instance Diagram



SMI-S 1.2.0 Revision 6

operating system device name. The volume manager provider will place into a primordial pool all disks that it discovers as a LogicalDisk and uses the Name property to specify the operating system device file name.

25.1.3 Export Class of the Volume Manager

The Volume Management profile exports LogicalDisk, which may be referred to as a volume in a typical host volume manager. For host volume managers, this is treated as a virtual disk or volume, and is where a file system or database would reside on.

25.1.4 Initializing OS Disks for Volume Manager Use

All disks initially discovered by the volume manager from the host's device tree are added to a Primordial Pool by creating an association between the Primordial Pool and a LogicalDisk instance. Typically, these discovered disks are those listed in the /dev directory. Disks on a host are not immediately available for volume manager use; they are first initialized for volume manager use by writing metadata to the disk. Any disks that are not yet initialized for volume manager use will become initialized as a side effect of creating a concrete StoragePool.

25.1.5 Creating Pools and Logical Volumes

Concrete StoragePools are created by the Block Services CreateOrModifyStoragePool method.

Any uninitialized disks that are added to the concrete StoragePool are initialized as a side-effect of adding the disk to the pool.

The Block Services methods CreateOrModifyElementFromStoragePool or CreateOrModifyElementFromElements are used to create and modify volumes. When specifying a primordial pool or uninitialized disks to create or modify volumes, any disks that are not yet initialized will be initialized as a side effect of adding the disks to a concrete pool and creating the volume. See 5.1 Description for more details on methods for creating pools and logical volumes.

25.1.6 Storage Settings for Volumes

Providers need to map a Quality of Service and any Storage Settings to a particular volume's redundancy or raid level. This is similar to creating StorageVolumes in the Block Services subprofile.

The StorageSetting, StorageSettingWithHints, and StorageCapabilities classes may be used to specify striping parameters such as number of stripe columns, or the extent stripe length. See Clause 5, "Block Services Package" for a description of these settings. The StorageSettings.Description string should be updated with an appropriate string describing the volume's settings. The Exported value in ExtentStatus[] of the LogicalDisk should be set if it is intended for application use.

25.1.7 Durable Names and Other Correlatable ids of the Profile

Each object's Name in the volume manager is not durable. The names can be changed at any time. However, names will always be unique and correlatable. The provider will present names that the underlying volume manager software creates using its own naming heuristics. When available, the Host Discovered Resources profile provides the connectivity and correlatable IDs of the host resources.

25.2 Health and Fault Management Considerations

Not defined in this standard.

25.3 Cascading Considerations

The Cascading subprofile may be used when the Host Discovered Resources profile is available on the host, where the Host Discovered Resource profile would be the leaf profile. In this case, all discovered disks by the provider are still placed in the primordial pool. Therefore, the behavior of what is in the primordial pool should not change based on the presence of another profile. The content should be consistent regardless of the presence of the Host Discovered Resources profile. The the description of the Cascading subprofile for usage with the Security Resource Ownership Subprofile.

25.4 Supported Subprofiles and Packages

Table 336: Supported Profiles for Volume Management

Registered Profile Names	Mandatory	Version
Access Points	No	1.2.0
Extent Composition	No	1.2.0
Location	No	1.2.0
Software	No	1.2.0
Copy Services	No	1.2.0
Disk Sparing	No	1.2.0
Multi System	No	1.2.0
Job Control	No	1.2.0
Cascading	Yes	1.2.0
Block Services Resource Ownership	No	1.2.0
Block Server Performance	No	1.2.0
Block Services	Yes	1.2.0
Health	Yes	1.2.0

25.5 Methods of the Profile

None

25.6 Client Considerations and Recipes

Use Clause 5: Block Services Package to create and modify volumes.

See recipes for creating volumes in Clause 5: Block Services Package.

Replacing a disk is done by using the Sparing subprofile. Newly added disks are first made and then are used to replace the old disk.

25.6.1 Storage Configuration

The Volume Management profile uses the StorageConfigurationService in the Block Services subprofile for creating and modifying objects in a StoragePool. Creating volumes with specified extents shall be done using the CreateOrModifyElementFromElements method. When specifying extents, or when using the InExtents[] parameter of CreateOrModifyStoragePool for creating storage pools as well as adding disks, then the specified extents shall be from among the extents returned from the StoragePool.GetAvailableExtents method. Any other extents may cause the operation to fail.

25.7 Registered Name and Version

Volume Management version 1.2.0

25.8 CIM Elements

Table 337: CIM Elements for Volume Management

Element Name	Requirement	Description
CIM_ComputerSystem (25.8.1)	Mandatory	
CIM_SystemDevice (25.8.2)	Mandatory	
CIM_HostedStoragePool (25.8.3)	Mandatory	
CIM_LogicalDisk (25.8.4)	Mandatory	
CIM_ElementSettingData (25.8.5)	Mandatory	
CIM_StorageSetting (25.8.6)	Mandatory	
CIM_AllocatedFromStoragePool (LogicalDisk from Pool) (25.8.7)	Mandatory	
CIM_AllocatedFromStoragePool (Pool from Pool) (25.8.8)	Mandatory	
CIM_StoragePool (Concrete) (25.8.9)	Mandatory	Logical Disks are allocated from 'concrete' pools.
CIM_StoragePool (Primordial) (25.8.10)	Mandatory	At least one primordial pool must exist for a host. This is the 'unallocated storage' of the host, and contains unused disks.
CIM_ElementCapabilities (25.8.11)	Mandatory	
CIM_StorageCapabilities (25.8.12)	Mandatory	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Mandatory	Addition of a new logical disk instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Mandatory	Deletion of a logical disk instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Mandatory	Change of status of a Logical Disk
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 Logical Disk

Table 337: CIM Elements for Volume Management

Element Name	Requirement	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StoragePool AND SourceInstance.CIM_StoragePool::OperationalStatus <> PreviousInstance.CIM_StoragePool::OperationalStatus	Optional	Experimental CQL - Change of status of a storage pool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Addition of a storage pool instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Mandatory	Deletion of a storage pool instance

25.8.1 CIM_ComputerSystem

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 338 describes class CIM_ComputerSystem.

Table 338: SMI Referenced Properties/Methods for CIM_ComputerSystem

Properties	Flags	Requirement	Description & Notes
CreationClassName		Mandatory	
Name		Mandatory	Unique identifier for the Host. IP address
ElementName		Mandatory	User friendly name
OperationalStatus		Mandatory	Overall status of the Host
NameFormat		Mandatory	Format for Name property.
Dedicated		Mandatory	This should include 0. This indicates that this computer system is not dedicated to volume management.
PrimaryOwnerContact		Optional	
PrimaryOwnerName		Optional	

25.8.2 CIM_SystemDevice

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 339 describes class CIM_SystemDevice.

Table 339: SMI Referenced Properties/Methods for CIM_SystemDevice

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

25.8.3 CIM_HostedStoragePool

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 340 describes class CIM_HostedStoragePool.

Table 340: SMI Referenced Properties/Methods for CIM_HostedStoragePool

Properties	Flags	Requirement	Description & Notes
PartComponent		Mandatory	
GroupComponent		Mandatory	

25.8.4 CIM_LogicalDisk

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 341 describes class CIM_LogicalDisk.

Table 341: SMI Referenced Properties/Methods for CIM_LogicalDisk

Properties	Flags	Requirement	Description & Notes
SystemCreationClassName		Mandatory	
SystemName		Mandatory	
CreationClassName		Mandatory	
DeviceID		Mandatory	Opaque identifier
ElementName		Mandatory	User friendly name
Name		Mandatory	Should be a durable name. As yet any name
ExtentStatus		Mandatory	
OperationalStatus		Mandatory	
BlockSize		Mandatory	
NumberOfBlocks		Mandatory	
IsBasedOnUnderlyingRedundancy		Mandatory	
NoSinglePointOfFailure		Mandatory	
DataRedundancy		Mandatory	
PackageRedundancy		Mandatory	
DeltaReservation		Mandatory	
ConsumableBlocks		Mandatory	

25.8.5 CIM_ElementSettingData

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 342 describes class CIM_ElementSettingData.

Table 342: SMI Referenced Properties/Methods for CIM_ElementSettingData

Properties	Flags	Requirement	Description & Notes
SettingData		Mandatory	

Table 342: SMI Referenced Properties/Methods for CIM_ElementSettingData

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	

25.8.6 CIM_StorageSetting

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 343 describes class CIM_StorageSetting.

Table 343: SMI Referenced Properties/Methods for CIM_StorageSetting

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	Opaque identifier
ElementName		Mandatory	User friendly name.... can be used for 'potted' settings for specific RAID levels.
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
DataRedundancyGoal		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	
PackageRedundancyGoal		Mandatory	
DeltaReservationGoal		Mandatory	
DeltaReservationMax		Mandatory	
DeltaReservationMin		Mandatory	

25.8.7 CIM_AllocatedFromStoragePool (LogicalDisk from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 344 describes class CIM_AllocatedFromStoragePool (LogicalDisk from Pool).

Table 344: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (LogicalDisk from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Antecedent		Mandatory	
Dependent		Mandatory	

25.8.8 CIM_AllocatedFromStoragePool (Pool from Pool)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 345 describes class CIM_AllocatedFromStoragePool (Pool from Pool).

Table 345: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool (Pool from Pool)

Properties	Flags	Requirement	Description & Notes
SpaceConsumed		Mandatory	
Dependent		Mandatory	
Antecedent		Mandatory	

25.8.9 CIM_StoragePool (Concrete)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 346 describes class CIM_StoragePool (Concrete).

Table 346: SMI Referenced Properties/Methods for CIM_StoragePool (Concrete)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
PoolID		Mandatory	
TotalManagedSpace		Mandatory	
RemainingManaged Space		Mandatory	
Primordial		Mandatory	Set to false

25.8.10 CIM_StoragePool (Primordial)

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 347 describes class CIM_StoragePool (Primordial).

Table 347: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial)

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
PoolID		Mandatory	
TotalManagedSpace		Mandatory	
RemainingManaged Space		Mandatory	
Primordial		Mandatory	Set to true

25.8.11 CIM_ElementCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 348 describes class CIM_ElementCapabilities.

Table 348: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Properties	Flags	Requirement	Description & Notes
Capabilities		Mandatory	
ManagedElement		Mandatory	

25.8.12 CIM_StorageCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory: Mandatory

Table 349 describes class CIM_StorageCapabilities.

Table 349: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
InstanceID		Mandatory	
ElementName		Mandatory	
NoSinglePointOfFailure		Mandatory	
NoSinglePointOfFailureDefault		Mandatory	
DataRedundancyMin		Mandatory	
DataRedundancyMax		Mandatory	
DataRedundancyDefault		Mandatory	
PackageRedundancyMin		Mandatory	
PackageRedundancyMax		Mandatory	
PackageRedundancyDefault		Mandatory	
DeltaReservationDefault		Mandatory	

Table 349: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Properties	Flags	Requirement	Description & Notes
DeltaReservationMax		Mandatory	
DeltaReservationMin		Mandatory	

EXPERIMENTAL

EXPERIMENTAL

Clause 26: Storage Element Protection SubProfile

26.1 Description

26.1.1 Overview

The Storage Element Protection subprofile defines classes and methods for managing access permission to a storage element—either a storage volume or logical disk. This subprofile also defines how long the protection shall stay in effect. It allows a client to protect data as required by changeable business and operational policies. Clients may modify access to a storage element for various reasons, including:

- *Regulatory Compliance* - Ensure that vital records are available, unaltered (immutable) and protected from accidental or malicious destruction. The degree of exposure and the retention period depend on the nature of the records.
- *Protection of Fixed Content* - Maintain in “Read-only” mode between cyclic refreshes of the data content.
- *Protection of Recovery Assets* - Protect data from accidental reuse. For example, make recovery logs “Read-only” or immutable.
- *Reclamation of Expired (Archive) Capacity* - After migration, delete or destroy data when elements are released for re-use.

26.1.2 Use Cases

In a typical scenario, a storage element is allocated with Read/Write permission. At a later time, when the element holds data that requires protection, the access permission is changed to Read-only with a retention period.

Changes in regulations, audit or litigation may require that the storage element be retained for a longer period. In this case, the retention period may be extended or alternatively set to a "never to expire" value. This new setting retains the current protection for an indefinite period--until litigation is resolved, for example.

Company policy may dictate that archived data, although still protected and retained for legal purposes, be unavailable even for Read-only. In this case, the element may be hidden from read-and-write access. It will be visible only to a storage administrator.

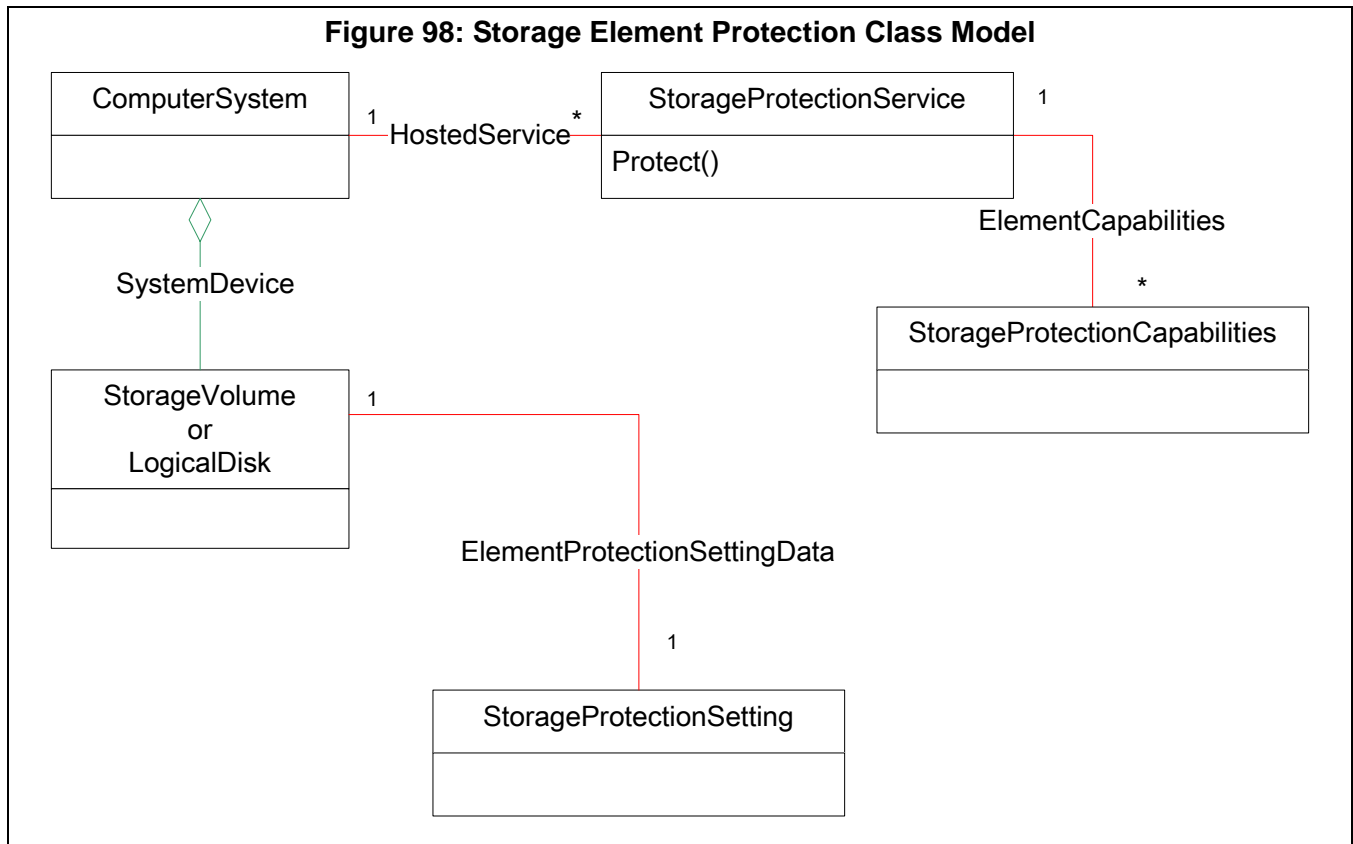
26.1.3 Functionality

A management application will interact with this subprofile in two ways—(1) the management application can retrieve and modify the access permission attribute and (2) the management application may define the period for which the access permission will remain in effect (the retention period). During the retention period, other functions shall be disabled to prevent the storage element from being reformatted, erased or otherwise (logically) destroyed. While this retention period is in effect, the access permission cannot be modified except to make it more restrictive. Once this period expires, the access permissions remain in effect, but they may now be modified. The management application may extend this retention period but shall not be able to shorten it.

26.1.4 Class Model

In order to support the desired protection functionality, this profile defines a new method, Protect, for the StorageProtectionService class. This method allows the client to set the protection-related configurations of a storage element, either a StorageVolume or LogicalDisk. When first called for a storage element, it creates a StorageProtectionSetting instance with the client requested configuration and associates it to the target element by the ElementProtectionSettingData association. If the target element already has a StorageProtectionSetting associated via ElementProtectionSettingData, then it modifies the properties of the existing instance of

StorageProtectionSetting, as shown in Figure 98. After the retention period has expired and every protection



configuration has been released, the StorageProtectionSetting instance will not automatically be removed by the instrumentation. However a state change indication will be sent to the management application so that it may remove the instance by using the DeleteInstance operation if needed.

Table 350 shows properties this profile defines for the StorageProtectionCapabilities class, which indicates the capability of the element protection feature of the associated StorageProtectionService, including the granularity of the retention period.

Table 350: Properties for StorageProtectionCapabilities

Property	Flags	Type	Descriptions & Notes
ProtectionTimeGranularity		uint16	Granularity for the time period of StorageProtectionSetting.RemainingProtectionTime. Possible values are: 0 (Unknown), 1 (Other), 2 (Second), 3 (Minute), 4 (Hour), 5 (Day)
SupportedStorageElementFeatures		uint16[]	Enumeration indicating which storage elements can be protected. Possible values: 1 - StorageVolume Protection 2 - LogicalDisk Protection
SupportedSynchronousActions		uint16[]	Methods that will not create a job. Possible values: 1 - Storage Element Protection
SupportedAsynchronousActions		uint16[]	Methods that will create a job. Possible values: 1 - Storage Element Protection

This profile also defines a new Setting class, `StorageProtectionSetting`, which contains the protection-related properties for a particular `StorageVolume` or `LogicalDisk` storage element, shown in Table 351. This class is associated to a storage element instance via the `ElementProtectionSettingData` association. A client can retrieve the protection-related configurations and statuses of a `StorageVolume` or `LogicalDisk` by traversing the `ElementProtectionSettingData` association if it exists. If that association is not found, no protection management is applied for the `StorageVolume` or `LogicalDisk`.

Table 351: Properties for `StorageProtectionSetting`

Property	Flags	Type	Descriptions & Notes
ProtectionControlled		boolean	Whether the storage element is under protection control or not. If this property is FALSE that indicates the storage device has protection feature or used to has but currently the service has been withdrawn or not available to obtain protection attributes by some accident.
Access		uint16	Read and write accessibility of the storage element. 1: Read/Write Enabled 2: Read Only 3: Write Once 4: Read/Write Disabled While it is not possible to use <code>Protect()</code> to transition to "Write Once", it's still needed for correct reporting of status
InquiryProtection		Uint16[]	Protected responses for SCSI inquiry commands. 1: No SCSI Inquiry Protection 2: Inquiry Disabled 3: Zero Capacity Returned This property is utilized in the protection of a <code>StorageVolume</code> and it is optional to implement
DenyAsCopyTarget		boolean	Whether the storage element can be specified as a copy target or not. If this property is TRUE then this storage element will not be selectable as a target of copy pair
LUNMappingConfigurable		boolean	Whether LU assignment to the storage element is configurable or not. This property is utilized in the protection of a <code>StorageVolume</code> and is optional to implement
ProtectExpirationSpecified		uint16	Duration type of the storage element protection. 1: None 2: Limited Expiration 3: Permanent

Table 351: Properties for StorageProtectionSetting

Property	Flags	Type	Descriptions & Notes
RemainingProtectionTime		datetime	Amount of remaining time before a management application can change the access permission.

26.1.5 Access permission

The overall state of the StorageVolume or LogicalDisk protection is indicated by the combination of several properties. Table 352, Table 353, Table 354, Table 355, and Table 356 show the possible values of each property listed in Table 351. These tables apply to properties in the StorageProtectionSetting class.

Table 352: Values for ProtectionControlled

Value	Description
TRUE	Storage element is under protection control.
FALSE	Storage element is NOT under protection control.

Table 353: Values for Access

Value	Description
0 (Unknown)	Accessibility status is unknown.
1 (Read/Write Enabled)	Both read and write commands are allowed.
2 (Read Only)	Read command is allowed; write command is prohibited.
3 (Write Once)	Read command is allowed; overwrite command is prohibited.
4 (Read/Write Disabled)	Both read and write commands are prohibited.

Table 354: Values for InquiryProtection

Value	Description
0 (Unknown)	Status is unknown
1 (No SCSI Inquiry Protection)	Protection method by the SCSI inquiry commands is not performed
2 (Inquiry Disabled)	All SCSI inquiry commands are rejected
3 (Zero Capacity Returned)	Size 0 is returned as a reply of SCSI read capacity command

Table 355: Values for DenyAsCopyTarget

Value	Description
TRUE	Storage element can not be specified as a copy target
FALSE	Storage element can be specified as a copy target

Table 356: Values for LUNMappingConfigurable

Value	Description
TRUE	LU assignment to the storage volume is configurable
FALSE	LU assignment to the storage volume is not configurable

26.1.6 Retention period

The Retention period (the amount of time that the settings are to remain locked) is also indicated by the combination of several properties. Table 357 and Table 358 show the meaning of each property value. These tables apply to properties in the StorageProtectionSetting class.

Table 357: Values for ProtectExpirationSpecified

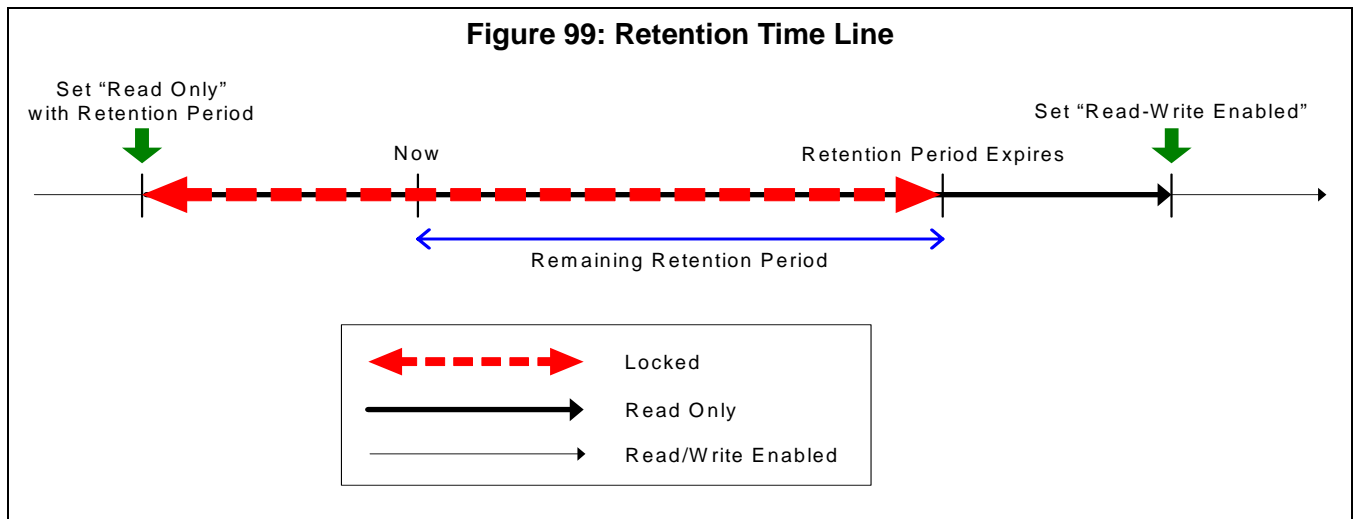
Value	Description
0 (Unknown)	Status is unknown.
1 (None)	The protection duration is not specified.
2 (Limited Expiration)	The protection expires after the time period
3 (Permanent)	The protection is permanent

Table 358: Values for RemainingProtectionTime

Value	Description
datetime	Amount of remaining time before a management application can change the access permission. It is a dynamic value which keeps decreasing by the time progress until it reaches the datetime equivalent of 0. The value will be decreased by the time period indicated by the StorageProtectionCapabilities.ProtectionTimeGranularity property

There are two ways to designate the duration of access permission, shown in Figure 99:

- Expiration Date - Defines a future date/time when access permission may be modified.
- Remaining Retention Period - Defines the remaining length of time for access permission.



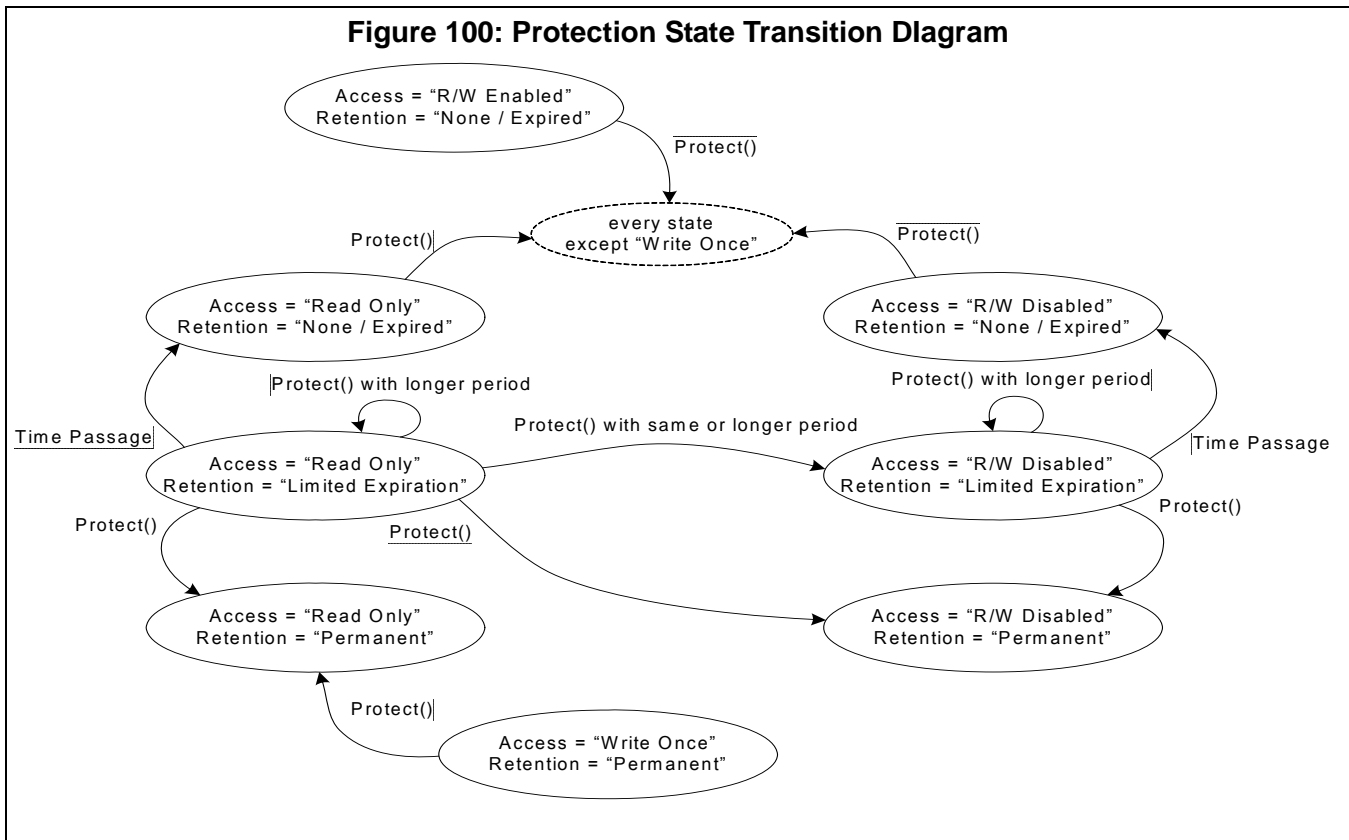
The use of an *Expiration Date* requires a reference to an agreed-upon reference clock. Without a trusted external date/time reference, the retention period will be open to spoofing, conflicts between individual component clocks (e.g., server and storage) and time zones issues. The inevitable nuances of individual implementations may require variations in the client application.

The use of *Remaining Retention Period* does not require a reference clock. There is no question of interpretation of whether or when the retention period will expire - it is either zero (expired) or not. The implementation is the responsibility of the provider and is hidden from the client. Providers may implement the retention function that works best for that provider, while remaining interoperable.

26.1.7 Protection State Transition

Figure 100 shows storage element protection state transition. When the retention period is not specified or expired, the storage element may transition to any state except *Write Once* permission by using the Protect method. Once a retention period is specified to a storage element, it may transition to a more restricted state only via the Protect method. It may transition to the other states only when the retention period has expired. Generally a storage element starts with a protection state of "Access = Read/Write Enabled, Retention = None/Expired" and Protect is used to set the protection to be more restrictive. If the storage element is write-once media such as a CD-ROM it will have a protection state of "Access = Write Once, Retention = Permanent".

Figure 100: Protection State Transition Diagram

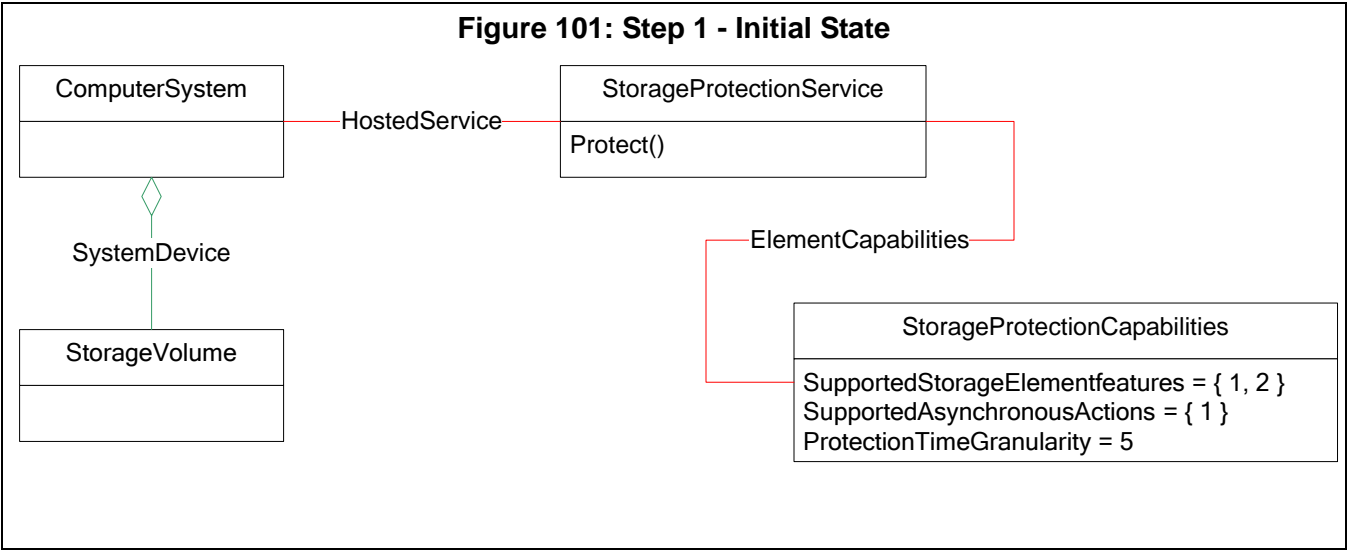


26.1.8 Sample Usage Scenario

Figure 101, Figure 102, Figure 103, Figure 104, and Figure 105 show the progression of a typical usage scenario for StorageVolume protection.

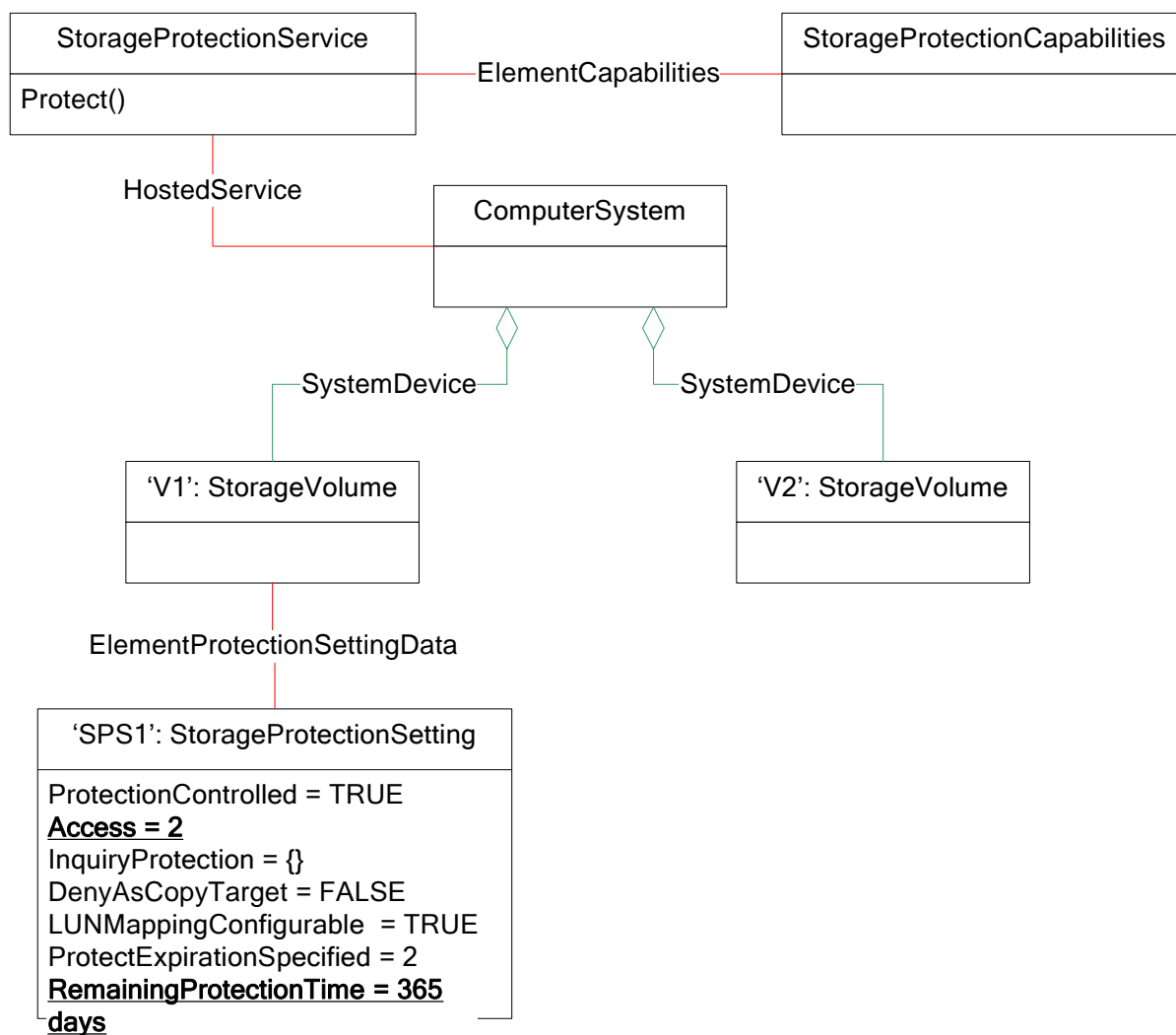
26.1.8.1 Step 1: StorageVolume not protected

Figure 101 shows the initial state of a StorageVolume that does not have protection enabled yet. In this situation, no instance of StorageProtectionSetting exists. However, it shows that the instrumentation has the capability to support the setting of the element protection properties because the StorageProtectionCapabilities SupportedStorageElementFeatures property includes the value 1 (StorageVolume Protection) and the SupportedAsynchronousActions property includes the value 1 (Storage Element Protection). The StorageProtectionCapabilities instance also has a value of 5 (Day) for the ProtectionTimeGranularity property which indicates the retention period specified on this device will be decreased by the granularity of a day.



26.1.8.2 Step 2: Volume Set to Read-only

In Figure 102, the **StorageVolume** is set to Read-only permission for a specific period of time. In this example, there are two **StorageVolumes**, 'V1' and 'V2'. By using the **Protect()** method of **StorageProtectionService**, volume 'V1' is set to Read-only access permission and a 365-day retention period. This operation creates new instance of **StorageProtectionSetting** ('SPS1') and associates it with the target **StorageVolume** 'V1'. After the **Protect** method completes, the **Access** property is now set to the value 2 (Read Only), and the **RemainingProtectionTime** is set to the value of 365 days.

Figure 102: Step 2 - Volume Set to Read-only

26.1.8.3 Step 3: Second Volume Set to Read-only

Figure 103 shows Set Read-only permission to another StorageVolume 'V2' after some amount of time.

After 30 days, the client decides to protect StorageVolume 'V2' by setting it to Read-only with a retention time of 365 days, same as 'V1'. A new instance of StorageProtectionSetting is created by the instrumentation to the target StorageVolume 'V2'. A single StorageProtectionSetting instance will not be shared because it has a different RemainingProtectionTime although both are configured with the same access permission.

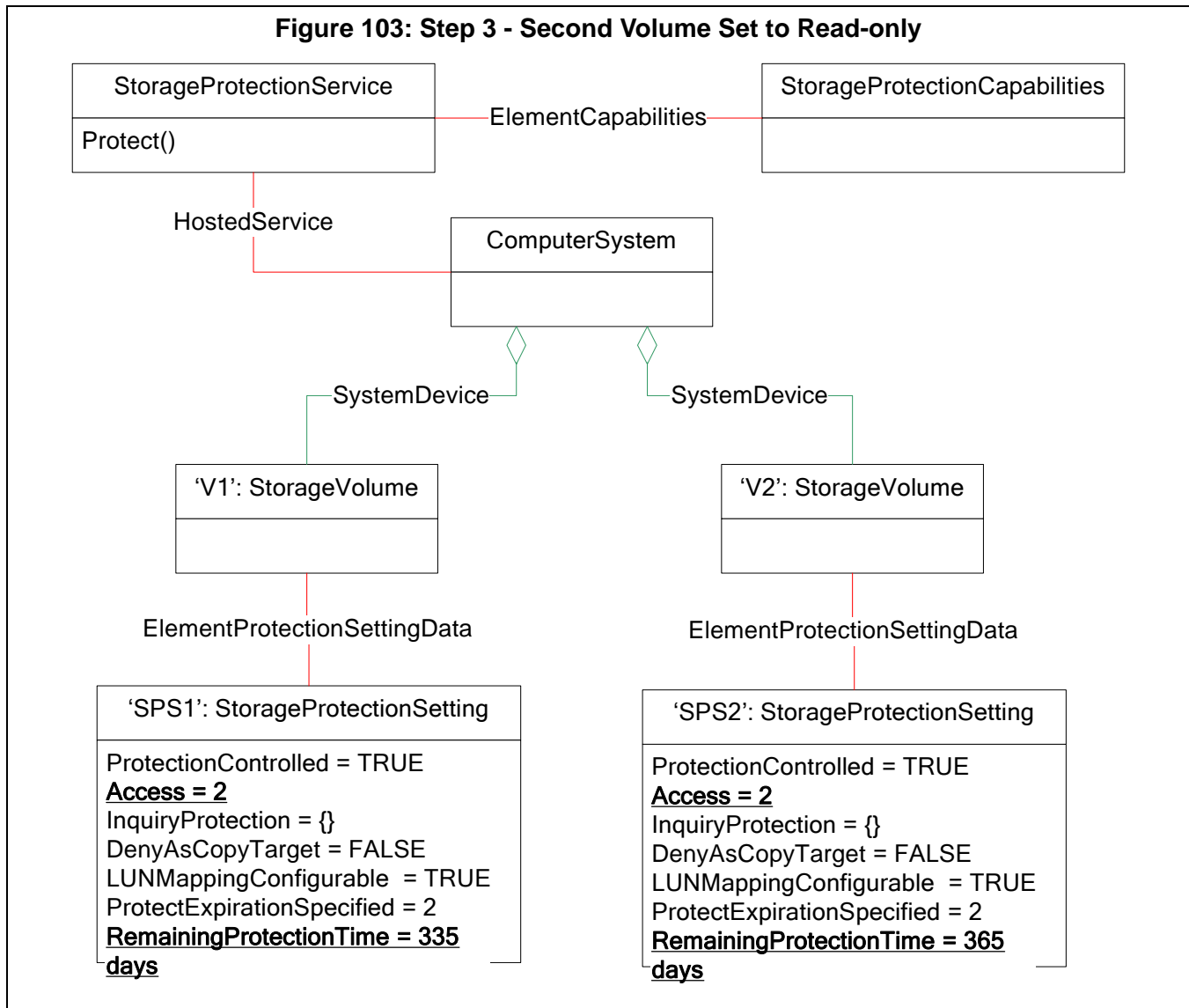
Figure 103: Step 3 - Second Volume Set to Read-only**26.1.8.4 Step 4: Volume Set to Read/Write Disabled**

Figure 104 shows access permission of StorageVolume 'V1' changed to Read/Write Disabled.

Within the retention period, the access permission may not be changed except to be made more restricted. Because StorageVolume 'V1' was set to Read-only permission, it is possible to modify it to Read/Write Disabled permission within its retention period because this setting is more restrictive than Read-only.

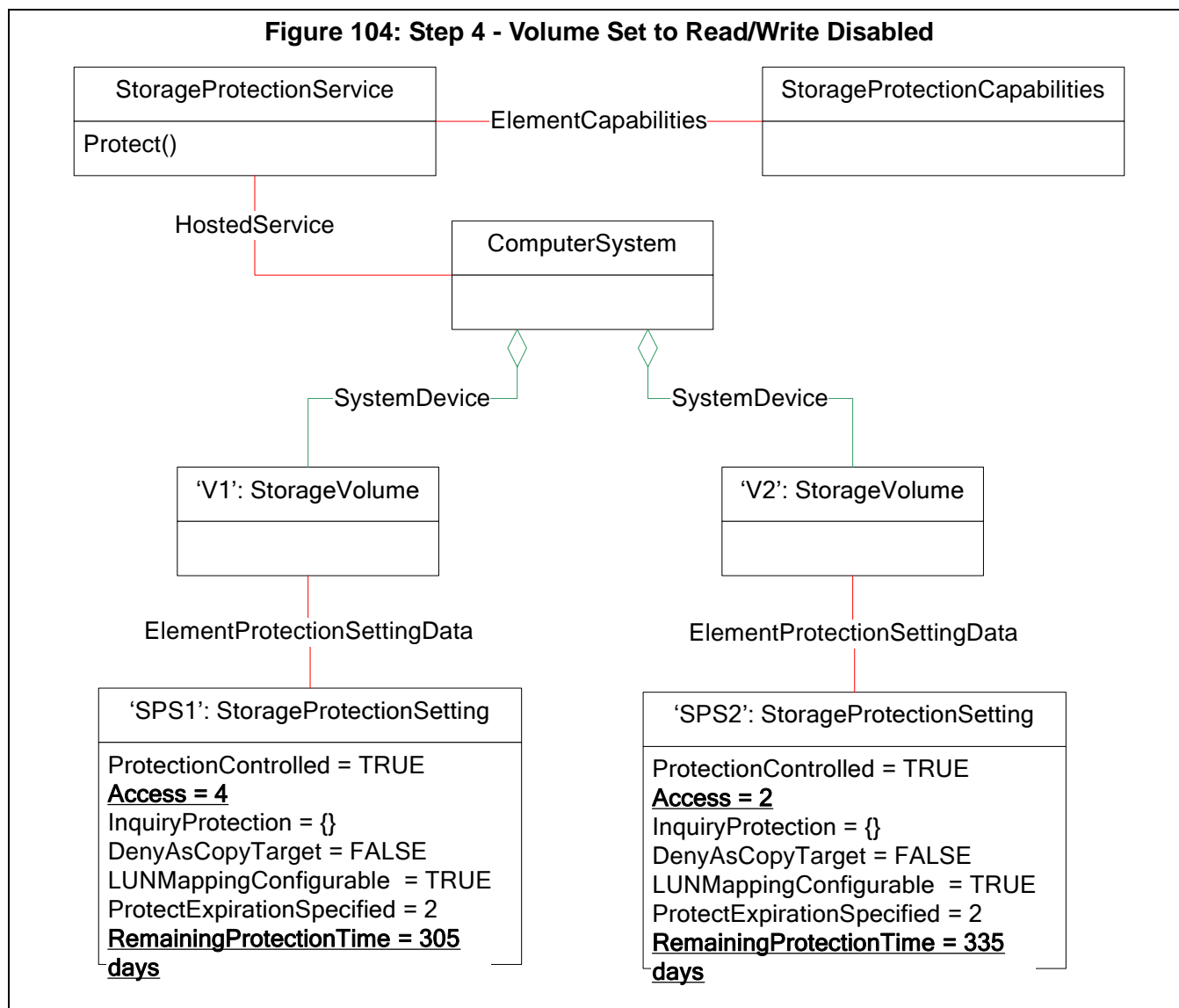
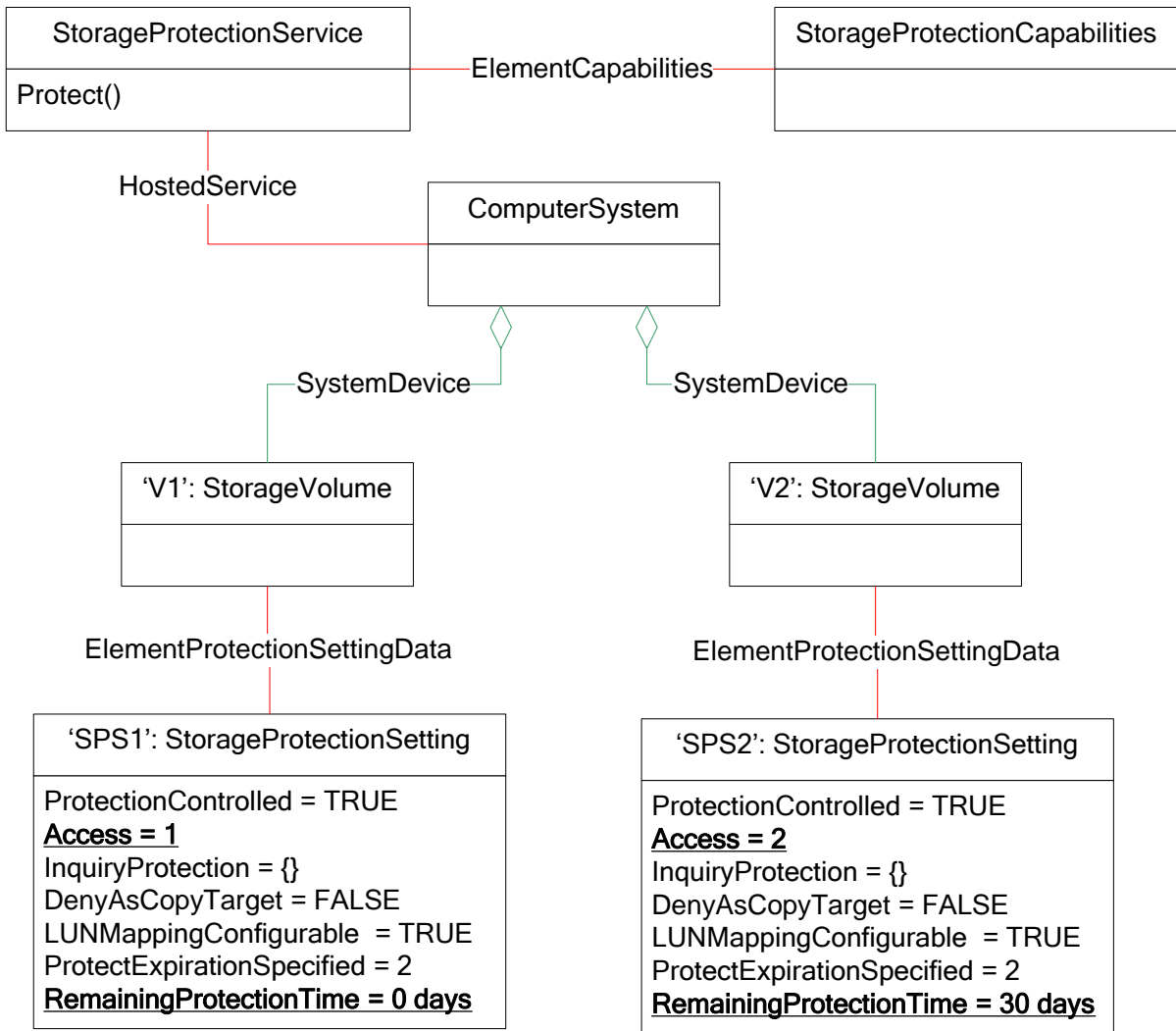
Figure 104: Step 4 - Volume Set to Read/Write Disabled**26.1.8.5 Step 5: Volume Access Change**

Figure 105 shows change of access permission of StorageVolume 'V1' to "Read/Write Enabled" after expiration.

After the passage of the specified time, the retention period of StorageVolume will expire. Therefore, its access permission can be modified to any level. The StorageProtectionSetting instance is not automatically deleted when the retention period has expired. The StorageVolume maintains its access permission configuration.

Figure 105: Step 5 Volume Access Changed



26.2 Health and Fault Management Consideration

Not defined in this standard

26.3 Cascading Considerations

Not applicable

26.4 Supported Profiles, Subprofiles, and Packages

Not applicable.

26.5 Methods of the Profile

26.5.1 Protect

This method, defined in Table 359, is found in the StorageProtectionService. It configures the protection attributes of StorageVolumes and LogicalDisks, which prevents them from being modified for a specific period of time. Values specified for this method shall be set as properties of the StorageProtectionSetting instance that is associated to the specified StorageVolume or LogicalDisk. This method can be used to extend the retention period, but not decrease it. The instrumentation shall always create a new instance of StorageProtectionSetting when protection is first applied, but it shall reuse the existing setting when modifying the protection setting.

Table 359: Methods of the Storage Element Protection Profile

Method: Protect			
Return Values:			
Value		Description	
0: Success		Method completed with no error.	
1: Not Supported		Method is not supported	
2: Unspecified Error		Unspecified error	
3: Timeout		Timeout happened during processing	
4: Failed		Method failed.	
5: Invalid Parameter		Specified parameter is not allowed	
6: Invalid State Transition		Specified access permission or retention period is not allowed in the current status.	
4096: Method parameters checked - job started		A Job was started	
Errors:			
Not defined in this standard			
Parameters:			
Qualifiers	Name	Type	Description/Values
OUT	Job	CIM_Job REF	Reference to the job created, if any
IN	Element	CIM_StorageExtent REF	StorageVolume or LogicalDisk to be configured.
IN	ElementType	uint16	The type of element being protected. 1: StorageVolume 2: LogicalDisk

Table 359: Methods of the Storage Element Protection Profile

Method: Protect			
IN	Access	uint16	Read and write accessibility of the storage element. 1: Read/Write Enabled 2: Read Only 4: Read/Write Disabled Note that it is not possible to transition to "3: Write Once" from other state
IN	InquiryProtection	uint16[]	The inquiry protection method for SCSI inquiry commands. 1: No SCSI Inquiry Protection 2: Inquiry Disabled 3: Zero Capacity Returned This may be specified when protecting a StorageVolume
IN	DenyAsCopyTarget	boolean	Whether the storage element can be specified as a copy target or not. If this property is TRUE then the storage element will not be selectable as a target of copy pair
IN	LUNMappingConfigure	boolean	Whether LU assignment to the StorageVolume is configurable or not. This may be specified when protecting a StorageVolume
IN	ProtectExpirationType	uint16	Duration type of the storage element protection. 1: None 2: Limited Expiration 3: Permanent
IN	TimePeriod	datetime	Amount of remaining time before a management application can change the access permission

26.6 Client Considerations and Recipes

26.6.1 Start Volume Protection

In this use case, a StorageVolume is used to store business data that needs to be protected from overwriting; there is a regulation that this kind of data should be held for three years. Therefore, a management application requests the instrumentation to set the StorageVolume for Read-only access permission and a three-year retention period.

```
// DESCRIPTION
//
```

```

// Start StorageVolume protection
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. Reference to the CIM_StorageVolume to be protected
//    has been found and the object path value is stored in
//    $StorageVolume->
// 2. Reference to the SNIA_StorageProtectionService
//    has been found and the object path value is stored in
//    $StorageProtectionService->
// 3. The SNIA_StorageProtectionCapabilities for this service
//    has been found and the instance value is stored in
//    $StorageProtectionCapabilities
// 4. Variable #ThreeYearsValue and #TwoYearsValue are the
//    type of datetime and have the value of a three and two year
//    time period, respectively

// Check for the capability
#SupportedFeatures[] =
    $StorageProtectionCapabilities.SupportedStorageElementFeatures

if (contains(1, #SupportedFeatures) == false) {
    <ERROR! StorageVolume protection feature is not supported>
}

// Invoke the protection method.
%InputArguments["Element"]           = $StorageVolume->
%InputArguments["ElementType"]       = 1// StorageVolume
%InputArguments["Access"]            = 2// Read Only
%InputArguments["ProtectExpirationType"] = 2// Limited Expiration
%InputArguments["TimePeriod"]        = #ThreeYearsValue// 3 years

#ReturnCode = InvokeMethod(
    $StorageProtectionService->,
    "Protect",
    %InputArguments,
    %OutputArguments )

if (#ReturnCode != 0) {
    <ERROR! CIM_StorageProtectionSetting has not been created.>
}

// Verify the protection setting.
$StorageProtectionSettings->[] = Associators(

```

```

$StorageVolume->,
"SNIA_ElementProtectionSettingData",
"SNIA_StorageProtectionSetting",
"Dependent",
"Antecedent",
false, false, NULL )

for #i in $StorageProtectionSettings->[] { // should be only one item.
  if( $StorageProtectionSetting->[#i].ProtectionControlled == false ) {
    <ERROR! CIM_StorageVolume is not under protection controlled.>
  }

  if( $StorageProtectionSetting->[#i].Access == 2 &&
      $StorageProtectionSetting->[#i].ProtectExpirationSpecified == 2 &&
      $StorageProtectionSetting->[#i].RemainingProtectionTime > #TwoYearsValue
      &&
      $StorageProtectionSetting->[#i].RemainingProtectionTime <=
        #ThreeYearsValue )
    {
      <EXIT: StorageVolume Protection configuration successful>
    }
  }
}

// if we get to this point, it was not set
<ERROR! StorageProtectionSetting has not been created.>

// end of the recipe

```

26.6.2 Extend the Retention Period

In this use case, a regulation has changed and the business data now need to be held for five years. A management application retrieves the value of the RemainingProtectionTime property of StorageProtectionSetting instance, which is associated to the target StorageVolume, and calculates the new value by adding two years to it. Then the application uses the StorageProtectionService.Protect() method to configure a new retention period.

```

// DESCRIPTION
//
// Extend the retention period
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. Reference to the CIM_StorageVolume to be protected
//    has been found and the object path value is stored in
//    $StorageVolume->
// 2. Reference to the SNIA_StorageProtectionService
//    has been found and the object path value is stored in
//    $StorageProtectionService->

```


Storage Element Protection SubProfile

```
// 3. The SNIA_StorageProtectionCapabilities for this service
//    has been found and the instance value is stored in
//    $StorageProtectionCapabilities
// 4. Variable #TwoYearsValue is the type of datetime and
//    has two year period of value.

// Check for the capability
#SupportedFeatures[] =
    $StorageProtectionCapabilities.SupportedStorageElementFeatures

if (contains(1, #SupportedFeatures) == false) {
    <ERROR! StorageVolume protection feature is not supported>
}

// Get current value for remaining protection time.
$StorageProtectionSettings[] = Associators(
    $StorageVolume->,
    "SNIA_ElementProtectionSettingData",
    "SNIA_StorageProtectionSetting",
    "Dependent",
    "Antecedent",
    false, false, NULL )

for #i in $StorageProtectionSettings[] { // should be only one item.
    if( $StorageProtectionSetting[#i].ProtectionControlled == false ) {
        <ERROR! CIM_StorageVolume is not under protection controlled.>
    }
    #RemainingProtectionTime =
        $StorageProtectionSetting[#i].RemainingProtectionTime
}

// Invoke the protection method.
// Set the time to protect the StorageVolume to the current time remaining + 2 more
//    years
%InputArguments["Element"]           = $StorageVolume->
%InputArguments["ElementType"]       = 1// StorageVolume
%InputArguments["Access"]            = 2// Read Only
%InputArguments["ProtectExpirationType"] = 2// Limited Expiration
%InputArguments["TimePeriod"]        = #RemainingProtectionTime + #TwoYearsValue

#ReturnCode = InvokeMethod(
    $StorageProtectionService->,
    "Protect",
    %InputArguments,
```

Storage Element Protection SubProfile

```
%OutputArguments )

if (#ReturnCode != 0) {
    <ERROR! SNIA_StorageProtectionSetting has not been created.>
}

// Verify the protection setting using prior found instance
for #i in $StorageProtectionSettings->[] {
    if( $StorageProtectionSetting->[#i].ProtectionControlled == false ) {
        <ERROR! CIM_StorageVolume is not under protection controlled.>
    }

    if( $StorageProtectionSetting->[#i].Access == 2 &&
        $StorageProtectionSetting->[#i].ProtectExpirationSpecified == 2 &&
        $StorageProtectionSetting->[#i].RemainingProtectionTime
            > #TwoYearsValue &&
        $StorageProtectionSetting->[#i].RemainingProtectionTime
            <= #RemainingProtectionTime + #TwoYearsValue )
    {
        <EXIT: StorageVolume Protection configuration successful>
    }
}

// if we get to this point, it was not set
<ERROR! SNIA_StorageProtectionSetting was not created>

// end of the recipe
```

26.7 Registered Name and Version

Storage Element Protection version 1.2.0

26.8 CIM Elements

Table 360: CIM Elements for Storage Element Protection

Element Name	Requirement	Description
SNIA_StorageProtectionSetting (26.8.1)	Mandatory	SNIA_StorageProtectionSetting class holds properties for the protection-related configuration and statuses of a storage element. It is associated to the StorageVolume or LogicalDisk class by SNIA_ElementProtectionSettingData. A management application can retrieve the protection-related information by traversing the ElementProtectionSettingData association. If is not found, it indicates no protection management is applied for the storage element
SNIA_ElementProtectionSettingData (26.8.2)	Mandatory	SNIA_ElementProtectionSettingData represents the association between the storage element to be protected and applicable protection setting.
SNIA_StorageProtectionCapabilities (26.8.3)	Mandatory	
SNIA_StorageProtectionService (26.8.4)	Mandatory	

26.8.1 SNIA_StorageProtectionSetting

Created By: Extrinsic: Protect

Modified By: Extrinsic: Protect

Deleted By: DeleteInstance

Class Mandatory: Mandatory

Table 361 describes class SNIA_StorageProtectionSetting.

Table 361: SMI Referenced Properties/Methods for SNIA_StorageProtectionSetting

Properties	Flags	Requirement	Description & Notes
ProtectionControlled		Optional	Whether the storage element is under protection control or not.
Access		Mandatory	Read and write accessibility of the StorageVolume. 1: Read/Write Enabled 2: Read Only 3: Write Once 4: Read/Write Disabled
InquiryProtection		Conditional	Conditional requirement: Storage Volumes used as storage elements. StorageVolume protection method for SCSI inquiry commands. 1: No SCSI Inquiry Protection 2: Inquiry Disabled 3: Zero Capacity Returned

Table 361: SMI Referenced Properties/Methods for SNIA_StorageProtectionSetting

Properties	Flags	Requirement	Description & Notes
DenyAsCopyTarget		Optional	Whether the storage element can be specified as a copy target or not.
LUNMappingConfigurable		Conditional	Conditional requirement: Storage Volumes used as storage elements. Whether LU assignment to the StorageVolume is configurable or not.
ProtectionExpirationSpecified		Mandatory	Duration type of the storage element protection. 1: None 2: Limited Expiration 3: Permanent
RemainingProtectionTime		Mandatory	Amount of remaining time before a management application can change the access permission.

26.8.2 SNIA_ElementProtectionSettingData

Created By: Extrinsic: Protect

Modified By: Static

Deleted By: External

Class Mandatory: Mandatory

Table 362 describes class SNIA_ElementProtectionSettingData.

Table 362: SMI Referenced Properties/Methods for SNIA_ElementProtectionSettingData

Properties	Flags	Requirement	Description & Notes
ManagedElement		Mandatory	The storage element to be protected.
SettingData		Mandatory	The protection setting and status of the storage element

26.8.3 SNIA_StorageProtectionCapabilities

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 363 describes class SNIA_StorageProtectionCapabilities.

Table 363: SMI Referenced Properties/Methods for SNIA_StorageProtectionCapabilities

Properties	Flags	Requirement	Description & Notes
ProtectionTimeGranularity		Mandatory	Granularity for the time period of StorageProtectionSetting.RemainingProtectionTime.
SupportedStorageElementFeatures		Mandatory	Value for storage element protection. 1 (StorageVolume Protection), 2 (LogicalDisk protection)
SupportedSynchronousActions		Mandatory	Value for storage element protection. 1 (Storage Element Protection)
SupportedAsynchronousActions		Mandatory	Value for element protection. 1 (Storage Element Protection)

26.8.4 SNIA_StorageProtectionService

Created By: Static

Modified By: Static

Deleted By: Static

Class Mandatory:

Table 364 describes class SNIA_StorageProtectionService.

Table 364: SMI Referenced Properties/Methods for SNIA_StorageProtectionService

Properties	Flags	Requirement	Description & Notes
Protect()		Mandatory	Configures the protection attributes of the storage element and prevent modification for a specific period of time. Values specified for this method will be set as properties of StorageProtectionSetting instance which is associated to the specified storage element. This method can be used to extend the retention period, but not for decreasing it.

EXPERIMENTAL
