SNIA
Storage Networking Industry Association

Storage Management Initiative Specification

1.1.1 Revision 1

SNIA Technical Position

5 June, 2007

Errata/Change Log

June 5, 2007

No errata have been identified for 1.1.1.

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

Suggestions for revisions should be directed to tcmd@snia.org.

Typographical Conventions

Deprecated Material

Sections identified as “Deprecated” contain material that is 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. Providers shall implement the deprecated elements in order to achieve backwards compatibility. Clients may use the deprecated elements and are directed to instead use the elements that are thus favored.

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. A sample of the typographical convention for deprecated content is included below:

DEPRECATED

Deprecated material appears here.

DEPRECATED

Experimental Material

Some of the content considered for inclusion in SMI-S 1.1.1 has yet to receive sufficient review to satisfy the adoption requirements set forth by the SMI committee within the SNIA. This content is presented here as an aid to implementers who are interested in likely future developments within the SMI specification. The content marked experimental may change as implementation experience is gained. There is a high likelihood that it will be included in an upcoming revision of the specification. Until that time, it is purely informational, and is clearly marked within the text.

A sample of the typographical convention for experimental content is included here:

EXPERIMENTAL

Experimental content appears here.

EXPERIMENTAL

Contents

List of Tables

List of Figures

Foreword

This foreword is not a normative part of the Storage Management Initiative Specification.

This Technical Specification defines a method for the interoperable management of a heterogeneous Storage Area Network (SAN), describes the information available to a WBEM Client from an SMI-S compliant CIM Server and an object-oriented, XML-based, messaging-based interface designed to support the specific requirements of managing devices in and through Storage Area Networks (SANs).

With any technical document there may arise questions of interpretation as new products are implemented. SNIA has established procedures to issue technical opinions concerning the standards developed by SNIA. These procedures may result in updates to the specification being published by SNIA.

These updates, while reflecting the opinion of the Technical Committee that developed the standard, are intended solely as supplementary information to other users of the standard. This standard as approved through the publication and voting procedures of the Storage Networking Industry Association, is not altered by these bulletins. Any subsequent revision to this standard may or may not reflect the contents of these Technical Information Bulletins.

This standard contains two annexes. Annex A is informative and is not considered part of this standard. Annex B is normative and is part of this standard.

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

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the Storage Networking Industry Association, 500 Sansome Street, Suite #504, San Francisco, CA 94111 USA.

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

Organization Represented	*Name of Representative*
Brocade Communications Systems	John Crandall
EMC Corporation	Kamesh Aiyer
	George Ericson
	Steve Terwilliger
Hewlett-Packard	Steve Peters
Hitachi Data Systems	Steve Quinn
IBM	Duane Baldwin
	Jack Gelb
	Mike Walker
iStor Networks, Inc	Scott Baker
Network Appliance	Alan Yoder
Sun Microsystems	Mark Carlson
	Paul von Behren
Symantec	Joe Fasano
	Steve Hand
TeraCloud	Bill Pierce
WBEM Solutions	Jim Davis

Introduction

The SNIA Storage Management Initiative Specification standard (SMI-S) is divided into a number of clauses and annexes:

Clause 1 defines the scope of this standard and places it in context of other standards and standards projects.

Clause 2 specifies definitions, symbols, abbreviations, and document conventions.

Clause 3 describes the underlying concepts of the interface, including its anticipated usage model and business rationale.

Clause 4 shows typographical conventions used in this document to indicate deprecated and experimental material. The clause also defines the terms *deprecated* and *experimental* as used in SMI-S.

Clause 5 enumerates the normative references that apply to this standard.

Clause 6 identifies high-level requirements and capabilities addressed by this standard.

Clause 7 defines the transport layer and reference model employed in the standard.

Clause 8 provides an overview of the object model underlying the standard.

Clause 9 defines the object model underlying the standard.

Clause 10 defines service discovery in the context of this standard.

Clause 11 defines the roles for the various entities of the management system defined by the standard.

Clause 12 defines the installation and upgrade process defined by the standard.

Annex A specifies a method of mapping CIM objects to SNMP MIB structures (informative).

Annex B establishes the declaration of compliance with the SNIA SMI Specification (normative).

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: Definitions, Symbols, Abbreviations, and Conventions

2.1 Definitions

2.1.1 Address masking

Address masking is a function of a host I/O controller (device driver) that filters access to certain storage resources on the SAN. It puts the responsibility of segregating I/O paths on the individual server system in the SAN and requires coordination of all servers to avoid access collisions. Also called Host-based LUN Masking.

2.1.2 Addressable Unit

Storage addressable unit (e.g., LUN, Virtual Disk, Logical Disk, Logical Volume, Volume Set).

2.1.3 Agent

An Object Manager that includes the provider service for a limited set of resources.

An Agent may be embedded or hosted and can be an aggregator for multiple devices.

2.1.4 Aggregation

A strong form of an association. For example, the containment relationship between a system and the components that make up the system can be called an aggregation. An aggregation is expressed as a Qualifier on the association class. Aggregation often implies, but does not require, that the aggregated objects have mutual dependencies.

2.1.5 ATM

Acronym for Asynchronous Transfer Mode.

2.1.6 Attributes

A collection of tags and values describing the characteristics of a service.

2.1.7 Attribute Reply (AttrRply)

A reply to an Attribute Request. (optional)

2.1.8 Attribute Request (AttrRqst)

A request for attributes of a given type of service or attributes of a given service. (optional)

2.1.9 Cardinality

The number of values that may apply to an attribute for a given entity. Refer to *UML* Standards. See Table 92.

2.1.10 CIM

Acronym for Common Information Model. An object-oriented description of the entities and relationships in a business' management environment maintained by the Distributed Management Task Force. CIM is divided into a Core Model and Common Models. The Core Model addresses high-level concepts (such as systems and devices), as well as fundamental relationships (such as dependencies). The Common Models describe specific problem domains such as computer system, network, user or device management. The Common Models are subclasses of the Core Model and may also be subclasses of each other.

2.1.11 CIMOM

Acronym for Common Information Model Object Manager.

2.1.12 Client

A process that issues requests for service. Formulating and issuing requests may involve multiple client processes distributed over one or more computer systems. The collection of client processes involved in formulating and issuing requests is known as a *consumer*.

2.1.13 Completion Semantics

Specifies how a method notifies its caller that its operations have completed. To this end, notification of completion is accomplished in either of two ways:

Asynchronous notification: Upon return of the method, its operations may not have yet completed. The caller is then required to employ some other mechanism to determine when the operations complete. Events, callbacks, and polling are examples of mechanisms available to the caller in this regard.

Synchronous notification: The thread calling the method blocks until the method's operations succeed or fail.

Completion semantics refer to the operations executed by the method, and *not* the method completion itself. For example, suppose we write a method to resync a split-mirror. We recognize that this could take an indeterminate amount of time, so we design a method, *resync()*, to spawn a task to manage the set of operations required for the resynchronization and then return to the caller. When the method, resync(), completes and returns to the caller, the resynchronization of the mirrors will [most likely] not have completed. So, the method has completed but its operations have not.

2.1.14 Consumer

A host, identified by HBA WWN or other identifier, that is allowed access to a storage addressable unit

2.1.15 Control Software

A body of software that provides common control and management for one or more disk arrays or tape arrays. Control software presents the arrays of disks or tapes it controls to its operating environment as one or more virtual disks or tapes. Control software may execute in a disk controller or intelligent host bus adapter, or in a host computer. When it executes in a disk controller or adapter, control software is often referred to as firmware.

2.1.16 Concurrency Control Protocol

A set of rules for identifying and resolving resource conflicts between multiple, non-cooperating clients. The three most common concurrency protocols are:

Lock ordering: Transactions are ordered according to the order of arrival of their operations at the resource(s).

Optimistic ordering: Transactions proceed until they are ready to commit, whereupon a check is made to see whether they have performed conflicting operations.

Timestamp ordering: Transactions are ordered according to the time they were initiated.

2.1.17 Cooperating Clients

A set of consumer processes that are aware of each other and are able to coordinate access to (and control of) resources among themselves

2.1.18 DA Advertisements (DAAdvert):

A solicited (unicast) or unsolicited (multicast) advertisement of Directory Agent availability.

2.1.19 Data Invariant

A data invariant is the name given to the consistency-state of shared data. A data invariant is always be TRUE. When the data invariant is violated, the invariant shall be protected via mutual exclusion. For example, suppose the user has a list of records and a record pointer, `i`, that is always set to point to the last record in the list. In this example, the invariant is *the record pointer always points to the last record*.

But observe what happens when the user appends a record to the list as follows:

(a) Add record to record[i].

(b)i += 1;

After (a) completes, but before (b) is invoked, *i* no longer points to the last record in the list. Now, suppose another thread comes along and attempts to read the last record in the list. In this case, the thread will get the penultimate record, not the last one – because *`i`* has not yet been updated. The solution to this problem is to serialize access to both operations using a lock or a semaphore.

BEGIN LOCK

(a) Add record to record[i].

(b)i += 1;

END LOCK

2.1.20 Device

A storage system that is addressable from the SAN.

2.1.21 DHCP

Acronym for dynamic host control protocol. An Internet protocol that allows nodes to dynamically acquire (“lease”) network addresses for periods of time rather than having to pre-configure them. DHCP greatly simplifies the administration of large networks, and networks in which nodes frequently join and depart.

2.1.22 Directory

A repository of information about objects that may be accessed via a Directory Service.

2.1.23 Directory Agent (DA):

In the context of SLP, a process that caches SLP service advertisements registered by Service Agents and forwards the service advertisements to User Agents on demand.

2.1.24 Discovery

Discovery provides information about what physical and logical SAN entities have been found within the management domain. Enough information is provided to support the creation of correct Topology maps. This information changes dynamically, as SAN entities are added, moved, or removed.

2.1.25 DLT

Acronym for Digital Linear Tape. A family of tape device and media technologies developed by Quantum Corporation.

2.1.26 DMTF

Distributed Management Task Force. An industry organization that develops management standards for computer system and enterprise environments. DMTF standards include WBEM, CIM, DMI, DEN and ARM. The DMTF has a web site at www.dmtf.org.

2.1.27 Enclosure

A box or cabinet.

2.1.28 Enumerate

This operation is used to enumerate subclasses, subclass names, instances and instance names in the target Namespace. If successful, the method returns zero or more requested elements that meet the required criteria.

2.1.29 Extent

A set of consecutively addressed FBA disk blocks that is allocated to consecutive addresses of a single file.

A set of consecutively located tracks on a CKD disk that is allocated to a single file.

A set of consecutively addressed disk blocks that is part of a single virtual disk-to-member disk array mapping. A single disk may be organized into multiple extents of different sizes, and may have multiple (possibly) non-adjacent extents that are part of the same virtual disk-to-member disk array mapping. This type of extent is sometimes called a logical disk.

2.1.30 Extrinsic Method

A method defined as part of CIM Schema. Extrinsic methods are invoked on a CIM Class (if static) or Instance (otherwise). An extrinsic method call is represented in XML by the <METHODCALL> element, and the response to that call represented by the <METHODRESPONSE> element. cf. Intrinsic Method

2.1.31 Fabric

Any interconnect between two or more Fibre Channel N_Ports, including point-to-point, loop, and Switched Fabric.

Switched Fabric: A fabric comprised of one or more Switches

2.1.32 FC-GS-3

Fibre Channel - Generic Services 3. Also abbreviated GS-3

2.1.33 FIP:

Acronym for Federal Information Processing Standard. Standards (and guidelines) produced by NIST for government-wide use in the specification and procurement of Federal computer systems.

2.1.34 Grammar

A formal definition of the syntactic structure of a language (see 2.1.99, "Syntax"), normally given in terms of production rules that specify the order of constituents and their sub-constituents in a sentence (a well-formed string in the language). Each rule has a left-hand side symbol naming a syntactic category (e.g., “noun-phrase” for a natural language grammar) and a right-hand side that is a sequence of zero or more symbols. Each symbol may be either a terminal symbol or a non-terminal symbol. A terminal symbol corresponds to one “lexeme” - a part of the sentence with no internal syntactic structure (e.g., an identifier or an operator in a computer language). A non-terminal symbol is the left-hand side of some rule.

2.1.35 GS-3

Refer to *FC-GS-3*

2.1.36 HBA

Host bus adapter, card that contains ports for host systems.

2.1.37 Host

A computer running an O/S.

2.1.38 HTTP

A request-reply protocol called the Hypertext Transfer Protocol, HTTP.

2.1.39 Hub

Interconnect element that supports a ring topology.

2.1.40 Inheritance Relationship

Refer to *UML* Standards.

2.1.41 Interconnect Element

Non terminal network elements (Switches, hubs, routers, directors).

2.1.42 Interface Definition Language (IDL)

A high-level declarative language that provides the syntax for interface declarations. Some examples of IDLs in common usage today are:

- DCE’s RPC IDL
- Microsoft’s DCOM IDL (based on the DCE IDL)
- OMG IDL (used to define the DOM XML interface)
- DMTF MOF (an IDL-derived specification).

2.1.43 Intrinsic Method

Operations made against a CIM server and a CIM Namespace independent of the implementation of the schema defined in the server. Examples of intrinsic methods in XML include the <IMETHODCALL> element, and the response to that call represented by the <IMETHODRESPONSE> element. cf. Extrinsic Method

2.1.44 Language-Binding

The association of a programming language (e.g., C++, Java, C) with an interface definition language. For example, OMG IDL supports many language bindings because it can be compiled into a variety of programming languages (C, C++, Java, ADA, COBOL, etc.). By contrast, Microsoft’s DCOM IDL only supports one language binding, C++. Similarly, Java IDL also supports only one language binding (Java).

Some IDLs do not support any [formal] language bindings. DMTF’s MOF, for example, is derived from OMG’s IDL but is used as a data modeling language more in the spirit of SQL than programmatic interfaces.

2.1.45 Lock Manager

Short name for Lock Management Server.

2.1.46 Logical Unit Number (LUN)

The SCSI identifier of a logical unit within a Target.

2.1.47 LTO

Acronym for Linear Tape Open.

2.1.48 LUN Mapping

The process of creating a disk resource and defining its external access paths, by configuring LUs (Logical Units) from the disk array logical disk volumes - either by grouping them as a single larger LU or by creating partitions. The Logical Unit Number (LUN) (2.1.46) is then mapped to an external ID descriptor (for example: a SCSI Port, Target ID and LU Number). An LU may be mapped for access from multiple ports and/or multiple target IDs, providing alternate paths for nonstop data availability.

LUN Mapping is a necessary task to be able to export the LUN to the Fabric/Server/etc. It can be done independently of any knowledge of the intended use of the LUN. Only LUNs that are exposed via a Port (2.1.62) are available for access.

2.1.49 LUN Masking

Process of configuring software in SAN nodes to determine which hosts have access to exported drives. LUN masking can be either server-based address masking or storage based port mapping. cf. Port Mapping

2.1.50 MAN

Acronym for Metropolitan Area Network. A network that connects nodes distributed over a metropolitan (city-wide) area as opposed to a local area (campus) or wide area (national or global). From a storage perspective, MANs are of interest because there are MANs over which block storage protocols (e.g., ESCON, Fibre Channel) can be carried natively, whereas most WANs that extend beyond a single metropolitan area do not currently support such protocols.

2.1.51 Marshalling

The set of operations by which a message is converted into a transfer syntax. In HTTP, requests and replies are marshaled into formatted ASCI-text strings.

2.1.52 Method

The name of [one or more] operation(s) performed by an instance of an object class. Methods are distinguished from operations as follows: A method is a name for one or more operations that may execute when the method is invoked. For example, when the method, `printSelf()`, is called, the operation of printing the state of the reference object is executed.

Synonyms are: Function, procedure, or subroutine. Usage of these terms should be deprecated.

In most models, a method is characterized by its name, return-type, parameters, completion semantics (asynchronous or synchronous), and side effects (e.g., event generation, message propagation, etc.).

Methods are *specified* in an IDL.

Methods are *declared* in source header files of a programming language (.h files, Java Interface files, etc.,).

Methods are *defined* (or *implemented*) in source implementation files (e.g.,.cpp,. java, class files).

Method specifications are language independent. Method declarations and implementations are, by construction, language dependent.

2.1.53 Monitoring

Monitoring provides management information about the current state of individual logical and physical SAN entities. This information changes dynamically, as SAN entities perform their functions, are serviced, experience errors, etc. Monitoring can only be done on SAN entities that are known via Discovery.

2.1.54 NAA

Acronym for Network Address Authority. A four bit identifier defined in FC-PH to denote a network address authority (i.e., an organization such as CCITT or IEEE that administers network addresses).

2.1.55 NDMP

Acronym for Network Data Management Protocol. A communications protocol that allows intelligent devices on which data is stored, robotic library devices, and backup applications to intercommunicate for the purpose of performing backups.

An open standard protocol for network-based backup of NAS devices. NDMP allows a network backup application to control the retrieval of data from, and backup of, a server without third-party software. The control and data transfer components of backup and restore are separated. NDMP is intended to support tape drives, but can be extended to address other devices and media in the future. The Network Data Management Task Force has a web site at HTTP://www.ndmp.org.

2.1.56 N_Port

Refer to Port. Node

A collection of Ports. A Fiber channel device with a group of ports.

An addressable entity connected to an I/O bus or network. Used primarily to refer to computers, storage devices, and storage subsystems. The component of a node that connects to the bus or network is a port.

2.1.57 Non-cooperating clients

A set of consumer processes that are independent of each other, compete for resources and execute independently of the other. User processes on a multi-user machine are non-cooperating clients with respect to the operating system.

2.1.58 Operation

An action executed within the body of a method. Operations are distinct from methods (see 2.1.52, "Method").

2.1.59 Out-of-Band

Transmission of management information for Fibre Channel components outside of the Fibre Channel network, typically over Ethernet.

2.1.60 PKI

Acronym for public key infrastructure. A framework established to issue, maintain, and revoke public key certificates accommodating a variety of security technologies.

2.1.61 Platform

Collection of Nodes.

2.1.62 Port

Connection point for links.

N_Port: A hardware entity that includes a Link_Control_Facility. It may act as an Originator, a Responder, or both.

N_Port identifier: A Fabric unique address identifier by which an N_Port is uniquely known. The identifier may be assigned by the Fabric during the initialization procedure. The identifier may also be assigned by other procedures not defined in FC-FS.

Port_Name: As defined in FC-FS.

2.1.63 Port Mapping

Function of a storage subsystem to define which hosts have access to exported drives. This configuration authorizes specified server HBA WWNs to access the secured LU while preventing other unauthorized servers/hosts from either seeing the secured LU or accessing the data contained on the secured LU. cf. LUN Masking

2.1.64 Protocol

A set of rules that define and constrain data, operations, or both. For example, xmlCIM uses XML as its transfer syntax, and HTTP as the request-reply protocol HTTP is layered over the TCP/IP network protocol.

2.1.65 Provider

A COM server that communicates with managed objects to access data and event notifications from a variety of sources, such as the system registry or an SNMP device. Providers forward this information to the CIM Object Manager for integration and interpretation.

class provider: A COM server that supplies class definitions. Class providers can support data retrieval, modification, deletion, enumeration, and query processing.

property provider: A type of provider that supports the retrieval and modification of the CIM properties.

2.1.66 Relationship

Refer to UML Standards. See Table 92.

2.1.67 Required Reference

Refer to UML Standards. See Table 92.

2.1.68 SA Advertisement (SAAdvert):

Information describing a service that consists of the Service Type, Service Access Point, lifetime, and Attributes.

2.1.69 SAN

Acronym for storage area network. (This is the normal usage in SNIA documents.)

Acronym for Server Area Network that connects one or more servers.

Acronym for System Area Network for an interconnected set of system elements.

A group of fabrics that have common leaf elements.

2.1.70 Scope

A set of services, typically making up a logical administrative group.

2.1.71 Semantics

The meaning or behavior associated with an entity. For example, we might say the semantics of the method, `resync_mirror()`, is encoded in the method name. By contrast, the semantics of the UNIX `ioctl()` method is encoded in the command parameter.

2.1.72 Server

A process that fields and/or dispatches requests. Honoring a request may involve more than one server process distributed over one or more computer systems. The collection of server processes that are involved in honoring a request are known as *service providers*.

2.1.73 Service Access Point

The network address and port number of a process offering a service.

2.1.74 Service Acknowledgement (SrvAck)

A reply to a SrvReg request.

2.1.75 Service Agent (SA)

In the context of SLP, this refers to a process working on behalf of one or more services to advertise the services in the network.

2.1.76 Service Agent Server (SAServer)

In the context of SLP, this refers to a process working on behalf of one or more Service Agents to listen on a particular port number for SLP service requests.

2.1.77 Service Deregister (SrvDereg)

A request to deregister a service or some attributes of a service. (optional)

2.1.78 Service Register (SrvReg)

A request to register a service or some attributes of a service.

2.1.79 Service Reply (SrvRply)

A reply to a Service Request.

2.1.80 Service Request (SrvRqst)

A request for a service on the network.

2.1.81 SES

Acronym for SCSI Enclosure Services. AT10 standard for management of environmental factors such as temperature, power, voltage, etc. (ANSI INCITS 305-1998 R2003)

2.1.82 Service Type

The class of a network service represented by a unique string, for example a namespace assigned by IANA (Internet Assigned Number Authority).

2.1.83 Service Type Reply (SrvTypeRply)

A reply to a Service Type Request. (optional)

2.1.84 Service Type Request (SrvTypeRqst)

A request for all types of service on the network. (optional)

2.1.85 Service Type Template

A formalized, computer-readable description of a Service Type.

2.1.86 Service URL

A Uniform Resource Locator for a service containing the service type name, network family, Service Access Point, and any other information needed to contact the service.

2.1.87 SLP

Acronym for Service Location Protocol.

2.1.88 SNIA

Acronym for Storage Networking Industry Association. An association of producers and consumers of storage networking products whose goal is to further storage networking technology and applications.

2.1.89 SNMP

Acronym for Simple Network Management Protocol. An IETF protocol for monitoring and managing systems and devices in a network. The data being monitored and managed is defined by a MIB. The functions supported by the protocol are the request and retrieval of data, the setting or writing of data, and traps that signal the occurrence of events.

2.1.90 SNMP Trap

A type of SNMP message used to signal that an event has occurred.

2.1.91 Soft Zone

A Zone consisting of Zone Members that are made visible to each other through Client Service requests. Typically, Soft Zones contain Zone Members that are visible to devices via Name Server exposure of Zone Members. The Fabric does not enforce a Soft Zone. Note that well known addresses are implicitly included in every Zone.

2.1.92 SPI

Acronym for SCSI Parallel Interface. The family of SCSI standards that define the characteristics of the parallel version of the SCSI interface. Several versions of SPI, known as SPI, SPI2, SPI3, etc., have been developed. Each version provides for greater performance and functionality than preceding ones.

2.1.93 SRM

Acronym for storage resource management. Management of physical and logical storage resources, including storage elements, storage devices, appliances, virtual devices, disk volume and file resources.

2.1.94 SSL

Acronym for Secure Sockets Layer. A suite of cryptographic algorithms, protocols and procedures used to provide security for communications used to access the world wide web. The characters “https:” at the front of a URL cause SSL to be used to enhance communications security. More recent versions of SSL are known as TLS (Transport Level Security) and are standardized by the Internet Engineering Task Force (IETF)

2.1.95 SSP

Acronym for Storage Service Provider.

2.1.96 Switch

Fibre channel interconnect element that supports a mesh topology.

2.1.97 Symmetric Virtualization Appliance

Synonym for an appliance that provides storage virtualization.Storage virtualization appliance is the preferred term.

2.1.98 Synchronous

A method that blocks the calling thread until all operations have completed or failed.

2.1.99 Syntax

(The structure of strings in some language. A language's syntax is described by a grammar. For example, the syntax of a binary number could be expressed as binary_number = bit [binary_number]

bit = “0” | “1”

Meaning that a binary number is a bit optionally followed by a binary number and a bit is a literal zero or one digit. The meaning of the language is given by its semantics.

2.1.100 TLS

Acronym for Transport Layer Security as defined in RFC 2246.

2.1.101 Transfer Syntax

The formal rules (i.e., the protocol) governing the format (or representation) of messages as they are transferred between clients and servers

2.1.102 UDP

Acronym for User Datagram Protocol. An Internet protocol that provides connectionless datagram delivery service to applications. Abbreviated UDP. UDP over IP adds the ability to address multiple endpoints within a single network node to IP.

2.1.103 UML Standards

Appendix D of the *Common Information Model (CIM) Specification, V2.0* (March 3, 1998).

Class - represented by a **rectangle**.

The class name either stands alone in the rectangle or is in the uppermost segment. If present, the segment below the segment containing the name contains the properties of the class. If present, a third region indicates the presence of methods.

Lines indicate:

- **Inheritance** relationships (blue lines with arrows) – Otherwise known as “is-a” relationships
- **Aggregation/component** relationships (green lines with a diamond shape at the “aggregating” end) - Otherwise known as “has-a” relationships
- **Dependency and other** relationships (red lines) – Some of which are “uses-a” relationships
- **Relationship Labels -** Inheritance relationships are not specifically labeled or named, while all other associations are named.
- **Cardinality** - the cardinalities of the references on both sides of an association are indicated by numeric values or an asterisk (*) at the endpoints of the association

The following cardinalities are typically used in the CIM Schema:

0..1 - Indicates an optional single-valued reference

1 - Indicates a **required**, single-valued reference

1..n or 1..* - Indicates either a single or multi-valued reference, that is **required***, 0..n or 0..* **-** Indicates an optional, single or multi-valued reference

Required Reference - the object and the association shall exist (or be instantiated) when the *other* referenced class is defined.

Weak Reference – indicated by the symbol, “w”, indicates that the referenced endpoint or class is “weak” with respect to the *other* class participating in the association. This means that the referenced class is scoped or named relative to the other class, and the identifying keys of the other class are placed as properties in the “weak” class.

Note that this is not standard UML convention, but an added symbol in CIM diagrams.

2.1.104 Universal Modeling Language (UML)

Refer to UML Standards. See Table 92.

2.1.105 URL

Uniform Resource Locator.

2.1.106 User Agent (UA)

In the context of SLP, a process that attempts to establish contact with one or more services. A User Agent retrieves service information from Service Agents or Directory Agents.

2.1.107 VAR

Value Added Reseller.

2.1.108 Volume Set

Synonym for virtual disk.

2.1.109 WAN

Acronym for Wide Area Network. A communications network that is geographically dispersed and that includes telecommunications links.

2.1.110 Weak Reference

Refer to UML Standards. See Table 92.

2.1.111 WBEM

Acronym for Web Based Enterprise Management. Web-Based Enterprise Management is an initiative in the DMTF. It is a set of technologies that enables interoperable management of an enterprise. WBEM consists of CIM, an XML DTD defining the tags (XML encodings) to describe the CIM Schema and its data, and a set of HTTP operations for exchanging the XML-based information. CIM joins the XML data description language and HTTP transport protocol with an underlying information model, CIM to create a conceptual view of the enterprise.

2.1.112 W3C

World Wide Web Consortium.

2.1.113 XML

Acronym for eXtensible Markup Language. A universal format for structured documents and data on the World Wide Web. The World Wide Web Consortium is responsible for the XML specification. cf. http://www.w3.org/XML/.

2.1.114 XML-CIM Listener

A server application that receives and processes XML-CIM Export Message requests and issues CIM Export Message responses.

2.1.115 XML-CIM Server

A Server that receives and processes XML-CIM Operation Requests and issues XML-CIM Operation Responses.

2.1.116 Zone

A group of ports and switches that allow access. Defined by a zone definition. cf. Hard Zone, Soft Zone

A collection of Zone Members. Zone Members in a Zone are made aware of each other, but not made aware of devices outside the Zone. A Zone can be defined to exist in one or more Zone Sets.

2.1.117 Zone Definition

The parameters that define a Zone: the Zone Name, number of Zone Members, and Zone Member definition.

2.1.118 Zone Member

An N_Port (or NL_Port) to be included in a Zone, as specified by its Zone Member Definition. N_Ports at well known addresses shall not be specified as Zone Members.

2.1.119 Zone Member Definition

The parameter by which a Zone Member is specified. A Zone Member may be specified by:

- a port on a Switch, (specifically by Domain_ID and port number); or,
- the device’s N_Port_Name; or,
- the device’s address identifier; or,
- the device’s Node_Name.

2.1.120 Zone Set

One or more Zones that may be activated or deactivated as a group.

Zone Set Name: The name assigned to a Zone Set.

Zone Set State: The state of a Zone Set, which may be either activated or deactivated.

Active Zone Set: The Zone Set that is currently activated. Only one Zone Set may be activated at any time.

2.2 Symbols and abbreviations

||	concatenate
= or EQ	equal
≈	approximately equal
API	application programming interface
CDB	command descriptor block
FC	Fibre Channel
HBA	host bus adapter
IDL	interface definition language
IETF	Internet Engineering Task Force
IMA	iSCSI Management API
IP	Internet Protocol
iSCSI	Internet SCSI
iSD	iSCSI Direct
LSB	least significant bit
OS	operating system
RFC	Request for Comments
SCSI	Small Computer System Interface
SAM-3	SCSI Architecture Model
SAN	storage area network

SPC-3	SCSI Primary Commands-3
TC	Technical Committee
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

2.3 Keywords

2.3.1 expected

A keyword used to describe the behavior of the hardware or software in the design models presumed by this standard. Other hardware and software design models may also be implemented.

2.3.2 invalid

A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt of an invalid bit, byte, word, field or code value shall be reported as an error.

2.3.3 mandatory

A keyword indicating an item that is required to be implemented as defined in this standard to claim compliance with this standard.

2.3.4 may

A keyword that indicates flexibility of choice with no implied preference.

2.3.5 may not

Keywords that indicates flexibility of choice with no implied preference.

2.3.6 obsolete

A keyword indicating that an item was defined in prior standards but has been removed from this standard.

2.3.7 opaque:

A keyword indicating that value has no semantics or internal structure.

2.3.8 optional

A keyword that describes features that are not required to be implemented by this standard. However, if any optional feature defined by this standard is implemented, it shall be implemented as defined in this standard.

2.3.9 reserved

A keyword referring to bits, bytes, words, fields and code values that are set aside for future standardization. Their use and interpretation may be specified by future extensions to this or other standards. A reserved bit, byte, word or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words or fields for zero values. Receipt of reserved code values in defined fields shall be reported as an error.

2.3.10 shall

A keyword indicating a mandatory requirement. Designers are required to implement all such requirements to ensure interoperability with other products that conform to this standard.

2.3.11 should

A keyword indicating flexibility of choice with a preferred alternative; equivalent to the phrase “it is recommended”.

2.4 Conventions

Certain words and terms used in this American National Standard have a specific meaning beyond the normal English meaning. These words and terms are defined either in Clause 2:, "Definitions, Symbols, Abbreviations, and Conventions" or in the text where they first appear.

Numbers that are not immediately followed by lower-case b or h are decimal values.

Numbers immediately followed by lower-case b (xxb) are binary values.

Numbers immediately followed by lower-case h (xxh) are hexadecimal values.

Hexadecimal digits that are alphabetic characters are upper case (i.e., ABCDEF, not abcdef).

Hexadecimal numbers may be separated into groups of four digits by spaces. If the number is not a multiple of four digits, the first group may have fewer than four digits (e.g., AB CDEF 1234 5678h)

Decimal fractions are initiated with a comma (e.g., two and one half is represented as 2,5).

Decimal numbers having a value exceeding 999 are separated with a space(s) (e.g., 24 255).

A numeric list (e.g., 1,2,3) of items indicate the items in the list are ordered (i.e., item 1 shall occur or complete before item 2).

In the event of conflicting information the precedence for requirements defined in this standard is

1) text,
2) tables, then
3) figures.

Clause 3: Business Overview

3.1 Preamble

Large Storage Systems and Storage Area Networks (SANs) are emerging as a prominent and independent layer of IT infrastructure in enterprise class and midrange computing environments. Examples of applications and functions driving the emergence of new storage technology include:

- Sharing of vast storage resources between multiple systems via networks,
- LAN free backup,
- Remote, disaster tolerant, on-line mirroring of mission critical data,
- Clustering of fault tolerant applications and related systems around a single copy of data.
- Archiving requirements for sensitive business information.
- Distributed database and file systems.

To accelerate the emergence of more functional and sophisticated storage systems in the market, the industry requires a standard management interface that allows different classes of hardware and software products supplied by multiple vendors to reliably and seamlessly interoperate for the purpose of monitoring and controlling resources. The SNIA Storage Management Initiative (SMI) was created to develop this specification (SMI-Specification or SMI-S), the definition of that interface. This standard provides for heterogeneous, functionally rich, reliable, and secure monitoring/control of mission critical global resources in complex and potentially broadly-distributed, multi-vendor storage topologies like SANs. As such, this interface overcomes the deficiencies associated with legacy management systems that deter customer uptake of more advanced storage management systems.

3.2 Business Rationale

This interface is targeted at creating broad multi-vendor management interoperability and thus increasing customer satisfaction. To that end, this specification defines an “open” and extensible interface that allows subsystems and devices within the global context of a large storage system to be reliably and securely managed by overlying presentation frameworks and management systems in the context of the rapidly evolving multi-vendor market. In specific, SAN integrators (like end-users, VARs, and SSPs) can, via this standardized management interface, more flexibly select between multiple vendors when building the hierarchy of software systems required to manage a large storage system independent of the underlying hardware systems. Additionally, storage integrators can more flexibly select between alternate hardware vendors when constructing storage configurations. Broad adoption of the standards defined and extended in this specification will provide increased customer satisfaction and will:

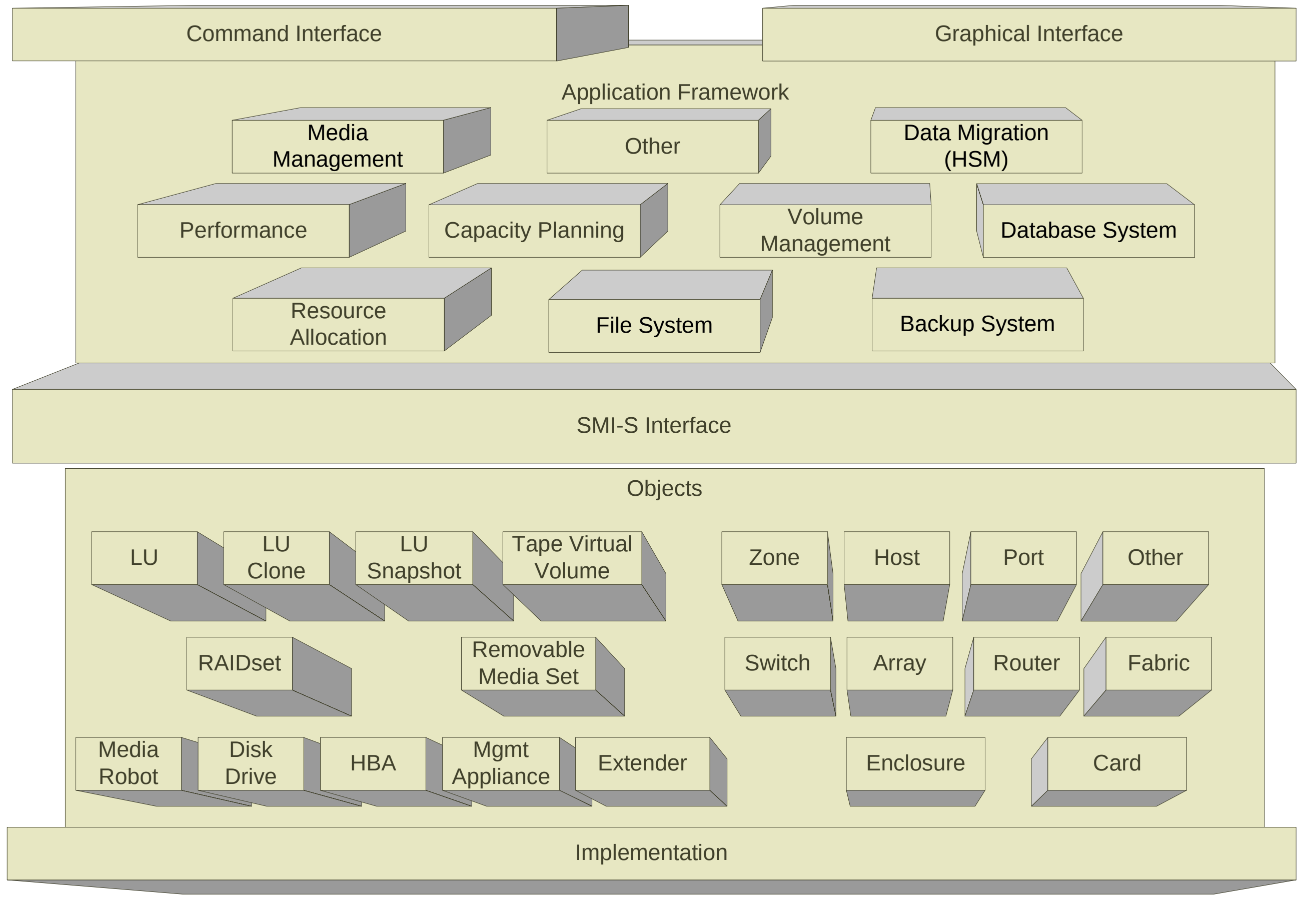
- More rapidly expand the acceptance of new storage management technology like SANs and iSCSI;
- Accelerate customer acquisition of new storage management technology;
- Expand the total market.

Additionally, a single common management interface allows SAN vendors and integrators to decrease the time required to bring new more functional technology, products, and solutions to market.

3.3 Interface Definition

This management interface allows storage management systems to reliably identify, classify, monitor, and control physical and logical resources in a storage system. The fundamental relationship of this interface to storage management software, presentation frameworks, user applications, SAN physical entities (i.e., devices), SAN discovery systems, and SAN logical entities is illustrated in Figure 1: "Interface Functions".

Figure 1: Interface Functions



The diagram illustrates that functions of the interface can be distributed across multiple devices (i.e., Switches or Array Controllers) and/or software systems (i.e., Discovery Systems). While the functionality of the interface is distributed within or across a storage environment, to insure that monitoring and control operations by clients are consistent and reliable, the state of a given resource is not certain to be valid if it is simultaneously available to clients from multiple unsynchronized sources.

Example: A request by an SRM application and a backup engine for the bandwidth available on a given Fibre Channel path should be coordinated by a single monitoring entity to insure information consistency. If the SRM application and Backup engine obtain different available bandwidth information for a given Fibre Channel path from multiple unsynchronized sources they could function in conflict and degrade the efficiency of the environment.

Addressing this concern is the responsibility of parties configuring Storage and Network management clients that rely on the primitives defined in the specification.

Note: Within this architecture (as depicted by the illustration above) entities like an appliance-based volume manager may potentially act as both a client and a server to the interface.

Example: A Host-based volume manager wants to construct a large storage pool from multiple SAN appliance based volumes, as well as volumes/LUNs originating from array controllers. In this case, the host based volume manager needs to inspect the characteristics of the volumes on both the SAN appliance and array controller prior to allocation. Additionally, the SAN appliance (which runs a volume manager) needs to inspect the properties of storage devices when building its volumes. As such, the SAN appliance in this case is both a client and server in the management environment, depending on the action being performed.

Figure 1: "Interface Functions" includes a number of strategic functional requirements for the interface. These capabilities will be introduced to the interface implementation over time, and may not be present in this version of the interface. The functionalities required to fully satisfy the needs of clients using a storage management interface include:

a) Clients need to be able to obtain sufficient information to discern the topology of the SAN or complex storage system;

b) Clients need to be able to reliably identify resources that have experienced an error/fault condition that has resulted in degraded/disabled operation;

c) Clients need to be able to construct a zone of allocation around a select group of host and storage resources;

d) Clients need to be able to identify nonvolatile storage resources available to a storage management system, to allow them to construct a storage pool of a consistent level of performance and availability;

e) Clients need to be able to identify third-party copy engines (and associated media libraries/robots) available to a cooperating backup engine, allowing it to allocate an engine/library/robot to a given backup task;

f) Clients need to be able to dynamically allocate non-volatile storage resources;

Note: Each volume to be utilized is subject to strict availability and performance requirements. As a result, the file system needs to inspect the properties of each volume prior to allocation.

g) Clients need to be able to access sufficient topology and component information to allow a Storage Resource Management (SRM) application like a performance monitor to examine topology and line utilization, such that performance bottlenecks can be exposed and capacity planning performed;

h) Clients need to be able to employ appropriate data reporting and tracking to allow capacity planning system to identify each storage pool in the SAN and then interact with the manager of each pool to assess utilization statistics;

i) Clients need to be provided with adequate controls for a privileged, user-written application to restrict the use of a volume to a specific host, set of hosts, or set of controller communications ports;

j) Clients need to be assured of timely propagation of data concerning the health and performance of the devices and subsystems in the SAN to fault isolation and analysis systems.

Example non-goals for this interface include:

a) The ability to select a logical communications port over which to send/receive data;

b) The ability to read or write data to a volume;

c) The ability to identify and recover from data communications errors and failures;

d) The ability to log a new communications device into a network.

3.4 Technology Trends

To be broadly embraced and long lived this management interface should respect and leverage key technology trends evolving within the industry. These include:

a) Improved Connectivity: Whether available In-band (i.e., over Fibre Channel/iSCSI) or available out-of-band (i.e., over a LAN/MAN/WAN), or available over a mix of both, virtually all devices in a storage management environment have (or soon will have), access to a common communications transport suitable for carrying management information content (e.g., TCP/IP), that is used to transmit a standardized encoding (e.g., CIM-XML) of recognized semantics (e.g., CIM);

b) Increased Device Manageability: Through a common, general-purpose network transport, provide the option to provide proxy services to provide access to (e.g., general purpose computer system) devices via this standardized management interface;

Example: A legacy array controller is incapable of running the software necessary to implement a management server for this interface and uses a proxy server on a SAN appliance to communicate within the management environment.

Example: An HBA is incapable of running the software necessary to implement a management server for this interface and uses a proxy server on its host system to communicate within the management environment.

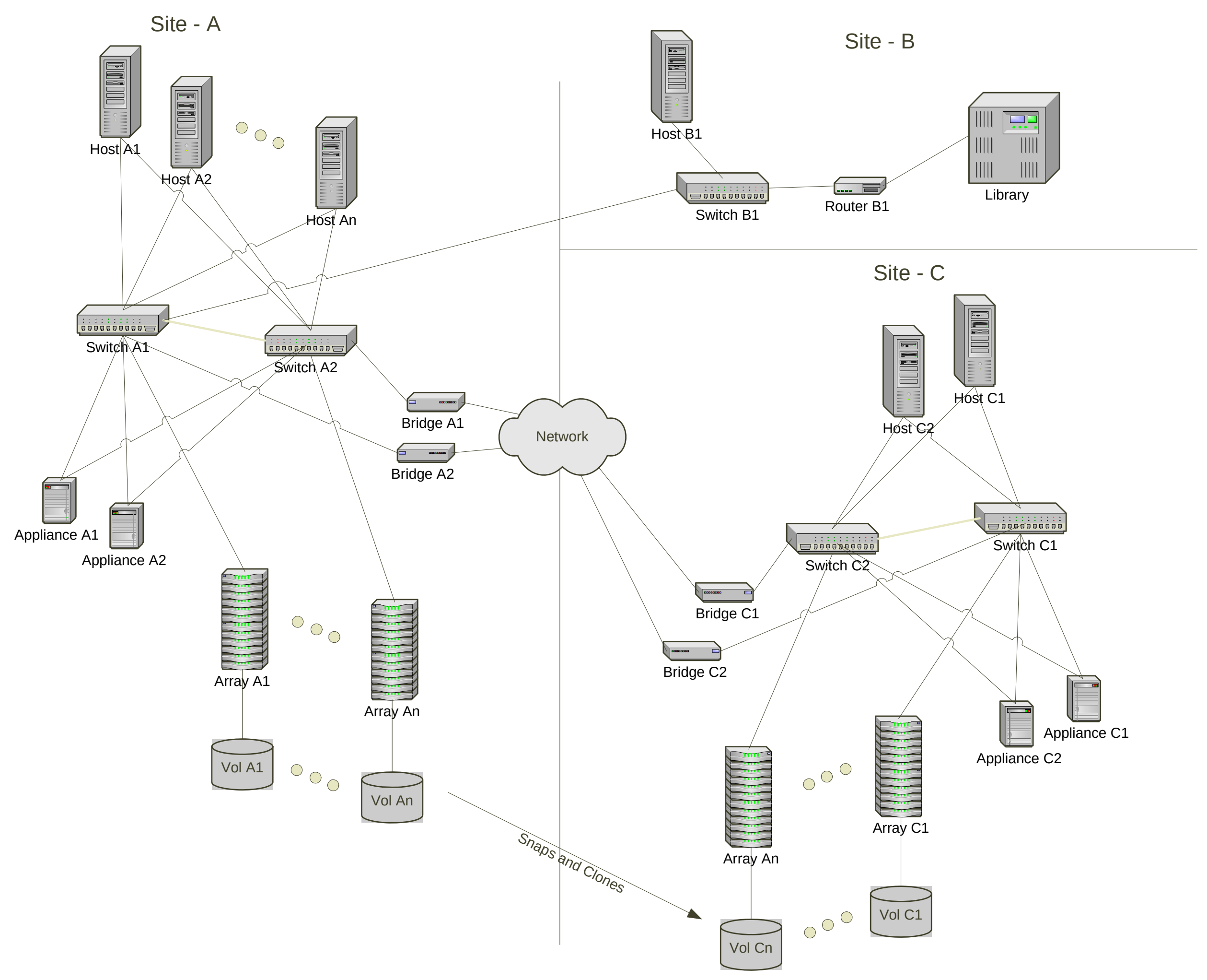
c) XML Standardization: XML is providing the ability to create management protocols with an extensible, platform independent, human readable, content describable communication language. This streamlines the task of developing infrastructure to support his interface and debug systems around the interface.

d) Object Independent Protocols: These protocols provide appropriate abstraction – separating the definition of the object model from the semantics/syntax of the protocol. Additionally, the transport-independent, content-description (i.e., markup) nature of XML allows it to be utilized by both web-enabled application and appliances;

e) Increased SAN Complexity: SANs are being configured with diverse classes of components and widely distributed topologies. Management clients and servers in the environment need to anticipate being widely distributed on systems, appliances and devices throughout large SAN topologies, while maintaining real-time distributed state for logical entities. Figure 2: "Large SAN

Topology" provides an example of a single SAN built from multiple classes of components spanning three physical locations (i.e., Sites A, B and C).”.

Figure 2: Large SAN Topology

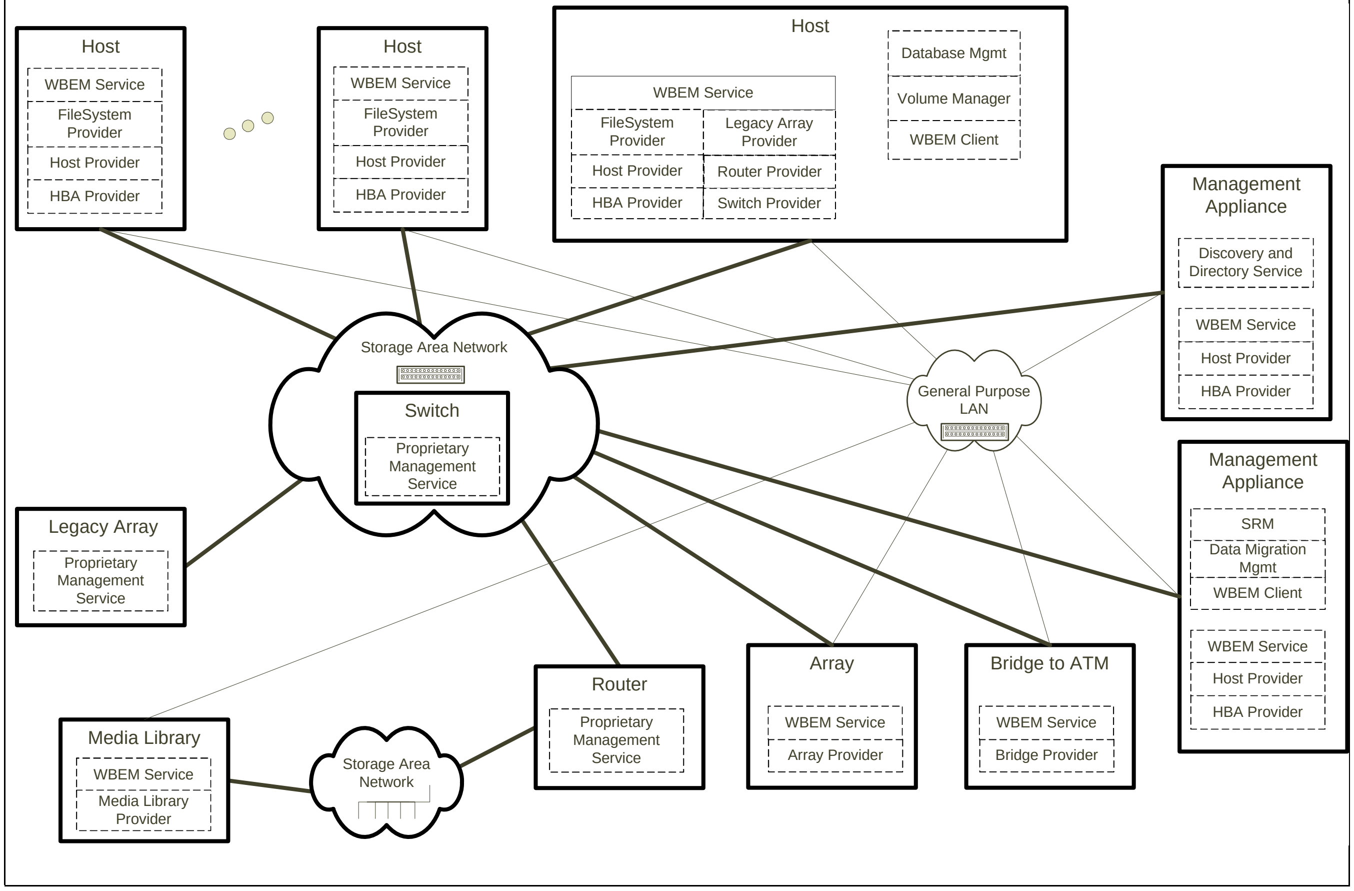


3.5 Management Environment

Clients and Servers of this interface can be widely distributed on systems, appliances, and devices across a network that includes one or more large SAN topologies.

The configuration in Figure 3: "Example Client Server Distribution in a SAN" provides an example client/server distribution using in-band TCP/IP communications, out of band TCP/IP communications, or employing proxy services to bridge legacy and/or proprietary communication interfaces. The device “Old Array Controller” is incapable of appropriate communication with clients and servers in the management environment to provide management access (i.e., a CIM Server). Access to the communications transport that clients and servers share for communication is achieved via a proxy service on the host computer in the upper right hand corner of the illustration. All other clients and servers communicate via direct access to a common communications transport.

Figure 3: Example Client Server Distribution in a SAN



3.6 Architectural Objectives

The following reflect architectural objectives of the interface. Some of these capabilities are not present in the initial release of the interface, but are inherent in its architecture and intended extensibility. They are intended to provide guidance concerning the present and future direction of development of the Storage Management Initiative Specification.

a) Consistency: State within a managed object and between objects remains consistent independent of the number of clients simultaneously exerting control, the distribution of objects in the environment, or the management action being performed;

b) Isolation: A client that needs to execute an atomic set of management actions against one or more managed objects is able to do so in isolation of other clients, who are simultaneously executing management actions against those same objects;

c) Durability: Consistency, and isolation are preserved independent of the failure of any entity or communications path in the management environment;

d) Consistent Name Space: Managed objects in a single management domain adhere to a consistent naming convention independent of state or reliability of any object, device, or subsystem in the SAN;

e) Distributed Security: Monitoring and control operations are secure. The architecture supports:

    1) Client authentication;
    2) Privacy (encryption) of the content of the messages in this protocol;
    3) Client authorization;

f) Physical Interconnect Independence: The interface will function independent of any particular physical interconnect between components, any supplier, or any topology;

g) Multi-vendor Interoperability: Clients and servers should use a common communication transport and message/transfer syntax to promote seamless plug compatibility between heterogeneous multi-vendor components that implement the interface;

h) Scalability: The size, physical distribution, or heterogeneity of the storage system does not degrade the quality or function of the management interface;

i) Vendor Unique Extension: The interface allows vendors to implement proprietary functionality above and beyond the definitions here-in to distinguish their products and services in the market independent of the release of a new version of the interface;

j) Volatility of State: This interface does not assume that objects are preserved in non-volatile repositories. Clients and servers may preserve object state across failures, but object preservation is not mandatory;

k) Replication: This interface provides no support for the automatic replication of object state within the management environment;

l) Functional Layering Independence: The design of this interface is independent of any functional layering a vendor chooses to employ in constructing the storage management systems (hardware and software) necessary to manage a storage environment;

m) Asynchronous or Synchronous execution: Management actions may execute either asynchronously or synchronously;

n) Events: This interface provides for the reliable asynchronous delivery of events to one or more registered clients;

o) Cancelable Management Actions: Long running synchronous or asynchronous directives need to be capable of being cancelled by the client. Cancellation needs to result in the termination of work by the server and resource consumed being released;

p) Durable Reference: Object classes that persist across power cycles and need to be monitored and controlled independent of SAN reconfiguration (i.e., logical volumes) need be identified via “Durable Names” to insure consistent reference by clients;

q) Dynamic installation and reconfiguration: New clients and servers need to be capable of being added to or removed from a SMI-S management environment without disrupting the operation of other clients or servers. In most cases, clients should be capable of dynamically managing new servers that have been added to a SMI-S environment.

r) Automatic discovery of new servers: When new management servers are added to the management system they should automatically become available to management clients without the need for manual configuration by administrations staff.

3.7 Disclaimer

The SNIA makes no assurance or warranty about the interoperability, data integrity, reliability, or performance of products that implement this specification.

Clause 4: Overview

4.1 Base Capabilities

To achieve the architectural objectives and support the key technological trends in Clause 3:, "Business Overview", this document describes an extensible, secure, auto-discoverable, object-oriented, XML-based messaging based interface designed to support the specific requirements of managing devices in and through storage systems. The top level protocol that implements this messaging based interface in this revision of the specification is called Web Based Enterprise Management (WBEM) and more specifically CIM/XML over http. To quickly become ubiquitous, SMI-S seeks to the greatest extent possible to leverage a number of existing enterprise management standards through this interface, such as:

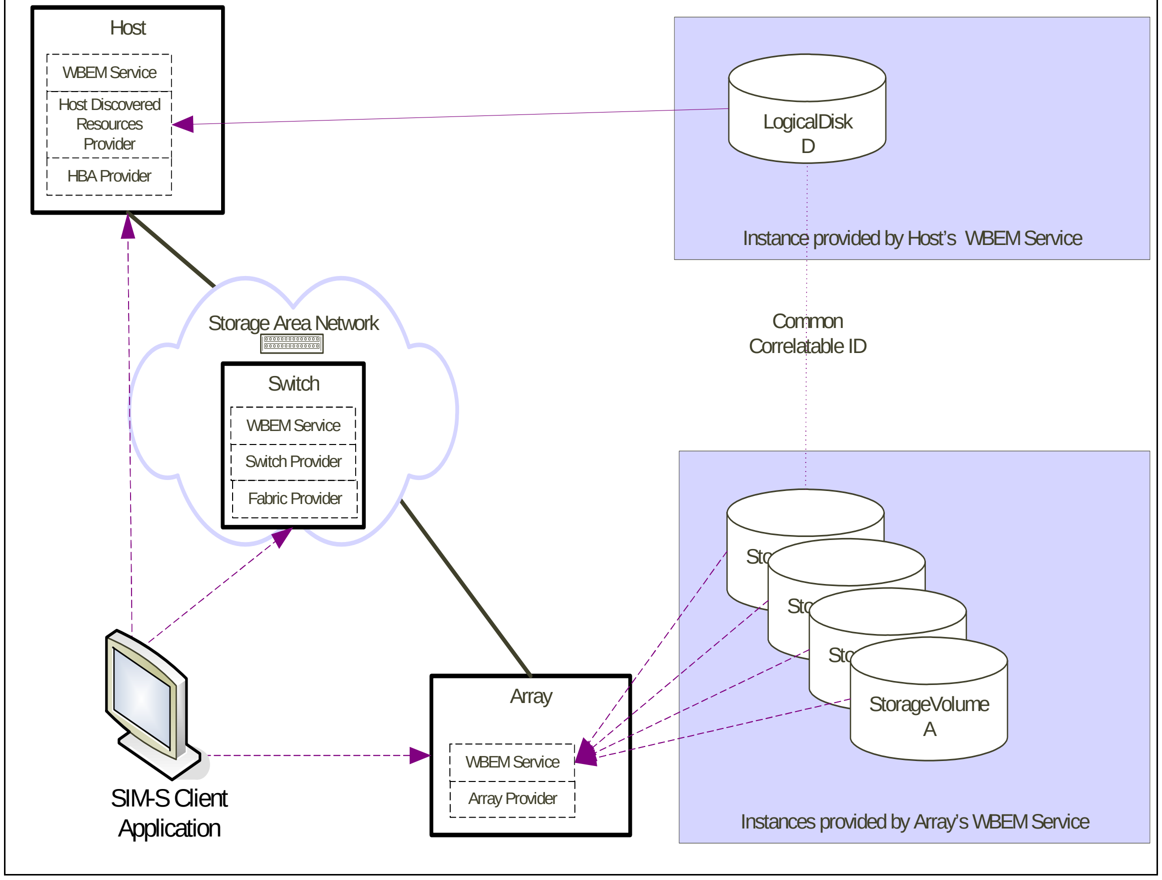
- The Distributed Management Task Force (DMTF) authored Common Information Model (CIM) and Web Based Enterprise Management (WBEM) standards,
- The standards written by ANSI on Fibre Channel and SCSI,
- The World Wide Web Consortium (W3C) for standards on XML,
- The Internet Engineering Task Force (IETF) for standards on HTTP, SLP, and iSCSI.

4.1.1 Object Oriented

A hierarchy of object classes with properties (a.k.a. attributes) and methods (a.k.a. directives) linked via the Universal Modeling Language (UML) modeling constructs of inheritance and associations defines most of the capabilities of the SMI-S. The SMI-S object model (which constitutes the bulk of this specification) is integrated with and part of the Common Information Model (CIM) at the DMTF. Implementers of this specification are encouraged to consult one of the many publicly available texts on UML or the uml.org web site (www.uml.org) to develop an understanding of UML. A brief tutorial on

UML is provided in the introduction material on the Clause 6:, "Object Model General Information" in this specification.

Figure 4: Object Model/Server Relationship



In Figure 4: "Object Model/Server Relationship", a SMI-S client obtains object classes and instances that it can use to manage the storage. At this level of discussion, the focus is on SMI-S conformant WBEM Clients and Servers. The WBEM Servers have providers for the various components that are responsible for the class and association instances that allow the underlying component implementation to be managed.

A standard, object-oriented interface, together with a standard interface protocol, allows WBEM Clients to discover, monitor, and control storage and network devices, regardless of the underlying implementation of those devices.

The goal of this document is to clearly and precisely describe the information expected to be available to a WBEM Client from an SMI-S compliant WBEM Service. It relies upon UML diagrams, easy-to-use tables and machine-readable, CIM-compliant Managed Object Format (MOF) (through the CIM model maintained at the DMTF). This is intended to ease the task of client implementation and to ease the task of using existing WBEM Servers. It should be noted that the MOF Interface Description Language is a precise representation of the object model in this specification, and developers are encouraged to learn this means of expression when implementing this interface. Programmers implementing this interface should reference MOF representations of the object model when faced with implementation decisions.

SMI-S compliant WBEM Servers provide instances in a manner conformant to one or more SMI-S profiles (7.1.1, "Profile Content"). The object model supporting these instances may be extended by the vendor as long as it remains conformant to the relevant SMI-S profiles. Generally, vendor-unique code is necessary in a WBEM client to take advantage of vendor defined model extensions. Regardless of the presence of vendor extensions, a generic WBEM client is able to leverage all SMI-S features defined for a supported profile.

Figure 5: "Canonical Inheritance" illustrates this requirement.

Figure 5: Canonical Inheritance

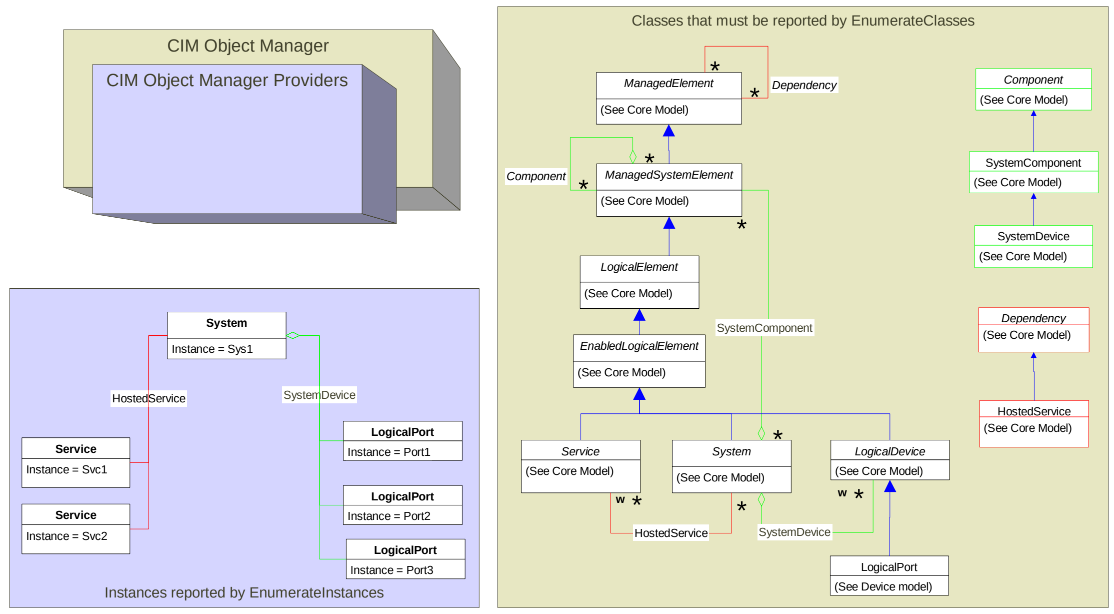


Figure 5: "Canonical Inheritance" illustrates that even though a Fibre Channel Switch may only report instances and allow associated method execution for certain objects, when asked by a client to enumerate its Object Classes it reports the entire hierarchy of classes in its tree. Similarly a server that instantiates an array controller reports the complete set of object classes that links it to the base canonical object of the SMI-S model. It is this single canonical root that allows any SMI-S client to discover, map, and operate upon the complete set of objects in a given SAN.

The object model presented in this specification is intended to facilitate interoperability not limiting the expression of unique features that differentiate manufacturers in the market For this reason, the object model provided only serves as a”core” to compel multi-vendor interoperability. In the interest of gaining a competitive advantage, a given vendor’s implementation of the interface may include additional object classes, properties, methods, events, and associations around this “core”. These vendor-unique extensions to the object model may, in select cases (e.g., extrinsic methods), require the modification of client code above and beyond that required to support the core.

4.1.2 Messaging Based

A messaging-based interface, rather than a more traditional procedure call interface, was selected so that platform and language independence could be achieved across the breadth of devices, clients, and manufacturers that may implement the interface. This messaging-based environment also eases the task of transporting management actions over different communications transports and protocols that may emerge as the computer industry evolves. An example fragment of an SMI-S CIM-XML message is provided in Figure 6: "Sample CIM-XML Message".

Figure 6: Sample CIM-XML Message

```
<!DOCTYPE CIM SYSTEM HTTP://www.dmtf.org/cim-v2.dtd/>
<CIMVERSION=”2.0” DTDVersion=”2.0”>
     <CLASS NAME=”ManagedSystemElement”>
        <QUALIFIER NAME=”abstract”></QUALIFIER>
        <PROPERTY NAME=”Caption” TYPE=”string”>
           <QUALIFIER NAME=”MaxLen” TYPE=”sint32”>
              <VALUE>64</VALUE>
           </QUALIFIER>
        </PROPERTY>
        <PROPERTY NAME=”Description” TYPE=”string”></PROPERTY>
        <PROPERTY NAME=”InstallDate” TYPE=”datetime”>
           <QUALIFIER NAME=”MappingStrings” TYPE=”string”>
              <VALUE>MIF.DMTF|ComponentID|001.5</VALUE>
           </QUALIFIER>
        </PROPERTY>
        <PROPERTY NAME=”Status” TYPE=”string”>
           <QUALIFIER NAME=”Values” TYPE=”string” ARRAY=”TRUE”>
              <VALUE>OK</VALUE>
              <VALUE>Error</VALUE>
```

4.2 Functionality Matrix

4.2.1 Overview

The functionality enabled by this version of the Storage Management Initiative Specification organized follows a multi-level model. Within each level of this model, several broad categories of management are described. This creates a functionality matrix, which serves two purposes. First, it organizes a complex set of capabilities enabled by the overall SMI-S approach. Second, it helps to ensure good management functionality coverage for the managed devices comprehended by SMI-S. This section provides an overview of the functionality matrix approach for describing the management functionality provided by this version of SMI-S. A blank functionality matrix is provided in Table 1.

Table 1: Functionality Matrix

	Fault Management	**Configuration Management**	**Accounting Management**	**Performance Management**	**Security Management**
Application Level					
File / Record Level					
Block Level					
Connectivity Level					
Device Level					

4.2.2 Multi-Level Model Of Networked Storage Management Functionality

The lowest level of the multi-level model of networked storage management functionality applies to managing the basic physical aspects of the elements found in a networked storage environment, and the upper levels are involved with managing the different logical levels supported by these managed elements. Each level in this model depends upon the lower levels being in place.

Shown in top-down order, the functionality levels are:

- (Level 5) Application Level Functionality,
- (Level 4) File/Record Level Functionality,
- (Level 3) Block Level Functionality,
- (Level 2) Connectivity Level Functionality,
- (Level 1) Device Level Functionality.

Managed physical elements in a networked storage environment shall support Level 1 functionality, and may support additional functionality levels as well, depending upon the logical capabilities of the managed physical element. The functionality supported by a managed element will normally involve a contiguous set of levels in this model. If a managed physical element supports functionality for a particular upper level, then it will also support functionality for each level between that level and Level 1.

As an example of this last point, consider a NAS Head device. It has a physical component (Level 1). It is connected to other physical components in the networked storage environment (Level 2). It deals with Block storage (Level 3), and it deals with Files (Level 4). A NAS Head device can therefore be expected to support functionality in levels 1 through 4 of this multi-level model of networked storage management functionality. Similarly, a regular NAS device would support management functionality in each of these same levels, although the functionality supported within each level might be slightly different, since the regular NAS device does not have a SAN back-end.

4.2.3 FCAPS

Within each level of this model, a basic set of functionality is needed that allows management applications to exercise FCAPS capabilities over elements supporting that level. FCAPS is a model of the working objectives of network management, and these same concepts are applied to each of the levels in the multi-level model of networked storage management functionality. A summary of FCAPS capabilities includes:

- **F**ault Management: Identifying, isolating, correcting, and logging managed element faults. Includes running diagnostics, generating fault alarms, and keeping error statistics,
- **C**onfiguration Management: Discovering, configuring, and monitoring managed elements. Includes adding, altering, and removing managed elements,
- **A**ccounting Management: Measuring and tracking usage of managed elements or services. Includes distributing resources, setting quotas, and billing,
- **P**erformance Management: Monitoring of performance, error rate, and utilization metrics for managed elements. Includes setting thresholds, problem reporting, logging of data, and examining historical data
- **S**ecurity Management: Ensuring legitimate use of managed elements or services. Includes checking user access rights, maintaining an audit trail log, generating security events and alarms, and maintaining data confidentiality where necessary.

By specifying FCAPS capabilities within each of its levels, this multi-level model is used to describe the functionality that is provided by SMI-S overall, and by individual profiles and subprofiles. The actual degree of support for FCAPS capabilities within each level is determined by individual SMI-S profiles.

4.2.4 Management Functionality Within Each Level Of The Model

4.2.4.1 (Level 1) Device Level Functionality

This level includes all functionality needed to allow management applications to deal with the physical aspects of managed elements in the networked storage environment. The physical aspects of HBAs, Switches, Storage Systems etc. are handled by functionality in this level. This level also handles functionality that is not exposed to other elements in the networked storage environment, like the managing of storage devices within a Storage System prior to their being allocated to storage pools that are accessible over the data network.

4.2.4.2 (Level 2) Connectivity Level Functionality

This level includes all functionality associated with allowing management applications to deal with the logical aspects of the managed connectivity between physical elements in the networked storage environment. This level is where things like Fibre Channel Fabrics and Zones are handled, and is also where iSCSI Sessions are handled. This level also handles the logical aspects of switch and extender connectivity.

4.2.4.3 (Level 3) Block Level Functionality

This level includes all functionality necessary to allow management applications to deal with storage volumes in a networked storage environment. This level applies to Logical Units, LUN Masking and Mapping, block aggregators like Volume Managers, etc. It also applies to block-level virtualization.

4.2.4.4 (Level 4) File/Record Level Functionality

This level includes all functionality associated with allowing management applications to deal with data objects like file systems in a networked storage environment. Note that this level not only applies to file systems -- it is also applies to records, for the structured usage of block storage by middleware applications such as databases and e-mail servers. This level provides the functionality that enables management applications to determine the capacity utilization of the storage volumes handled by the Block Level Functionality.

4.2.4.5 (Level 5) Application Level Functionality

This level includes all functionality needed to allow management applications to deal with managed applications in the networked storage environment. This level applies to database applications, e-mail server applications, etc. that work directly with the data objects handled by the File/Record Level Functionality.

4.2.5 Referring To Levels And Capabilities In The Multi-level Model

To simplify talking about the different levels and capabilities within this multi-level model of networked storage management functionality, the following short-hand notation may be used in SMI-S.

Individual functionality levels are referred to as L1 through L5, and a single letter appended to this level indicates a particular kind of FCAPS capability. For instance, fault management functionality within the connectivity layer would be referred to as L2F functionality, and configuration management functionality for a physical device would be referred to as L1C functionality.

4.2.6 Functionality Descriptions in SMI-S Profiles

To make it easier to understand the management functionality coverage provided by individual profiles and subprofiles in this SMI-S document, each profile lists the functionality provided by the profile and its subprofiles. If a function is provided by a subprofile, this is indicated, including whether the subprofile is optional or required. Functionality listed in the profile is organized by Level, and within each Level by FCAPS category, as defined here by the Functionality Matrix.

4.3 Capabilities of This Version

This section summarizes, at a high level, the capabilities provided by this SMI-S version based on the Functionality Matrix, and is organized by Level.

4.3.1 Device Level

4.3.1.1 Fault Management

SMI-S device profiles that include the Health Package (8.2.1.6) provide capabilities for reporting of the SAN device health and status, including the type, category, and source of the failures. Asynchronous notification for changes in device health status is also provided via the Indications Subprofile (8.2.4.2).

4.3.1.2 Configuration Management

SMI-S defines the capabilities needed for the discovery, configuration, and monitoring of devices in a SAN. Asynchronous notification for changes in device configuration is provided via the Indications Subprofile (see 8.2.4.2).

4.3.1.3 Accounting Management

Other than basic device discovery, SMI-S provides no specific capabilities for device Accounting Management.

4.3.1.4 Performance Management

SMI-S enables performance management of some SAN devices (see 8.2.8.11).

4.3.1.5 Security Management

SMI-S provides device-level security via basic authentication capabilities. See the SMI-S Security section (8.2.5.1) and Device Credentials Profile (8.2.1.4) for more information. Note that the secure communication between a device proxy CIM Server and the device is outside of the scope of SMI-S.

4.3.2 Connectivity Level

4.3.2.1 Fault Management

SMI-S provides the capability to identify the heath of interconnects between SAN devices, mainly via LogicalPort.OperationalStatus (see 8.2.2.1, "Parallel SCSI (SPI) Target Ports Subprofile", 8.2.2.2, "FC Target Port Subprofile", 8.2.2.3, "iSCSI Target Ports Subprofile", 8.2.2.4, "Direct Attach (DA) Port Subprofile", 8.2.3, "Common Initiator Port Subprofiles Overview", 8.2.3.1, "Parallel SCSI (SPI) Initiator Port Subprofile", 8.2.3.2, "Fibre Channel Initiator Port Subprofile", 8.2.3.3, "iSCSI Initiator Port Subprofile", 8.2.3.4, "Back End Ports Subprofile (DEPRECATED)" and 8.2.6.6, "Switch Profile". Asynchronous notification for changes in link health status is also provided via the Indications Subprofile (8.2.4.2).

4.3.2.2 Configuration Management

SMI-S defines the capabilities needed for the discovery, configuration, and monitoring of interconnects between devices in a SAN. Asynchronous notification for changes in the fabric configuration is provided via the Indications Subprofile.

4.3.2.3 Accounting Management

Connectivity-level Accounting Management is enabled in SMI-S via basic discovery capabilities and usage tracking via the optional Fabric Path Performance Subprofile.

4.3.2.4 Performance Management

SMI-S enables performance management of SAN Interconnects via both the FCPortStatistics (8.2.6.6.8.10) class and also the transport-independent Fabric Path Performance Subprofile.

4.3.2.5 Security Management

SMI-S provides Connectivity-level security via basic device authentication capabilities, Zone Control and Enhanced Zoning subprofiles, and the Fabric Security subprofile.

4.3.3 Block Level

4.3.3.1 Fault Management

SMI-S Block-level profiles that include the Health Package (8.2.1.6) provide capabilities for reporting of block level health and status, including the type, category, and source of the failures.

4.3.3.2 Configuration Management

SMI-S defines the capabilities needed for the discovery, configuration, and monitoring block-level resources. This includes ability to discover, create, delete, and modify StorageVolumes in the SAN.

4.3.3.3 Accounting Management

SMI-S enables accounting management of Block-level resources via basic discovery and discovery of access rights and mappings.

4.3.3.4 Performance Management

SMI-S provides performance management capabilities for SAN Block-level resources (as provided by Arrays, Virtualization systems and Volume Managers) via the Block Server Performance subprofile (8.2.8.11).

4.3.3.5 Security Management

SMI-S provides the ability to manage (create/delete, enable/disable) connectivity and access rights to Storage Volumes in the SAN.

4.3.4 File/Record Level

4.3.4.1 Fault Management

SMI-S NAS profiles provide Indications support on OperationalStatus for the FileSystems and FileShares.

4.3.4.2 Configuration Management

SMI-S NAS profiles provide discovery of logical storage (StoragePools) and storage extents on logical disks.

4.3.4.3 Accounting Management

This version of SMI-S defines no unique accounting management capabilities at the File level.

4.3.4.4 Performance Management

This version of SMI-S defines no unique performance management capabilities at the File level.

4.3.4.5 Security Management

This version of SMI-S defines no unique security management capabilities at the File level.

4.3.5 Application Level

This version of SMI-S does not address functionality at the application level.

4.4 Operational Environment

Figure 7: Operational Environment

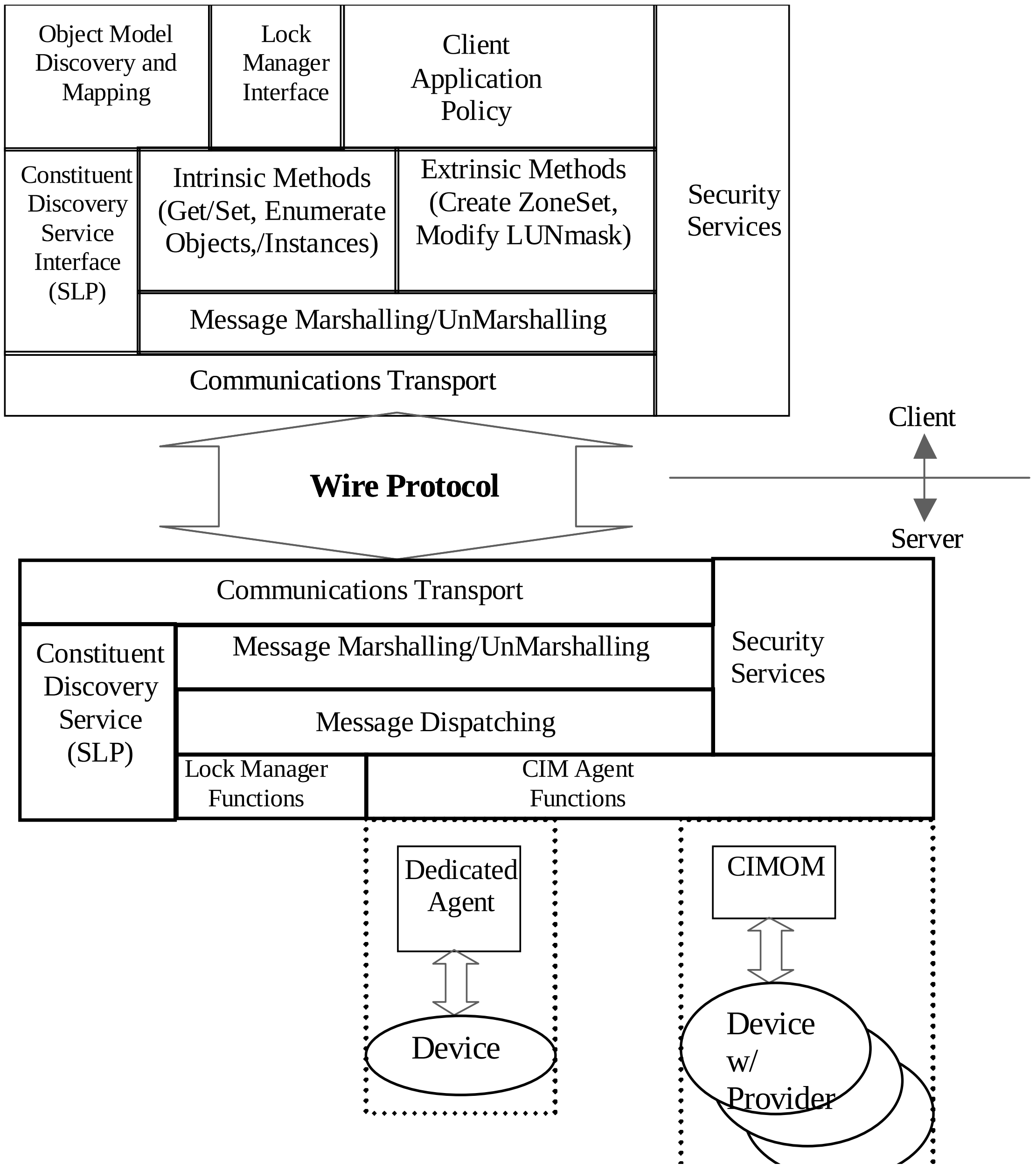

Object Model Discovery and Mapping
Lock Manager Interface
Client Application Policy
Security Services
Constituent Discovery Service Interface (SLP)
Intrinsic Methods (Get/Set, Enumerate Objects,/Instances)
Extrinsic Methods (Create ZoneSet, Modify LUNmask)
Message Marshalling/UnMarshalling
Communications Transport
Wire Protocol
Client
Server
Communications Transport
Security Services
Constituent Discovery Service (SLP)
Message Marshalling/UnMarshalling
Message Dispatching
Lock Manager Functions
CIM Agent Functions
Dedicated Agent
CIMOM
Device
Device w/ Provider

Figure 7: "Operational Environment" illustrates activities that either clients or servers need to account for in or to provide facilities to support:

- The discovery of constituents in the managed environment;
- The discovery of object classes as well as related associations, properties, methods, indications, and return status codes that are provided by servers in the managed environment;
- The security or resources and communications in the environment;
- The locking of resources in the presence of non-cooperating clients; (the definition of locking is left for a future version of the specification)

- The marshalling/un-marshalling of communication messages;
- The execution of basic methods that are “intrinsic” to the construction, traversal, and management of the object model provided by the distributed servers in a SAN;
- The execution of object specific “extrinsic” methods that provide clients the ability to change the state of entities in the SAN.

In addition, to facilitate ease of installation, startup, expansion, and upgrade requirements for implementations are specified for the developers of clients and servers.

4.5 Using this Specification

This specification is insufficient as a single resource for the developers of SMI-S clients and servers. Developers are encouraged to first read the DMTF specifications on CIM, CIM Operations over HTTP, and CIM-XML, as well as obtaining familiarity with UML and the IETF specification on Service Location Protocol (SLP).

A developer implementing SMI-S clients/servers should read this specification in sequence noting that 7.1.1, "Profile Content" is intended principally as a reference relative to the particular device type that is being provided or managed in a SMI-S environment.

4.6 Language Bindings

As a messaging interface, this specification places no explicit requirements for syntax or grammar on the procedure call mechanisms employed to convert SMI-S messages into semantics consumable by modern programming languages. The syntax and grammar used to express these semantics is left at the discretion of each SMI-S developer.

Several open-source codebases are available for programmers who wish to streamline the task of parsing SMI-S messages into traditional procedure call semantics and using these semantics to store object instances. Consult the WBEMsource initiative (http://wbemsource.org) for current language bindings available to implement the SMI-S interface.

Clause 5: Transport and Reference Model

5.1 Introduction

5.1.1 Overview

The interoperable management of storage devices and network elements in a distributed storage network requires a common transport for communicating management information between constituents of the management system. This section of the specification details the design of this transport, as well as the roles and responsibilities of constituents that use the common transport (i.e., a reference model).

5.1.2 Language Requirements

To express management information across the interface, a language is needed that:

- Can contain platform independent data structures,
- Is self describing and easy to debug,
- Can be extended easily for future needs.

The World Wide Web Consortium's (W3C) Extensible Markup Language (XML) was chosen as the language to express management information and related operations, as it meets the requirements above.

5.1.3 Communications Requirements

Communications protocols to carry the XML based management information are needed that:

- Can take advantage of the existing ubiquitous IP protocol infrastructures,
- Can be made to traverse inter- and intra-organizational firewalls,
- Can easily be embedded in low cost devices.

The Hyper Text Transport Protocol (HTTP) was chosen for the messaging protocol and TCP was chosen for the base transfer protocol to carry the XML management information for this interface as they meets the requirements above.

5.1.4 XML Message Syntax and Semantics

In order to be successful, the expression of XML management information (messages) across this interface needs to follow consistent rules for semantics and syntax. These rules are detailed in this specification. They are of sufficient quality, extensibility, and completeness to allow their wide adoption by storage vendors and management software vendors in the industry. In addition, to facilitate rapid adoption, existing software that can parse, marshal, un-marshal, and interpret these XML messages should be widely available in the market such that vendor implementations of the interface are accelerated. The message syntax and semantics selected should:

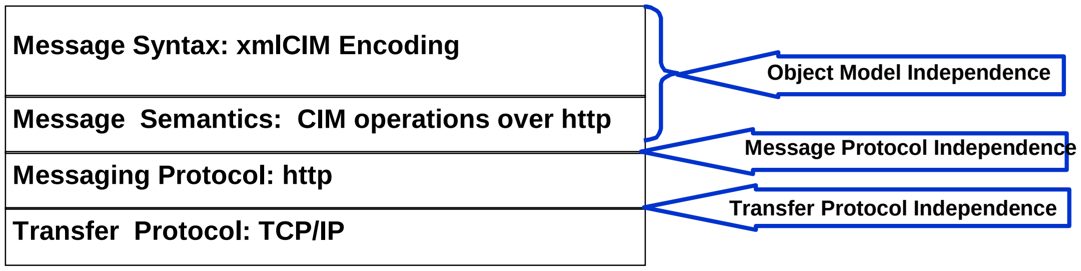
- Be available on multiple platforms,
- Have software implementations that are Open source (i.e., collaborative code base),
- Have software implementations available in Java and C++,
- Leverage industry standards where applicable,
- Conform with W3C standards for XML use.
- Be object model independent (i.e., be able to express any object model).

Virtually the only existing industry standard in this area is the WBEM standards *CIM Operations over HTTP and Specification for the Representation of CIM in XML* as developed and maintained by the DMTF. The WBEMsource initiative is a collaboration of open source implementations, which can be leveraged by storage vendors to prototype, validate, and implement this interface in products. Specifically designed for transporting object-model-independent management information, the CIM-XML message syntax was chosen because it meets the requirements enumerated above. This specification augments the capabilities of CIM-XML in the area of discovery to facilitate ease of management.

5.2 Transport Stack

The complete transport stack for this interface is illustrated in Figure 8: "Transport Stack". It is the primary objective of this interface to drive seamless interoperability across vendors as communications technology and the object model underlying this interface evolves. Accordingly, the transport stack has been layered such that (if required) other protocols can be added as technology evolves. For example, should SOAP or IIOP become prominent, the content in the stack below could be expanded with minimal changes to existing product implementations in the market.

Figure 8: Transport Stack


Message Syntax: xmlCIM Encoding
Message Semantics: CIM operations over http
Messaging Protocol: http
Transfer Protocol: TCP/IP
Object Model Independence
Message Protocol Independence
Transfer Protocol Independence

Again, this interface uses two specifications from the DMTF to fully implement the message syntax and semantics for this interface.

The first specification, *CIM Operations over HTTP* details a basic set of directives (semantics) needed to manage any schema over HTTP. The requirement for this basic set of directives is common to nearly to all management frameworks (e.g., create object, delete object, create instance, and delete instance). This class of directive is referred to in this document as an “intrinsic method”. *CIM Operations over HTTP* also provides a client the ability to execute directives that are unique to the specification of a particular object class within a schema (example: chop <method>, apple <object-class>). This class of directive is referred to in this specification as an “extrinsic method”.

The second specification, *Specification for the Representation of CIM in XML*, Version 2.1 details the precise W3C compliant syntax and grammar for encoding CIM into XML.

While some vendors may choose alternate transfer and message protocols for unique implementations, implementations of the transport stack elements listed above are required for conformance with this standard.

It should be noted that this specification places no restriction on the physical network selected to carry this transport stack. For example, a vendor can choose to use in-band communication over Fibre-channel as the backbone for this interface. Another vendor could exclusively (and wisely) choose out-of-band communication over Ethernet to implement this management interface. Additionally, select vendors could choose a mix of in-band and out-of-band physical network to carry this transport stack.

5.3 Reference Model

5.3.1 Overview

As shown in Figure 9: "Reference Model", the Reference Model shows all possible constituents of the management environment in the presence of the transport stack for this interface.

Figure 9: Reference Model

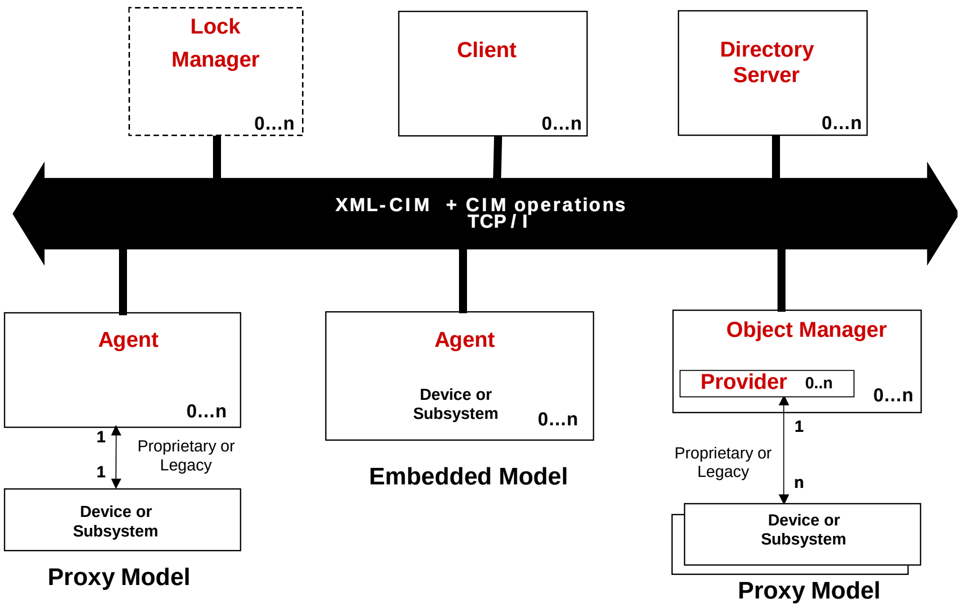


Figure 9: "Reference Model" illustrates that the transport for this interface uses CIM Operations over HTTP with xmlCIM encoding and HTTP/TCP/IP to execute intrinsic and extrinsic methods against the schema for this interface.

Note: It is envisioned that a more complete version of this reference model would include the Lock Manager. However, the Lock Manager in SMI-S Release 1 is preliminary and subject to change. As a result, it is shown as a dotted box to illustrate where the role would fit.

5.3.2 Roles for Interface Constituents

5.3.2.1 Client

A Client is the consumer of the management information in the environment. It provides an API (language binding in Java or C++ for example) for overlying management applications (like backup engines, graphical presentation frameworks, and volume managers) to use.

5.3.2.2 Agent

An agent is a CIM Server. It shall implement those functional profiles, as defined in the DMTF specifications, necessary to satisfy the SMI-S profile with which it conforms. Often, an agent controls only one device or subsystem, and is incapable of providing support for complex intrinsic methods like schema traversal. An agent can be embedded in a device (like a Fibre Channel Switch) or provide a proxy on a host that communicates to a device over a legacy or proprietary interconnect (like a SCSI based array controller).

Embedding an agent directly in a device or subsystem reduces the management overhead seen by a customer and eliminates the requirement for a stand-alone host (running the proxy agent) to support the device.

Embedded agents are the desired implementation for “plug and play” support in an SMI-S managed environment. However, proxy agents are a practical concession to the legacy devices that are already deployed in storage networked environments. In either case, the minimum CIM support for agents applies to either agent deployments.

5.3.2.3 CIM Server

A CIM Server is an object manager that serves management information from one or more devices or underlying subsystems through providers. As such an Object Manager is an aggregator that enables proxy access to devices/subsystems and can perform more complex operations like schema traversals. An object manager typically includes a standard provider interface to which device vendors adapt legacy or proprietary product implementations.

5.3.2.4 Provider

A provider expresses management information for a given resource such as a storage device or subsystem exclusively to a CIM Server. The resource may be local to the host that runs the Object Manager on or may be remotely accessed through a distributed systems interconnect.

5.3.2.5 Lock Manager

This version of the specification does not support a lock manager.

5.3.2.6 Directory Server (SLP Directory Agent)

A directory server provides a common service for use by clients for locating services in the management environment.

5.3.3 Cascaded Agents

This specification discusses constituents in the SMI-S environment in the context of Clients and Servers (Agents and Object Managers). This version of the specification does not allow constituents in a SMI-S management environment to function as both client and server.

Clause 6: Object Model General Information

6.1 Model Overview (Key Resources)

6.1.1 Overview

The SMI-S object model is based on the Common Information Model (CIM), developed by the DMTF. For a more complete discussion of the full functionality of CIM and its modeling approach, see http://www.dmtf.org/standards/cim/.

Readers seeking a more complete understanding of the assumptions, standards and tools that assisted in the creation of the SMI-S object model are encouraged to review the following:

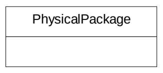
- CIM Tutorial
(http://www.dmtf.org/education/tutorials)
- CIM UML Diagrams and MOFs
(http://www.dmtf.org/standards/standard_cim.php)
- CIM System / Device Working Group Modeling Storage
(http://www.dmtf.org/standards/)

Managed Object File (MOF) is a way to describe CIM object definitions in a textual form. A MOF can be encoded in either Unicode of UTF-8. A MOF can be used as input into an MOF editor, parser or compiler for use in an application.

The SMI-S model is divided into several *profiles*, each of which describes a particular class of SAN entity (such as disk arrays or FibreChannel Switches). These profiles allow for differences in implementations but provide a consistent approach for clients to discover and manage SAN resources. IN DMTF parlance, a *provider* is the instrumentation logic for a profile. In many implementations, providers operate in the context of a *CIM Server* that is the infrastructure for a collection of providers. A WBEM *client* interacts with one or more WBEM Servers.

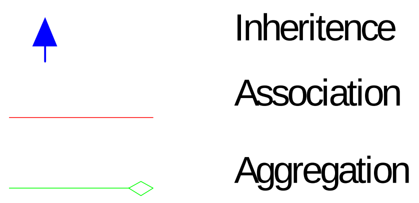
6.1.2 Introduction to CIM UML Notation

CIM diagrams use a subset of Unified Modeling Language (UML) notation.

Most classes are depicted in rectangles. 

The class name is in the upper part and *properties* (also known as *attributes* or *fields*) are listed in the lower part. A third subdivision added for *methods, if they are included*. A special type of class, called an *association*, is used to describe the relationship between two or more CIM classes

Three types of lines connect classes.

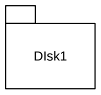


The CIM documents generally follow the convention of using blue arrows for inheritance, red lines for associations and green lines for aggregation. The color-coding makes large diagrams much easier to read but is not a part of the UML standard.

The ends of some associations have numbers (cardinality) indicating the valid count of object instances. Cardinality is expressed either as a single value (such as 1), or a range of values (0..1 or 1..4);"*" is shorthand for 0..n.

Some associations and aggregations are marked with a "W" at one end indicating that the identity of this class depends on the class at the other end of the association. For example, fans may not have worldwide unique identifiers; they are typically identified relative to a chassis.

This document uses two other UML conventions.

The UML Package symbol 

is used as a shortcut representing a group of classes that work together as an entity. For example, several classes model different aspects of a disk drive. After the initial explanation of these objects, a single disk package symbol is used to represent the entire group of objects.

Schema diagrams include all of a profile's classes and associations; the class hierarchy is included and each class is depicted one time in the schema diagram. Instance diagrams also contain classes and associations but represent a particular configuration; multiple instances of an object may be depicted in an instance diagram. An instance may be named with an instance name followed by a colon and a class name (underlined). For example,

Array: ComputerSystem
Switch: ComputerSystem

represent an array and a switch – two instances of <COMPUTER SYSTEM> objects.

6.2 Techniques

6.2.1 CIM Fundamentals

This section provides a rudimentary introduction to some of the modeling techniques used in CIM, and is intended to speed understanding of the SMI-S object model.

Associations as Classes

CIM presents relationships between objects with specialized classes called *associations* and *aggregations.* In addition to references to the related objects, the association or aggregations may also contain domain-related properties. For example, ControlledBy associates a controller and a device. There is a many-to-many cardinality between controllers and devices (i.e., a controller may control multiple devices and multi-path devices connect to multiple controllers); each controller/device connection has a separate activity state. This state corresponds to the AcccessState property of ControlledBy association linking the device and the controller.

Logical and Physical Views

CIM separates physical and logical views of a system component, and represents them as different objects – the "realizes" association ties these logical and physical objects together.

Identity

Different agents may each have information about the same organic object and may need to instantiate different model objects representing the same thing. Access control is one example: a switch zone defines which host device ports may access a device port. The switch agent creates partially populated port objects that are also created by the HBA and storage system agents. The ConcreteIdentity association is used to indicate the associated object instances are the same thing. ConcreteIdentity is also used as a language-independent alternative to multiple inheritance. For example, a FibreChannel port inherits from a generic port and also has properties of a SCSI controller. CIM models this as FCPort and ProtocolController objects associated by ConcreteIdentity.

Extensibility

CIM makes allowances for additional values in enumerations that were not specified in the class Derivation by adding a property to hold arbitrary additional values for an enumeration. This property is usually named OtherXXXX (where XXXX is the name of the enumeration property) and specifying "other" as the value in the enumeration property indicates its use. For an example see the ConnectorType and OtherTypeDescription properties of Slot object in the CIM_Physical MOF. See http://www.dmtf.org/standards/cim/cim_schema_v211/.

Value/ValueMap Arrays

CIM uses a pair of arrays to represent enumerated types. ValueMap is an array of integers; Values is an array of strings that map to the equivalent entry in ValueMap. For example, PrinterStatus (in the CIM_Device MOF) is defined as follows:

```
ValueMap {"1", "2", "3", "4", "5", "6", "7"},
Values {"Other", "Unknown", "Idle", "Printing", "Warm-up,
"Stopped Printing", "Offline"},
```

A status value of 6 means "Stopped Printing". A client application can automatically convert the integer status value to a human-readable message using this information from the MOF.

Return Codes

When a class definition includes a method, the MOF includes Value/ValueMap arrays representing the possible return codes. These values are partitioned into ranges of values; values from 0 to 0x1000 are used for return codes that may be common to various methods. Interoperable values that are specific to a method start at 0x1001; and vendor-specific values may be defined starting at 0x8000. Here's an example of return codes for starting a storage volume.

```
 ValueMap {"0", "1", "2", "4", "5", ".", "0x1000",
"0x1001", "…", "0x8000.."},
 Values {"Success", "Not Supported", "Unknown", "Time-out,
"Failed", "Invalid Parameter", "DMTF_Reserved",
"Method parameters checked - job started",
"Size not supported",
"Method_Reserved", "Vendor_Specific"}]
```

Model Conventions

This is a summary of objects and associations that are common to multiple profiles.

PhysicalPackage represents the physical storage product. PhysicalPackage may be sub-classed to ChangerDevice, but PhysicalPackage accommodates products deployed in multiple chassis.

Producer models asset information including vendor and product names. Product is associated with PhysicalPackage.

SoftwareIdentity models firmware and optional software packages. InstalledSoftwareIdentity associates SoftwareIdentity and ComputerSystem, ElementSoftwareIdentity associates SoftwareIdentity and LogicalDevices (a superclass of devices and ports).

Service models a configuration interface (for example, a switch zoning service or an array access control service). Services typically have methods and properties describing the capabilities of the service. A storage system may have multiple services; for example, an array may have separate services for LUN Masking and LUN creation. A client can test for the existence of a named service to see if the agent is providing this capability.

LogicalDevice (for example, FCPort) is a superclass with device subclasses (like and DiskDrive and TapeDrive) and also intermediate nodes like Controller and FCPort. Each LogicalDevice subclass shall be associated to a ComputerSystem with a SystemDevice aggregation. Due to the large number of LogicalDevice subclasses, SystemDevice aggregations are often omitted in instance diagrams in this specification.

This specification covers many common storage models and management interfaces, but some implementations include other objects and associations not detailed in the specification. In some cases, these are modeled by CIM schema elements not covered by this document. When vendor-specific capabilities are needed, they should be modeled in subclasses of CIM objects. These subclasses may contain vendor-specific properties and methods and vendor-specific associations to other classes.

6.2.2 Modeling Profiles

In addition to modeling SAN components, SMI-S servers shall model the profiles they provide. This information is used two ways:

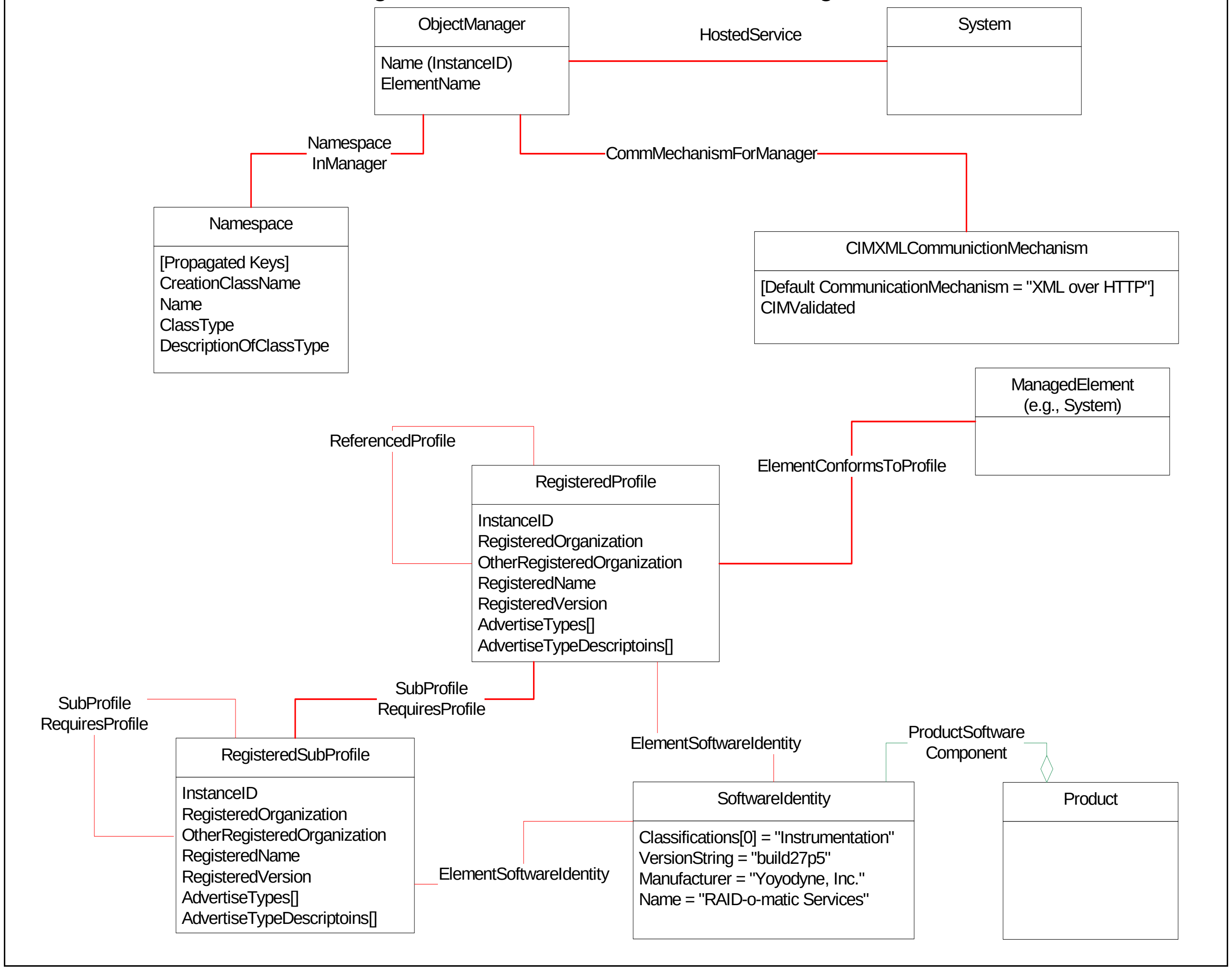
- Clients can quickly determine which profiles are available
- An SLP component can query the SMI-S Server and automatically determine the appropriate SLP Service Template information (see Clause 10:, "Service Discovery", and Table 2, “SLP Properties”)

Table 2: SLP Properties

Property Name	Use
SupportedRegisteredProfiles	Defines the organization defining the profile, the RegisteredProfile and RegisteredSubprofile. Setting this to “SNIA” indicates that one of the SNIA SMI-S profiles applies

A client can traverse the Server Profile in each SMI-S server to see which Profiles (and objects) claim SMI-S compliance.

Figure 10: Server Profile Instance Diagram



The RegisteredProfile describes the profiles that a CIM Server claims are supported. The RegisteredSubprofile is used to define the optional features supported by the system being modeled. A client can traverse the associations in the Server Profile see which Profiles and subprofiles claim SMI-S compliance.

6.2.3 CIM Naming

There may be multiple SMI-S Servers in any given storage network environment. It is not sufficient to think of the name of an object as just the combination of its key values. The name also serves to identify the Server that is responsible for the object. The name of an object (instance) consists of the Namespace path and the Model path. The Namespace path provides access to a specific SMI-S server implementation and is used to locate a particular namespace within a Server. The Model path provides full navigation within the CIM Schema and is the concatenation of the class name and key-qualified properties and values.

The namespace has special rules. It should uniquely identify a SMI-S Server. However, a SMI-S Server may support multiple namespaces. How an implementation defines Namespaces within a SMI-S server is not restricted. However, to easy interoperability SMI-S implementations should manage all objects within a Profile in one Namespace.

6.2.4 Correlatable and Durable Names

6.2.4.1 Overview

Management applications often read and write information about managed objects in multiple CIM namespaces or between CIM and some other storage management namespace. When an object in one namespace is associated with an object in another namespace, each namespace may represent some amount of information about the same managed resource using different objects. A management application understands when objects in different namespaces represent the same managed resource by the use of a unique common identifier, referred to as a “correlatable name”. A correlatable name is designated as a mandatory property for any objects representing managed resources that may be seen from multiple points of view. These durable names are used by management applications for object coordination.

A related concept is referred to as “durability”. Some names may be correlatable at a particular point in time, but may change over time (e.g., a durable name is a hardware-assigned port or volume name and a correlatable, non-durable ID is a DHCP IP address). No name is permanently durable (e.g., even a name derived from hardware may change due to FRU replacement). A client application should assume that a stored durable name remains valid over time where a non-durable may not remain valid over time.

Correlatable names are unique within a defined namespace. In some cases, that namespace is world-wide; requiring compliance to standards defined by a naming authority. In other cases, the namespace is the hosting system or some set of connected systems (e.g., operating system device names are unique to the containing host).

A name may be expressed in different formats (e.g., numeric value are sometimes displayed as decimal or hexadecimal, the hexadecimal value sometimes has a leading “0x” or a trailing “h”). To assure interoperability, mandatory formats are specified by this standard.

A necessary technique associated with correlatable names involves the use of CIM properties that describe the format or namespace from which the name is derived. CIM key-value combinations are unique across instances of a class, but CIM does not fully address cases where different types of identifiers are possible on different instances of an object. It is therefore necessary to ensure that multiple sources of information about managed resources use the same approach for forming correlatable names whenever different types of identifiers are possible.

When different types of identifiers are possible, the profile specifies the possible name formats and namespaces for durable and correlatable IDS, the preferred order that each implementation should use if multiple namespaces are available, and the related properties that a client uses to determine the namespace.

Correlatable, durable names are mandatory for the following objects:

- SCSI Logical Units (such as storage volumes or tape drives) that are exported from storage systems
- External Ports on hosts and storage devices
- Fibre Channel ports on interconnect elements
- Fibre Channel fabric (modeled as AdminDomain)
- ComputerSystem objects that server as top-level systems for all SMI-S profiles
- Operating System Device Names

CIM keys and correlatable names are not tightly coupled. For some classes, they may be the same, but this is not mandatory as long as all correlatable names are unique and management applications are able to determine when objects in different namespaces are providing information about the same managed resource.

The common types of information used for names include the SCSI Device Identifiers from the Identification Vital Product Data page (i.e., VPD page 83h), Fibre Channel Name_Identifiers (i.e., World Wide Names), Fully Qualified Domain Names, and IP Address information. See 6.2.4.2, "Guidelines for SCSI Logical Unit Names", 6.2.4.3, "Guidelines for Port Names", and 6.2.4.4, "Guidelines for Storage System Names" for general information on the advantages and disadvantages of certain types of names. The details for each class requiring durable correlatable names are provided in the profiles subclauses of this document.

6.2.4.2 Guidelines for SCSI Logical Unit Names

The preferred logical unit identifier is returned from a SCSI INQUIRY command in VPD page 83h.

Note: Legacy systems may lack correlatable names as SCSI standards prior to SAM-3 and SPC-3 did not clearly define logical unit names, however this has been clarified to be logical unit names and recent systems have converged in compliance.

The Unit Serial Number VPD page (i.e., SCSI Inquiry VPD Page 80h) returns a serial number, but the SPC-3 standard allows this either be a serial number for a single logical unit or a serial number of the target device. There's no mechanism to discover which approach the device is using. If a client is not coded to understand which products provide per-logical unit or per-target serial numbers, then it should not use the Unit Serial Number VPD page as a logical unit name.

The Identification Vital Product Data page (i.e., VPD page 83h) returns a list of identifiers with metadata describing each identifier. The metadata includes:

- Code Set (i.e., binary verses ASCII)
- Association (i.e., indicates the SCSI object to which the identifier applies (e.g., for a logical unit, port, or target device))
- Type (i.e., the naming authority for identifiers of the structure of information about target ports)
- Protocol Identifier (i.e., indicates the SCSI transport protocol to which the identifier applies)

To identify a logical unit name the Association shall be set to zero. The preferred Types for logical units are 3 (i.e., NAA), 2 (i.e., EUI), and 8 (i.e., SCSI Name). However type 1 (i.e., T10) is allowed. If the code set in the inquiry response indicates the identifier is binary, the CIM representation is hexadecimal-encoded.

6.2.4.3 Guidelines for Port Names

The following is a list of optimal names for ports based on the transport type:

- Fibre Channel ports use Port World Wide Names (i.e., FC Name_Identifier)
- iSCSI has three types of ports:
    - The combination of IP address and TCP port number serve as the primary correlatable name for iSCSI target ports. Note that this information is stored in two separate properties and hence there is no single correlatable name.
    - The logical element (iSCSIProrotolEndpoint) that represents the SCSI port. The SCSI logical port shall be named with an iSCSI name.
    - The underlying physical ports (typically Ethernet ports). Ethernet ports names shall use the MAC address.

- Parallel SCSI (SPI) and ATA ports typically do not have names, they are identified by a bus-relative address typically set with jumpers. In configurations where these drives are not shared by multiple hosts, the host-relative name acts as the name.
- CIM port classes do no include NameFormat; the appropriate format is determined by the transport implied by the port subclass.

SCSIProtocolEndpoint represents SCSI protocol running through a port. In many cases, there is one-to-one mapping between SCSIProtocolEndpoint and some subclass of LogicalPort and the name requirements are identical. For iSCSI, there many be multiple Ethernet ports per SCSIProtocolEndpoint instance. The IP address and TCP port number are modeled in IPProtocolEndpoint and TCPProtocolEndpoint. iSCSIProtocolEndpoint Name holds the iSCSI initiator or target name.

6.2.4.4 Guidelines for Storage System Names

Each profile has a ComputerSystem or AdminDomain instance that represents the entire system. There are a variety of standard and proprietary names used to name storage systems. Unlike SCSI logical units and ports, there is no particular name format in common use. There are advantages and disadvantages to certain types of names.

IP addresses have an advantage in human recognition; (e.g., administrators are accustomed to referring to systems by their IP addresses). The downsides are that IP addresses are not necessarily durable (e.g., DHCP) are not necessarily system-wide (e.g., some storage systems have multiple network interfaces), and are not necessarily unique (e.g., NAT allows the same IP address to be used in multiple network zones).

Full Qualified Domain Names are friendlier than IP addresses and may fix the durability issue of IP addresses (e.g., a host name may be constant even when the IP address changes). But storage systems do not necessarily have access to their network names. Network names are typically handled through a central service such as DNS. When a client application opens a connection to a remote system, it asks the local system to resolve the name to an IP address, the local system redirects the request to the DNS server, the IP address is returned and the client application opens the connection. If the remote system is the storage system, this sequence requires the DNS server to know about the storage system, but not vice-versa. A storage system is only required to know about DNS if software on the storage system acts as a network client using host names. And, like IP addresses, a storage system may have several network interfaces with different FQDNs.

Transport-specific names are specific to a particular storage transport (e.g., Fibre Channel or iSCSI). There are some good standard names (e.g., FC platform names or iSCSI Network Entity names). The disadvantage of transport-specific names is that they are not able to be consistently used on storage systems supporting multiple transports or in configurations with transport bridges (e.g., a client may have no mechanism to issue FC commands to an FC device behind an FC/iSCSI bridge).

SCSI target names solve the transport-specific issue. Before the SAM-3 and SPC-3 standards there was not a standard SCSI system name, however with SPC-3, the Identification Vital Product Data page association value 2 was defined for a target name. At this time, the SPC-3 standard is too new to be in common use. Most storage systems include some vendor-specific way to get a target name, but client is not able to use these names without specific knowledge of the vendor-specific interface.

At this time, no single storage system name format is in common use. The best approach is for implementations to expose several names, along with information that tells the client how to interpret the name. The OtherIdentifyingInfo and IdentifyingDescriptions array properties of ComputerSystem provide the list of names and interpretations. However, IdentifyingDescriptions is not an enumerated type; and as a result, any string is valid from a CIM perspective.

6.2.4.5 Standard Formats for Correlatable Names

Correlatable names shall be used and formatted consistently. Storage volume names are more complex that other element names (i.e., the same format may be used in different namespaces). For example several common INQUIRY Vital Product Data page names use the IEEE NAA format and as a result a client is not able to correlate names from different namespaces.

6.2.4.5.1 Standard Formats for Logical Unit Names

For disks and arrays, multiple name formats are in common use. Table 3 specifies standard formats for storage volume names.

Table 3: Standard Formats for StorageVolume Names

Description	Format property and value(valuemap)	Format of Name
SCSI VPD page 83 type 3, Association 0, NAA 0101b	NameFormat = NAA(9), NameNamespace = VPD83Type3(1)	NAA name with first nibble of 5. Recommended format (8 bytes long) when the ID is directly associated with a hardware component. Formatted as 16 un-separated upper case hex digits (e.g., '21000020372D3C73')
VPD page 83, type 3h, Association=0, NAA 0110b	NameFormat = NAA(9), NameNamespace= VPD83Type3(1)	NAA name with first nibble of 6. Recommended format (16 bytes long) when IDs are generated dynamically. Formatted as 32 un-separated upper case hex digits.
VPD page 83, type 3h, Association=0, NAA 0010b	NameFormat = NAA(9), NameNamespace = VPD83Type3(1)	NAA name with first nibble of 2. Formatted as 16 un-separated upper case hex digits
VPD page 83, type 3h, Association=0, NAA 0001b	NameFormat = NAA(9), NameNamespace = VPD83Type3(2)	NAA name with first nibble of 1. Formatted as 16 un-separated upper case hex digits
VPD page 83, type 2h, Association=0	NameFormat = EUI64(10), NameNamespace = VPD83Type2(3)	Formatted as 16, 24, or 32 un-separated upper case hex digits
VPD page 83, type 1h, Association=0	NameFormat = T10VID(11), NameNamespace = VPD83Type1(4)	Formatted as 1 to 252 bytes of ASCII.
VPD page 80, serial number	NameFormat = Other(1), NameNamespce = VPD80(5)	Only if serial number refers to logical units rather than the enclosure. 1-252 ASCII characters
Concatenation of Vendor, Product, SerialNumber	NameFormat = SNVM(7), NameNamespace = SNVM(7)	3 strings representing the vendor name, product name within the vendor namespace, and serial number within the model namespace. Strings are delimited with a '+' and spaces are included. Vendor and Product are fixed length: Vendor ID is 8 bytes, Product is 16 bytes. SerialNumber is variable length and may be up to 252 bytes in length. If one of these fields contains a plus sign, it shall be escaped with a backslash ('\+'). The concatenation is done to provide world-wide uniqueness; clients should not parse this name.

Table 3: Standard Formats for StorageVolume Names

Description	Format property and value(valuemap)	Format of Name
FC Node WWN	NameFormat = NodeWWN(8) Name-Namespace = NodeWWN(6)	16 un-separated upper case hex digits (e.g., '21000020372D3C73')

Storage volumes may have multiple standard names. A page 83 logical unit identifier shall be placed in the Name property with NameFormat and Namespace set as specified in Table 3. Each additional name should be placed in an element of OtherIdentifyingInfo. The corresponding element in IdentifyingDescriptions shall contain a string from the Values lists from NameFormat and NameNamespace, separated by a semi-colon. For example, an identifier from SCSI VPD page 83 with type 3, association 0, and NAA 0101b - the corresponding entry in IdentifyingDescriptions[] shall be “NAA;VPD83Type3”.

For other types of devices, the logical unit name shall be in the Name property; NameFormat and NameNamespace are not valid properties of these other device classes.

6.2.4.5.2 Standard Formats for Port Names

Table 4, “Standard Formats for Port Names” specifies standard formats for port names

Table 4: Standard Formats for Port Names

Name Type	Name Format	Details
An IP interface's MAC	Network Port Permanent Address property; no corresponding format property	Six upper case hex bytes, bytes are delimited by colons ':'
World Wide Name (i.e., FC Name_Identifier)	FCPort Permanent Address property; no corresponding format property	16 un-separated upper case hex digits (e.g., '21000020372D3C73')
	SCSIProtocolEndpoint Name property; ConnectionType = 2 (Fibre Channel)	16 un-separated upper case hex digits (e.g., '21000020372D3C73')
Parallel SCSI Name	SPI Port Name property; no corresponding format property	String - platform-specific name representing the name. Note that this name is only correlatable relative to the system containing the port.
	SCSIProtocolEndpoint Name property; ConnectionType = 3 (Parallel SCSI)	String - platform-specific name representing the name.

Note that iSCSI Network Portals do not have a single correlable name. The combination of IPProtocolEndpoint IPv4Address or IPv6Address and TCPProtocolEndpoint PortNumber uniquely identifies the network portal, but since these are two properties, they do not form a correlatable name.

6.2.4.5.3 Standard Formats for Fabric Names

A fabric is modeled as AdminDomain. AdminDomain.Name shall hold the fabric name (i.e., WWN) and AdminDomain.NameFormat shall be set to “WWN”. AdminDomain.Name shall be formatted as 16 unseparated upper case hex digits.

6.2.4.5.4 Standard Formats for Storage System Names

Due to the limited list of possible formats, the Name property is not considered an essential identifier for SMI-S. SMI-S clients should use OtherIdenfyiingInfo property as described in Table 5.

Providers shall supply at least one Durable or Correlatable Name as an element in the IdentifyingDescriptions[] array. The corresponding array elements of OtherIdentifyingInfo[] shall include a value from Table 5 for all elements of IdentifyingDescriptions[]. The elements in the IdentifyingDescriptions array are strings and may contain white space between words. Whenever white-space appears, it shall consist of a single blank; other white-space characters and multiple consecutive blanks shall not be used.

At least one of the values in IdentifyingDescriptions[] shall be something other than “SCSI Vendor Specific Name” or “Other Vendor Specific Name”.

OtherIdentifyingInfo[0] should be assigned the most preferable name by the instrumentation.

In all cases, if the name is returned to the instrumentation in binary, the corresponding entry in OtherIdentifyingInfo holds a hexadecimal-encoded representation of the value returned. Standard names defined in binary are called out in Table 5.

Other ComputerSystem properties should be set as follows:

Name is a CIM key and shall be unique for ComputerSystem instances within the CIM namespace. SMI-S clients should not assume Name is either durable or correlatable.

NameFormat is an enumerated type describing the Name property. Only a few of the defined values are appropriate for storage systems. Use “IP” if Name is derived from an IP address of Fully Qualified Domain Name. Use “HID” if Name is derived from a hardware ID. Use “OID” if Name is a unique ID determined by some unique ID generating logic.

ElementName is a friendly name; SMI-S clients should not assume that ElementName is unique, correlatable, or durable since a customer may provide the same info for multiple systems.

Table 5: Standard Formats for Storage System Names

IdentifyingDescriptions [x] value	Description		Format of OtherIdentifyinginfo[x]
T10 Target Name Type 1	An identifier from a Identification Vital Product Data page response with Association equal to 2.	Type 1 (T10)	1 to 252 bytes of ASCII
T10 Target Name Type 2		Type 2 (EUI)	16, 24, or 32 un-separated upper case hex digits (e.g., '21000020372D3C73')
T10 Target Name Type 3		Type 3 (NAA)	16 or 32 un-separated upper case hex digits (e.g., '21000020372D3C73')
T10 Target Name Type 8		Type 8 (SCSI Names)	iSCSI Names (see 6.2.4.9)
T11 FC-GS-4 Platform Name	A platform name as defined in T11 FC-GS-4 standard		Up to 508 hex digits (254 bytes) as specified by T11 FC-GS-4 subclause on Platform Name. Format as unsep-arated as hex digits . Platform Name Format Byte shall be included.
T11 RNID Name	An RNID names as defined in T11 FC FS standard.		32 unseparated hex digits.
iSCSI Network Entity Name	An iSCSI Network Entity name.		iSCSI Names (see 6.2.4.9)
Ipv4 Address	An IP V4 name		Four decimal bytes delimited with dots ('.')

Table 5: Standard Formats for Storage System Names (Continued)

IdentifyingDescriptions [x] value	Description	Format of OtherIdentifyinginfo[x]
Ipv6 Address	An IP V6 name	'x:x:x:x:x:x:x:x', where the 'x's are the uppercase hexadecimal values of the eight 16-bit pieces of the address. Examples: 'FEDC:BA98:7654:3210:FEDC:BA98:7654:3210', '1080:0:0:0:8:800:200C:417A' Leading zeros in individual fields should not be included and there shall be at least one numeral in every field. (This format is compliant with RFC 2373.) In addition, omitting groups of zeros or using dotted decimal format for an embedded IPv4 address is prohibited.
Fully Qualified Domain Name	A fully qualified domain name.	A legal DNS name (fully qualified) consisting of strings delimited by periods.
Node WWN	The Fibre Channel Node WWN. The provider shall assure that the same Node WWN shall be available through all FC ports within a target device.	16 un-separated upper case hex digits (e.g., '21000020372D3C73')
T10 Unit Serial Number VPD page	SCSI Inquiry VPD page 80 response is a serial number This name may be unique for a specific logical unit or for the target (e.g., storage system). These names are only valid if the instrumentation is certain that all logical units in a system return the same value. Since there is no mechanism to test whether the value is unique per target or per logical unit, this value is not interoperably correlatable and should not be used	1-252 ASCII characters
SCSI Vendor Specific Name	This is a name accessible through a vendor-specific SCSI command. A client with a priori knowledge may be able to correlate this based on vendor and Product IDs.	unknown
Other Vendor Specific Name	This is a name accessible through some non-SCSI vendor-specific interface.	unknown

6.2.4.5.5 Operating System Device Names

Each operating system has different conventions for naming devices. Many operating systems provide multiple names for the same device instance. In this version of the specification, operating system device name formats are recommended.

The case of names specified by operating system interfaces shall be preserved.

Operating system device names are unique within the namespace of the scoping system and are not unique between systems.

Table 6 specifies the format for names of tape devices.

Table 6: Standard Operating System Names for Tape Devices

Operating System	Format	Notes
AIX	/dev/rmtX	X represents a hexadecimal number and may be more than one character
HP-UX	/dev/rmn/Xm	X represents a hexadecimal number and may be more than one character
Linux	/dev/stX	X represents one or two lower case alphabetic characters
Solaris	/dev/rmt/Xn	X represents a hexadecimal number and may be more than one character
WIndows	\\.\\TAPEX	X represents a decimal number

Some operating systems treat disk partitions as virtual devices; applications operate on partitions as if they were disks. The model requires two classes for each partition, LogicalDisk and GenericDiskPartition. Other operating systems allow applications to operate on the entire disk without partitions. Linux allows both.

Table 7 specifies the format for LogicalDisk.Name of disk partitions

Table 7: LogicalDisk.Name for disk partitions

Operating System	Format	Notes
Linux	dev/sdXY or /dev/hdXY	where X represents one or two lower case alphabetic characters and Y represents an integer between 1 and 15
Solaris	/dev/dsk/cXtXdXsX	X represents one or two lower case alphabetic characters
WIndows	C: or the file name of mount point	C represents an uppercase letter

Table 8 specifies the format for GernericDiskParition.Name and DeviceId properties for disk partitions

Table 8: GenericDiskParittion.Name for disk partitions

Operating System	Format	Notes
Linux	sdXY or hdXY	X represents one or two lower case alphabetic characters
Solaris	/dev/dsk/cXtXdXsX	where X represents one or two lower case alphabetic characters and Y represents an integer between 1 and 15
WIndows	Disk #X, Partition #X	X represents a decimal digit

Table 9 specifies the format for LogicalDisk.Name for unpartitioned disks.

Table 9: Standard Operating System Names for Unpartitioned DIsks

Operating System	Format	Notes
AIX	/dev/hdiskX	X represents a hexadecimal number and may be more than one character
HP-UX	/dev/dsk/cXtYdZ	X , Y, and Z represents hexadecimal number and may be more than one character in length
Linux	/dev/sdX or /dev/hdX	X represents one or two lower case alphabetic characters
Windows	\\.\PHYSICALDRIVEx	x represents a a decimal number and may be more than one character

6.2.4.6 Case Sensitivity

Names and NameFormats are case sensitive and the cases provided in Table 9 shall be used If not otherwise specified, uppercase should be used.

6.2.4.7 Testing Equality of correlatable Names

The implementation shall only compare objects of the same class or parent class. For objects that do not require the use of additional properties, a simple direct comparison is sufficient, providing the format for the mandatory correlatable name as identified in this section or the specific profile is adhered to.

For objects that do require the use of additional properties (e.g., NameFormat), the correlatable names of objects representing the same entity should compare positively, negatively, or indicate clearly when a comparison is ambiguous.

- if the two objects have the same NameFormat and Name, then they refer to the same resource
- if the two objects have the same NameFormat and different Names, then they refer to different resources
- if the two objects have different NameFormats, whether the Names are the same or different, then it is unknown whether they refer to the same resource

This reduces the possibility that a match is missed by a string equals comparison simply because of an incompatibility of formats rather than non-equality of the data.

6.2.4.8 Operating System Device Names

6.2.4.9 iSCSI Names

The iSCSI standards define three text formats for names that apply to various iSCSI elements. The three formats are: iSCSI qualified name (iqn), IEEE Extended Unique Identifier (eui), and ANSI T10 NAA. The format is included in the name as a three-letter prefix. The three formats are explained in more detail.

The iSCSI qualified name (iqn) format is defined in [iSCSI] and contains (in order):

1 - The string “iqn.”

2 - A date code specifying the year and month in which the organization registered the domain or subdomain name used as the naming authority string.

3 - The organizational naming authority string, which consists of a valid, reversed domain or subdomain name.

Optionally, a ':', followed by a string of the assigning organization's choosing, which shall make each assigned iSCSI name unique.

Figure 11: "iSCSI Qualified Names (iqn) Examples" contains examples of iSCSI-qualified names that may be generated by “EXAMPLE Storage, Inc.”

Figure 11: iSCSI Qualified Names (iqn) Examples

```
           Organizational      Subgroup Naming Authority
                   Naming      and/or string Defined by
Type  Date           Auth      Org. or Local Naming Authority
+--++-----+ +---------+ +--------------------------------+
|  ||     | |         | |                                |
iqn.2001-04.com.example:diskarrays-sn-a8675309
iqn.2001-04.com.example
iqn.2001-04.com.example:storage.tape1.sys1.xyz
iqn.2001-04.com.example:storage.disk2.sys1.xyz
```

The IEEE Registration Authority provides a service for assigning globally unique identifiers [EUI]. The EUI-64 format is used to build a global identifier in other network protocols.

The format is "eui." followed by an EUI-64 identifier. Figure 12: "iSCSI EUI Name Examples" contains an example.

Figure 12: iSCSI EUI Name Examples

```
Type  EUI-64 identifier (ASCII-encoded hexadecimal)
+--++--------------+
|  ||              |
eui.02004567A425678D
```

Type "naa." - Network Address Authority

The ANSI T10 FC-FS standard defines a format for constructing globally unique identifiers [FC-FS] referred to as an Network Address Authority (NAA) format. The iSCSI name format is "naa." followed by an NAA identifier (ASCII-encoded hexadecimal digits).

Figure 13: "iSCSI 64-bit NAA Name Examples" contains an example of an iSCSI name with a 64-bit NAA value: type NAA identifier (ASCII-encoded hexadecimal)

Figure 13: iSCSI 64-bit NAA Name Examples

```
+--++--------------+
|  ||              |
naa.52004567BA64678D
```

Figure 14: "iSCSI 128-bit NAA Name Examples" contains an example of an iSCSI name with a 128-bit NAA value: type NAA identifier (ASCII-encoded hexadecimal)

Figure 14: iSCSI 128-bit NAA Name Examples

```
+--++------------------------------+
|  ||                              |
naa.62004567BA64678D0123456789ABCDEF
```

6.3 Health and Fault Management

6.3.1 Objectives

Health and Fault Management is the activity of anticipating or detecting failures through monitoring the state of the storage network and its components and intervening before services can be interrupted. A service in this case is the realization of storage through several interconnected devices connected, configured for a dedicated purpose. The purpose is the delivery of software application functionality in support of some business function.

6.3.2 Overview

- Express states and statuses with standard meanings.
- Define the use of comprehensive error reporting in determining the type, category, and source of failures.
- Define the quality associated with errors rather than qualities.
- Define explicit failure scopes rather than requiring HFM enabled application to construct them.

6.3.3 Terms

Error:

An unexpected condition, result, signal or datum. An error is usually caused by an underlying problem in the system such as a hardware fault or software defect. Errors can be classified as correctable (recoverable) or uncorrectable, detectable or undetectable.

Fault:

A problem that occurs when something is broken and therefore not functioning in the manner it was intended to function. A fault may cause an error to occur.

Fault Region:

Many devices or applications can attempt to fix themselves upon encountering some adverse condition. The set of components which the device or application can attempt to fix is called the Fault Region. The set may include part or all of other devices or applications. Having the Fault Regions declared helps a HFM application, acting as a doctor, to do no harm by attempting to interfere and thereby adversely effect the corrective action being attempted.

Health and Fault Management (HFM):

Health and Fault Management is the activity of anticipating or detecting debilitating failures through monitoring the state of the storage network and its components and intervening in before services can be interrupted. A service in this case is the realization of storage utilization through several interconnected devices connected, configured for a dedicated purpose. The purpose is the delivery of software application functionality in support of some business function.

Operational status:

These values indicate the current status(es) of the element. Various operational statuses are defined (e.g. OK, starting, stopping, stopped, In Service, No Contact).

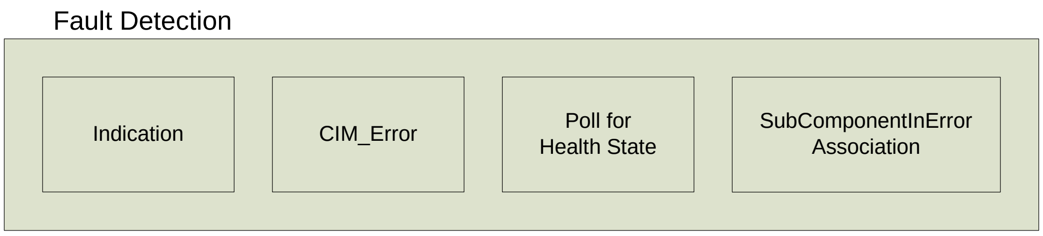
Health State:

These values indicate the current health of the element. This attribute expresses the health of this element but not necessarily that of its subcomponents.

6.3.4 Description of Health and Fault Management

The goal of effective administration requires devices and applications that comprise storage services to report their status and the nature of their errors in standard terms. These terms need to be understandable by a client without device specific knowledge.

Figure 15: Basic Fault Detection



There are four basic ways for a SMI-S client to detect an error or fault condition. These are:

- Health state and Operational status - Polling.
- Error - Standard errors returned from CIM operations.
- Indications - Subscribe for and receive asynchronous Indications.
- Fault Regions (experimental) - Walk the CIM model looking for RelatedElementCausingError associations.

6.3.4.1 Operational Status and Health State (Polling)

Operational Status and Health State are the two properties that will be used to monitor health. These two properties could convey very different statuses and may at times be related or independent of each other. For example, you may have a disk drive with the Operational Status of “Stopped” and the HealthState of 0 (expired) or 100 (excellent). Now the reason the disk drive is stopped could vary from the fact that it had a head crash (HealthState = 0) to the situation where it was stopped for the routine maintenance (HealthState = 100).

Table 10 is an example of how HealthState can disambiguate health for a for a disk drive, various values for OperationalStatus and HealthState:

The table shows, for a disk drive, various possible values for OperationalStatus and HealthState. Note that there are many cases not shown.

Table 10: OperationalStatus for Disk Drive

OperationalStatus	Description	HealthState	Description	Comment
2	OK	5	OK	Everything is fine
2	OK	10	Degraded/Warning	Some soft errors
3 or 2	Degraded or Predicted Failure	15	Minor Failure	Many soft errors
3 or 2	Degraded or Predicted Failure	20	Major Failure	Some hard errors
3	Degraded	10	Good	A subcomponent has failed (no data loss)
10	Stopped	5	OK	Drive spun down normally
10	Stopped	30	Non-recoverable Error	Head crash
8	Starting	10	Degraded/Warning	Will update HealthState once fully started
4	Stressed	5	OK	Too many I/O in progress, but the drive is fine.
15	Dormant	5	OK	The drive is not needed currently

The property OperationalStatus is multi-valued and more dynamic. It tends to emphasize the current status and potentially the immediate status leading to the current status; whereas, the property HealthState is less dynamic and tends to imply the health over a longer period of time. Again, in the disk drive example, the disk drive's operational status may change many times in a given time period. However, in the same time period, the health of the same drive may not change at all.

6.3.4.2 Standard Errors and Events

Standardization of error and events are required so that the meaning is unambiguous and is given to comparisons.

Error and Alert indications

HFM clients shall not be required to be embodied with specific knowledge of the devices and applications in order to derive the quality of the error from the datum. The device and application shall express the quality of the error rather than the quantity interpreted with *a priori* knowledge to determine that error condition is present. For example, a device needs to express that it is too hot rather than requiring the HFM enabled application to determine this from the temperature datum and device specific knowledge of acceptable operating conditions.

Standard errors are defined for each Profile / Subprofile. The definitions will be contained in the profiles / subprofiles. Standard errors are not the only error codes that can be returned, but are the only codes that a generic client will understand.

6.3.4.3 Indications

Indications are asynchronous messages from CIM servers to clients. A client must register for them. Each SMI-S profile/subprofile contains lists of indication filters that clients use to indicate the information it is interested in. The message itself is defined in the SMI-S indication subprofile.

6.3.4.4 Event Correlation and Fault Containment

Automation will require that an error arising through control and configuration activities, as a side effect of them, or by failures caused by defects can be directly correlatable. Error categories like network cabling failures or network transmission errors will help organize the types of error that can be produced. Standard errors, like impending disk media failure, will be required as well.

Once the errors have been collected and correlated, the HFM enabled application can produce an impact list sorted by likelihood. Some of the error correlation can be determined by the common

affect through the manifestation of the RelatedElementCausingError association to be described later. The alerts themselves can report its correlation with other alerts.

Potential faults can then be derived from errors for each component. Deriving such a list may require a dialog between the HFM enabled application and the device or application in question such that the HFM enabled application is assisted in the production of the list.

If permitted, then control and configuration operations may be executed to contain the fault. The pallet of these operations will be those operations already available through SMI-S. However, special operations may arise from the HFM design work as well. Fault containment will include the reconfiguration of the storage service with alternative components, leaving failing components or interconnections isolated.

Define the use of comprehensive error reporting in determining the type, category, and source of failures

Much like a physician, the HFM enabled application is notified or consulted when symptoms appear. The HFM enabled application then develops a prognosis based on the manifestation of the ailment. At times, the HFM enabled application will perform diagnostic procedures. The end result of the process is to produce a list of possible causes, ranked by probability, and associated recommended procedures.

Also like a doctor, the HFM enabled application will settle for enabling the patients to heal themselves. That is the HFM enabled applications cannot be expected to heal the device in all cases. A significant portion of all possible corrective actions will require the intervention of people or device unique knowledge.

The simplified state diagram shown in Figure 16: "Health Lifecycle" follows the fault mitigation life cycle for HFM.

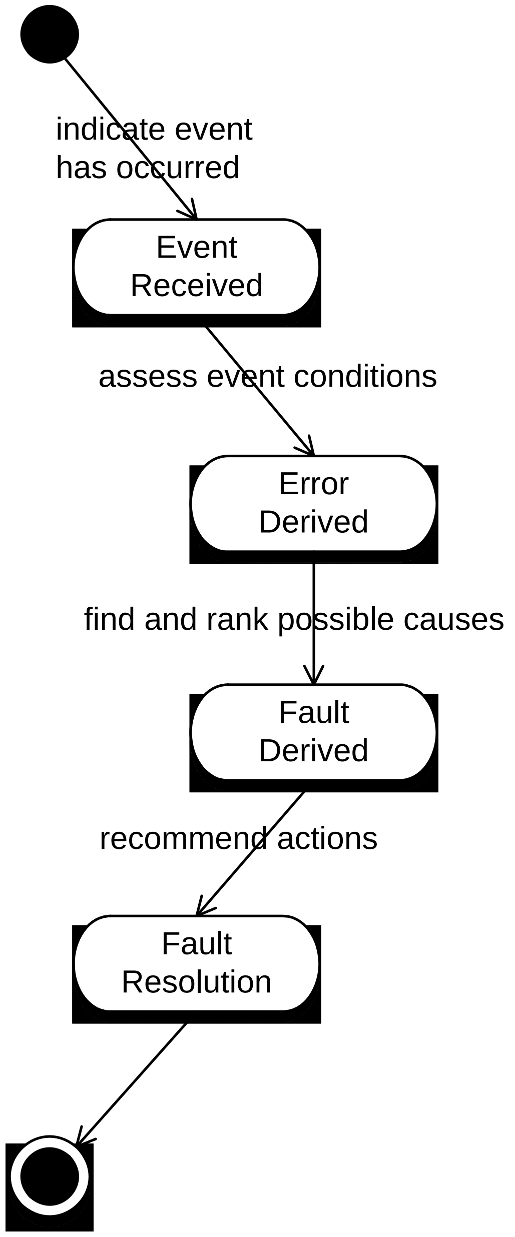
The device or application manifests an event, either by a state change, error returned from a WBEM operation, or an alert indication.

The event is recognized by the HFM enabled application and accessed by the HFM enabled application. It may be that the event indication does the represent the existence of an error. An error condition may be heralded by a single or multiple events occurring in some order. The process of examining and characterizing event as errors is called error handling.

Once it is determined that an error condition is present, then possible causes are sought and ranked by likelihood. The causes themselves describe a potential problem or fault with the component in question.

Alternatively, the device or application may report the fault directly, through an alert indication, optionally with recommended actions.

Figure 16: Health Lifecycle

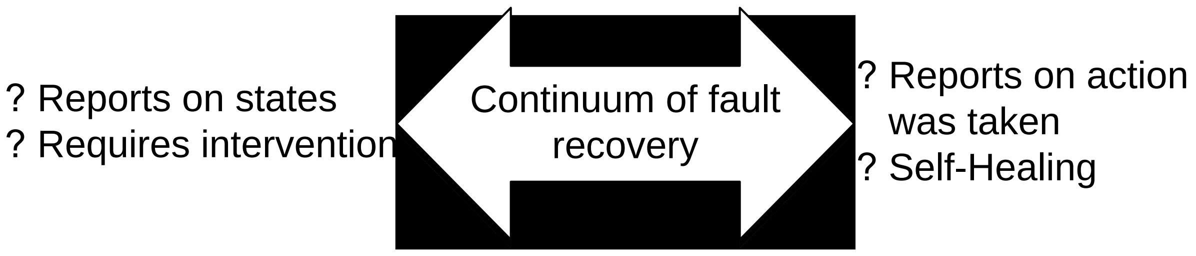


Fault resolution may not require the intervention of an operator or field technician. It is these faults that can be handled entirely by the HFM enabled application. Otherwise, the HFM enabled application can not actively participate in whole fault resolution life cycle. In this case, the HFM enabled application would wait for the end state of fault resolution to come to being before ending its fault mitigation exercise.

Faults are contained and components repaired or replaced. The instructions to the HFM enabled application for what can be done to repair the fault are the recommended actions. Fault Containment includes fencing off the faulty component and maintaining the service. To be minimally effective, the HFM enabled application contains the fault. The repair may or may not be done with human intervention.

The devices and application that comprise a storage system have themselves some level of self diagnostics and report functionality.

Figure 17: Continuum



There is a range of ability of devices and applications to recover from failures and to report on the error recovery actions taken. The variance of capabilities for device and applications can be plotted on a continuum. At one end of continuum, the device or application recognizes a fault condition and takes action, reporting on the action taken and any further action required to service it. At the other end of the continuum, the device can only report on that states and requires intervention both in the detection of fault conditions and taking corrective action.

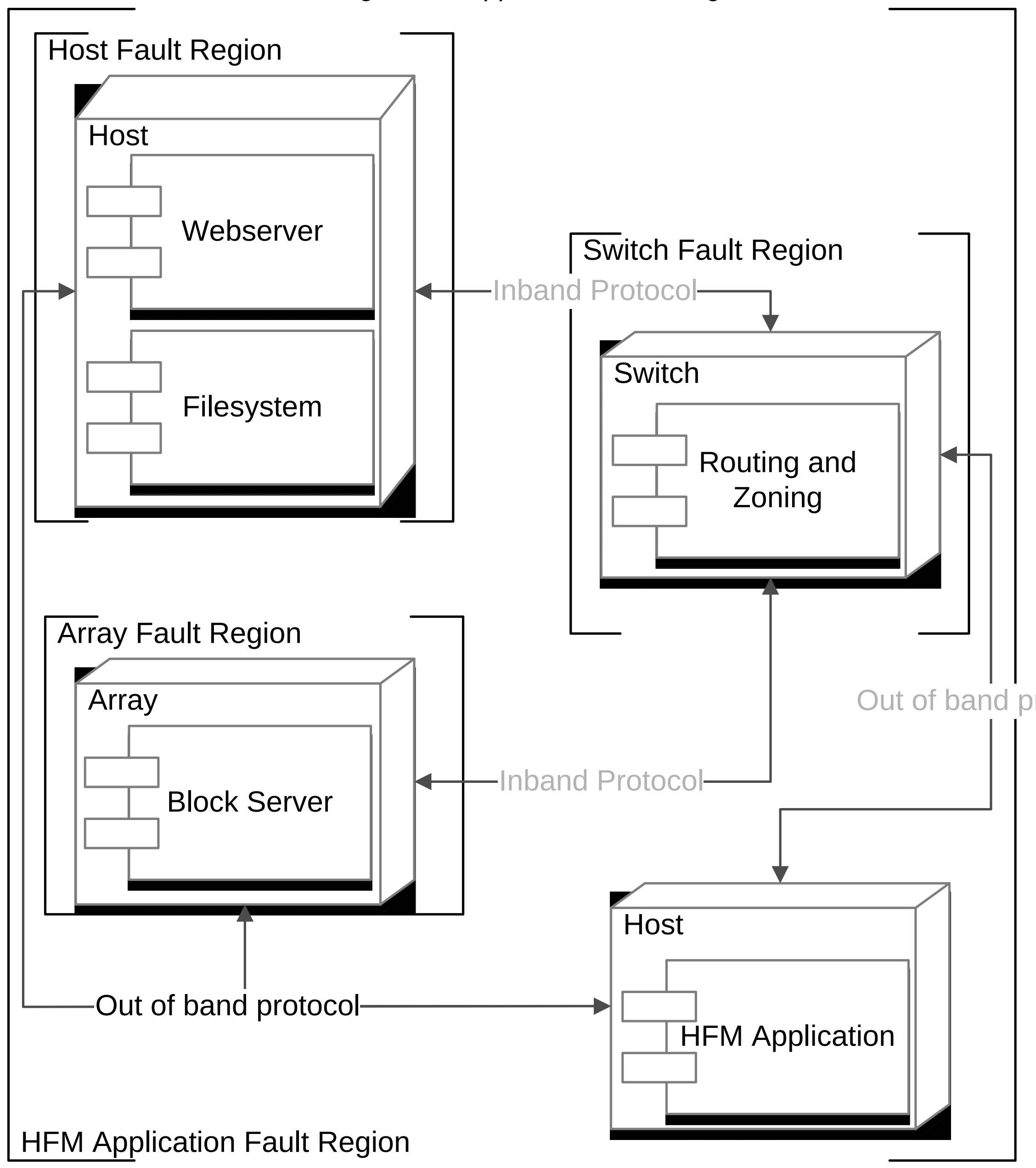
There are limits to what an HFM enabled application can do. Obviously, if the device or application can not report states, errors and alerts in a standard way or can not report this data at all, then there is little an external implementation can do.

However, few, if any, of these devices and applications can monitor and correct the service as a whole. It is for this reason, the HFM implementation is needed to augment the effectiveness of the administrator.

6.3.4.5 Fault Regions

A scope can be applied to the effect of errors and the associated fault. A fault may affect a component, a device or application, storage service, or all the above. This scope defines the area of influence for fault containment. For example, the device itself may monitor its components and perform fault mitigation on its own. The plot of components whose errors are handled by a given fault mitigation entity is the fault region. The scope of effect of this fault region shall be defined.

Figure 18: Application Fault Region



Error handling is initiated by the interception of error events. For example, a switch may recognize the failure of one it ports and reroute traffic to a working port. In this case, the fault region is defined as the switch itself. If the failure event is publicly consumable, other fault mitigation entities can also handle the error as well. The failure of a drive may be mitigated one way in the array fault region and mitigated differently in the HFM enabled application fault region. For example, the array fault mitigation entity can bring a volume off line if the failure of the disk brings the set of disks below the minimum required for quorum. At the same time, the HFM enabled application can reconfigure the storage service to create a replacement volume and then restore the failed volume's data from backup.

The HFM enabled application is one of the several possible storage network scope fault mitigation entities. As discussed previously, this broad scope is necessary to mitigate faults where the faults cannot be entirely mitigated by the storage device or application alone. It is necessary that fault mitigation entities like the HFM enabled application can observe the activities of the fault mitigation entities contained within their fault regions such that they do no harm. Device or application should express what error conditions are to be handled inside their own fault domain and how an HFM enabled application can detect that such fault containment is occurring. State changes on components may BE sufficient representation of these activities.

In general, the HFM enabled application fault region mitigation may not necessarily include the same actions that the host, switch, or array may take to fix them.

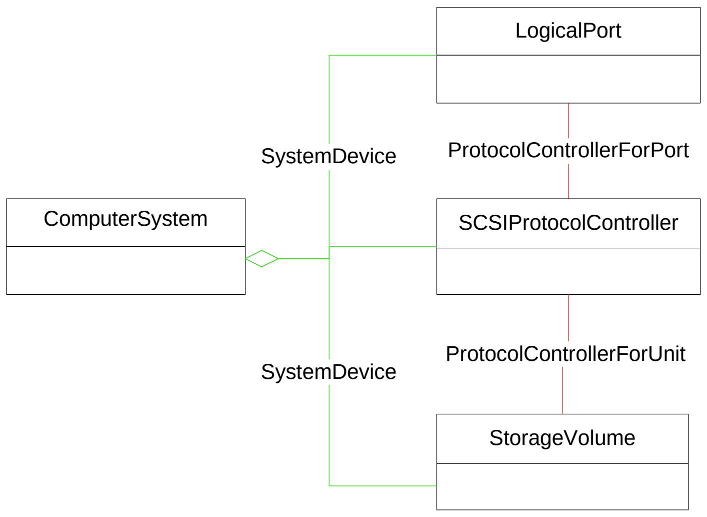
EXPERIMENTAL

6.3.4.6 Examples

6.3.4.6.1 Array Example

The scenario presented is related to a storage array that contains one or many ports. A port is off-line. This port effects the serving of a volume to a host.

Figure 19: Array Instance



<u>Indication</u>

An AlertIndication is produced by the array notifying the HFM enabled application of the failure. The indication reports the Object Name of the ProtocolController that has failed through its AlertingManagedElement property. When storage capacity configuration operations are attempted on storage related to the failed ProtocolController, an Error is reported. The error reports the Object Name of the ProtocolController that has failed through the ErrorSource property. Error is a class introduced in CIM 2.9 that provides a mechanism to express error number, category, recommended actions and the like.

<u>Standard Errors</u>

It is mandatory to report error conditions through both AlertIndication and Error in those cases where Error is returned when the method call failed for reasons other than the method call itself. For example, if the device port is down then a method call can fail because of this condition. It is expected that the device will report a port error AlertIndication to listening clients as well.

Operational status and Health State (Polling)

A client that gets the top Computer system instance should see an operational status of degraded and a health state of good if the data wasn't lost. At the same time, reading the instance of Computer system for the broken controller would see an operational status of "stopped" and a health state of "non-recoverable Error".

EXPERIMENTAL

Fault Region

The RelatedElementCausingError association defines the relationship between a CIM Instance that is reporting an error status and the component that is the cause of the reported status. The Port and a Volume using the port both report error status and the RelatedElementCausingError association reports that the ProtocolController through which the Volume is exposed has failed and at least some of the volumes are no longer visible externally to the array. The array itself would be thereby degraded.

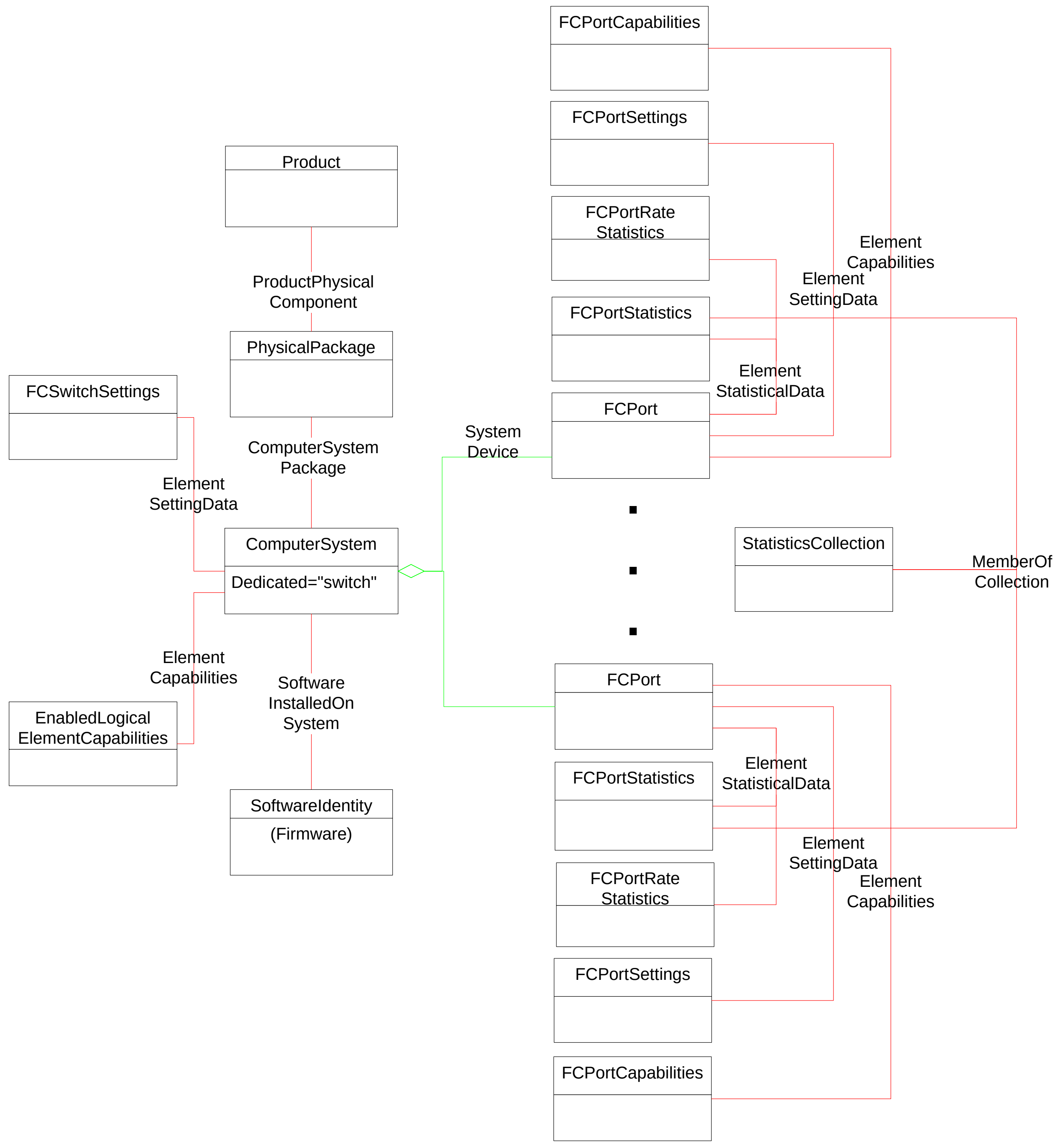
The RelatedElementCausingError association is independent of all other associations. It is only use to report error associations and comes into existence only when necessary. Once the error has been handled, the association is removed from the model.

EXPERIMENTAL

6.3.4.6.2 Switch Example

The scenario presented is related to a FC Switch that contains many ports. One of the ports is off-line.

Figure 20: Switch Example



Indication

An AlertIndication is produced by the switch notifying the HFM enabled client of the failure. The indication reports the Object Name of the FC port (FCPort) that has failed through its AlertingManagedElement property.

Standard Errors

A call to Port settings, port capabilities, or statistics cause an Error to be reported. The error reports the Object Name of the FCPort that has failed through the ErrorSource property.

It is mandatory to report error conditions through both AlertIndication and Error in those cases where Error is returned when the method call failed for reasons other than the method call itself. For example, if the device is over heat, then a method call can fail because of this condition. It is expected that the device will report an over heat AlertIndication to listening clients as well.

EXPERIMENTAL

Fault Region

The RelatedElementCausingError association defines the relationship between a CIM Instance that is reporting an error status and the component that is the cause of the reported status. The failed port would report error status and the RelatedElementCausingError association reports that the PortStatistics and PortSettings are effected. The switch itself would be thereby degraded.

The RelatedElementCausingError association is independent of all other associations. It is only use to report error associations and comes into existence only when necessary. Once the error has been handled, the association is removed from the model.

EXPERIMENTAL

6.4 Policy

6.4.1 Objectives

Policy in the context of SMI-S refers to the common expression of policy in the management of storage. The specific objectives to be addressed by policy include:

a) Provide for the exposure of element specific policies which control the behavior of management for the element. This includes:

   1) Native behavior currently unexposed through a standard interface;

   2) Behavior that can be implemented on behalf of the element by the SMI-S implementation;

b) Provide a common expression that can be extended for vendor specific behavior;

c) Provide for a mechanism that allows for embedded policy implementations;

d) Provide for the implementation of policy external to individual elements;

e) Provide for policies that work across multiple profiles and implementations of those profiles;

f) Provide a policy mechanism that scales to enterprise environments;

g) Extend existing DMTF standard policy models that are used for network and security;

h) Provide a mechanism that allows SMI-S clients to determine the level of (SMI-S) policy support.

6.4.2 Overview

Policy is the expression of management behavior such that administrators and other management software can control that behavior, tailoring it to accomplish specific goals. Policy based storage management holds the promise of reducing the cost and complexity of the mostly manual management of storage resources today. Policies provide for a level of automation while allowing control over the behavior of that automation. Policies are envisioned as being implemented by management software and device vendors and manipulated and extended by administrators in order to achieve specific results in their environment.

Any good IT organization has specific policies and procedures as well as best practices that are followed (largely through manual tasks) by the IT personnel in managing the IT environment. The best organizations have documented these policies and have a process for updating and revising them. Policy based management allows for the creation and maintenance of management policies that automate the management of IT environments to achieve the desired goals of the business. These policies can themselves be managed just as the best organizations manage their written policies and procedures.

Note: Management of policy implementations including policy services and management of policies themselves is left for a future

Note: SMI-S.

6.4.3 Policy Terms

A number of concepts have required new terms to be defined as follows:

Policy Client: An CIM Client that creates or manipulates instances of policy classes in a CIM Server.

Policy Implementation: The implementation of policy class instances in a CIM Server (i.e., providers).

Policy Based:.An SMI-S compliant implementation that supports one or more policy profiles directly.

Policy Enabled: An SMI-S profile, subprofile or package that includes properties and methods that are used in one or more policy profiles.

6.4.4 Policy Definition

The expression of Policy in the CIM Model takes the form of policy rules that aggregate conditions and actions whereby upon successful evaluation of the conditions the actions are taken. Up until recently, the specialization of these base classes was along the lines of specific extensions for domains such as Networking and Security. Rather than create domain specific extensions for storage, a more general approach wa s taken.

The new extensions to the CIM Policy Model are meant to enable policies that act on anything that is itself modeled in CIM, testing conditions on instances in the model and invoking methods to manipulate those instances.

Note: More information on the CIM Policy model and its application can be found in the DMTF Policy whitepaper. This clause does not attempt to duplicate that material.

6.4.4.1 Query Condition

The base PolicyCondition class is extended in such a way as to allow a condition to query any state that exists in implementations of the model. The new class QueryCondition allows a query string of unrestricted complexity to be used, just as any other CIM Client, to interrogate the model and create results. The presence of these results indicates that the condition evaluated to true. The absence means that the condition is false.

6.4.4.2 Method Action

The base PolicyAction class is extended so that any arbitrary method (extrinsic or intrinsic) can be invoked and appropriate parameters can be passed. This is also accomplished by a query string that, in this case, specifies how to use the results from the QueryCondition(s).

6.4.4.3 Query Condition Result

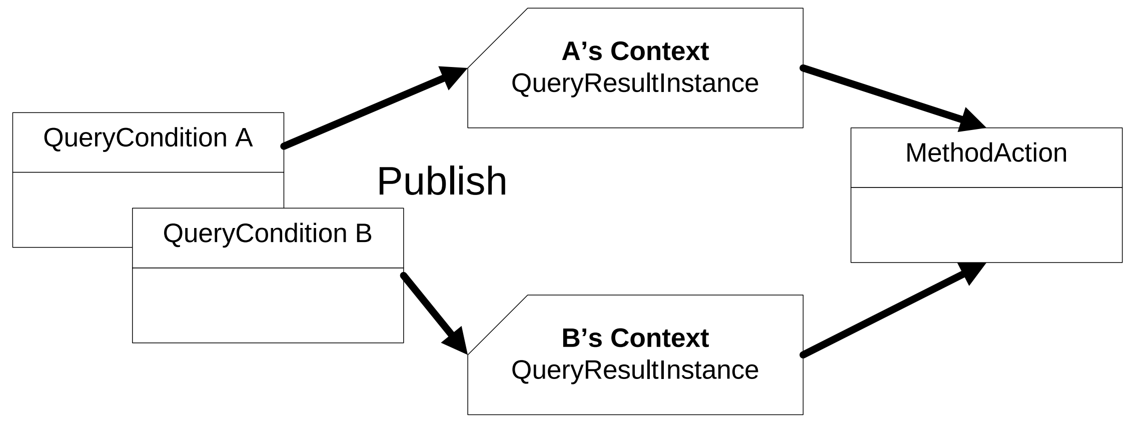
The result of a QueryCondition when it evaluates to TRUE (FALSE produces no results by definition) is one or more "rows" of embedded objects each with a predefined classname of QueryResultInstance whose properties match (both name and type) the query select criteria. This result is used possibly by other query conditions and method actions.

6.4.4.4 Method Action Result

The result of a MethodAction is an instance indication that has scope and life only within an enclosing PolicyRule. This result is used by other method Policy-Based Support

The implementation of policy can be limited to static instances of a policy model, specifically due to the cost in resources to implement full query language support. This is similar to the situation with Indication Filters in that static instances of QueryCondition and MethodActions and their associated query strings are available when a CIM Client interrogates the model for the device. Attempts to create new instances of these classes will fail because the logic embodied in the query strings is hard coded by the implementation.

Figure 21: Use of Results as Context in the Execution of a Policy Rule



There are basically three levels of support:

Full Dynamic Policy Rules – a CIM Client can create new instances of PolicyRule, QueryCondition and MethodAction. Full support for a query language is implied in this level of support

Dynamic Rules with Static Components – a CIM Client may discover existing QueryConditions and MethodActions, but may not create new ones. PolicyRule instances may be created to combine them in unlimited ways.

Static Rules and Components – the full logic of the rule instances that already exist is hard coded by the implementation. The CIM Client only has control of what the policy applies to (via, for example. PolicySetAppliesToElement).

Any level of policy implementation may use the association PolicySetAppliesToElement to apply the policy to one or more elements in the same object manager, but the query strings need to already specify how the association is used.

6.4.4.5 Capabilities

The level of policy based support is driven by a capabilities class described in the policy subprofile. Normative text for implementing this capability class is covered there.

6.4.5 Policy Recipes

The principal recipe for policy is the creation of policies in a CIM Server as follows:

```
// DESCRIPTION
// This recipe describes how to create a policy. The assumption is made that
// there is only one policy implementation present in the system.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1. The rule name is known as the #PolicyRuleName variable.
// 2. The condition name is known as the #PolicyConditionName variable.
// 3. The condition query is known as the #Query variable.

// MAIN
// Step 1. Create a Policy Rule
$PolicyRule = newInstance(“CIM_PolicyRule”)
$PolicyRule.setProperty(“PolicyRuleName”, #PolicyRuleName)
$PolicyRule.setProperty(“Enabled”, 2)// disabled
$PolicyRule.setProperty(“SequencedActions”, 1)// mandatory
$PolicyRule.setProperty(“ExecutionStrategy”, 3)// do until failure
$PolicyRule-> = CreateInstance($PolicyRule)

// Step 2. Create a Policy Condition
$QueryCondition = newInstance(“CIM_QueryCondition”)
$QueryCondition.setProperty(“PolicyConditionName”, #PolicyConditionName)
$QueryCondition.setProperty(“Query”, #Query)
$QueryCondition.setProperty(“QueryLanguage”, “CQL”)
$QueryCondition-> = CreateInstance($QueryCondition)

// Step 3. Associate Condition to the Rule
$PolicyConditionInRule = newInstance(“CIM_PolicyConditionInPolicyRule”)
$PolicyConditionInRule.setProperty(“GroupComponent”, $PolicyRule->)
$PolicyConditionInRule.setProperty(“PartComponent”, $QueryCondition->)
$PolicyConditionInRule-> = CreateInstance($PolicyConditionInRule)

// Step 4. Create the Action
$MethodAction = newInstance(“CIM_MethodAction”)
<Assign values to MethodAction attributes>
$PolicyAction-> = CreateInstance($MethodAction)

// Step 5. Associate the Action to the Rule
$PolicyActionStruct = newInstance(“CIM_PolicyActionStructure”)
$PolicyActionStruct.setProperty(“GroupComponent”, $PolicyRule->)
$PolicyActionStruct.setProperty(“PartComponent”, $PolicyAction->)
$PolicyActionStruct.setProperty(“ActionOrder”, 1)// Group 1
$PolicyActionStruct-> = CreateInstance($PolicyActionStruct)

// Step 6. Enable the Rule
$PolicyRule.Enabled = 1// Enabled
ModifyInstance($PolicyRule->,
     $PolicyRule,
     false,
     {“Enabled”})
```

EXPERIMENTAL

6.5 Standard Messages

6.5.1 Overview

Management of computer resources is, at times, fraught with exceptional conditions. SMI-S provides the means by which storage related computing resources can be controlled, configured, and, to some extent, monitored. This clause defines standard messages used in reporting the nature of these exceptional condition. Standard Messages are the expression of exceptional conditions in a managed device or application in a standard form. In other words, the indication of this condition as a standard message enables a client application that relies solely on SMI-S for instrumentation to take meaningful action in response.

There are two types of SMI-S enabled client applications supported by standard messages. The first type actively configures and controls. It requires the details why these types of operations failed to complete successfully. The second type of client application is a passive observer of state changes from the SMI-S Agent. It is solely an observer.

Failures in active management may arise for three reasons. The first type of failure is caused by invalid parameters or an invalid combination of parameters to an extrinsic or intrinsic CIM Operation. The second type of failure may also be caused by reasons other than the way in which the operation was requested of the SMI-S agent. The third type of failure may be result from an exception condition in the WBEM Infrastructure itself.

The monitoring client waits for indications of exception condition on the device or application it is monitoring.

A CIM Operations may be successful and return a response or they may be unnecessarily and return an error. The error is the combination of a standard CIM status code, like CIM_ERR_FAILED, a description, and Error instance. This clause uses the term *Error* for the Error instance returned.

A particular combination of state changes within the computer resource may arise from a single condition. The profile, subprofile, or package designers may choose to indicate the condition directly. This indication can be sent to the client, asynchronously, as a AlertIndication instance. This clause uses the term *Alert* for the AlertIndication instance. The combination of the standard message and the enclosing vehicle is called a standard event.

See 8.2.1.6, "Health Package" for further details on this mechanism.

The Errors and Alerts produced need to be interoperability interpreted by the client application that receives them. Without such interoperability, the client developer would behavior details of the computer resource in question from other sources than SMI-S. This situation is undesirable for functionality specified in SMI-S because it means that the functionality specification is incomplete.

Some types of exceptional conditions may be both the Error resulting from some CIM Operation and an Alert, like ‘system is shutting down’. The same standard message should be conveyed either an Error or an Alert such that both types of clients can interpret the indication in the same manner. Additionally, these types of exceptional conditions may be indicated from a read or write CIM Operation.

6.5.2 Required Characteristics of Standard Messages

Declaring and Producing Standard Messages

Standard Messages are defined in registries. Each registry is the collection of standard messages defined by a particular working group. In the case of SNIA, the registry is defined by particular working groups. Each working group works on a part or domain of the storage management problem. Each message as a unique id within the content of an owning organization, SNIA in this case, and working group.

Each message in the registry shall define values for the five message properties, OwningEntity, MessageID, Message, MessageArguments, and MessageFormatString. Since registries are a collection of messages and each registry is defined within the context of a owning entity, the owning entity is implied.

The message, as conveyed in an Error or Alert, and received by a client, shall contain the OwningEntity, MessageID, Message, and MessageArguments. See “Standard Events” in the 8.2.1.6, "Health Package".

When the Message is produced, the variables defined in the MessageFormatString are replaced with the values from the MessageArguments array in the order in which the variables are defined. The MessageArguments array is an array of strings. So the implementation shall coerce the value in its native CIM data type to a string before adding that value to the MessageArguments. A client may coerce that value back to its native data type using the string coercion rules for each CIM data type.

An argument present in the MessageArguments array may itself be an array. The coercion of this array argument to a string element in the MessageArgument shall result in each value of the array argument to be delimited in the resulting string by a comma. If a value within the array argument contains a whitespace, then the value of that element shall appear in the MessageArgument element contained within matching double quotes in the resulting common delimited list of array argument elements. The resulting comma delimited list of array arguments elements shall contain no whitespace characters other that those that are part of a element value.

Neither the Message nor the MessageArguments shall contain non-printable characters other than the whitespace.

The Message shall be localized in the language requested by the client. See the *CIM Operations over HTTP* specification, version 1.2, for details on internationalization with WBEM.

A Standard Message may be conveyed with an Error or an Alert. The omission of specific values for the other properties in the Error or Alert instance does not imply that this message may not be conveyed in the omitted form.

Table 11: Example Standard Message Declaration

Message Property	Value
OwningEntity	SNIA
MessageID	MP5
MessageFormatString”	Parameter <Position> of the <Method Type> method, <Method Name> , is invalid producing <Status Code> . <Additional Status>
MessageArguments	Position: The position the errant argument appears in the declaration of the method, from left to right. Method Type: intrinsic or extrinsic Method Name Status Code: CIM Status Code <status code> Additional Status: Additional circumstances describing the error (ex. Parameter out of range).

Given the following method declaration:

```
  uint32 RequestStateChange(
           [IN, Description (“…”),
            ValueMap { "2", "3", "4", "5", "6", "7..32767",
               "32768..65535" },
            Values { "Start", "Suspend", "Terminate", "Kill", "Service",
               "DMTF Reserved", "Vendor Reserved" }]
        uint16 RequestedState,
           [IN, Description ("…")]
        datetime TimeoutPeriod);
```

A client makes the following call

```
RequestStateChange(“1”, null);
```

“1” is an invalid RequestedState. Therefore, the target of the CIM Operations will produce a Error.

Table 12: Example Standard Message Values

Message Property	Value
OwningEntity	SNIA
MessageID	MP5
Message	Parameter 0 of the extrinsic method, RequestStateChange, is invalid producing CIM_ERR_INVALID_PARAMETER CIM Error. Parameter out of range.
MessageArguments	“0” "extrinsic" RequestStateChange “CIM_ERR_INVALID_PARAMETER” “Parameter out of range”

6.5.2.1 Common Messages

6.5.2.1.1 Message: Authorization Failure

Owning Entity: SNIA
Message ID: MP1
Message Format String: <Type of Operation> Access is Denied

Table 13 describes the message arguments.

Table 13: Authorization Failure Message Arguments

Message Argument	Data Type	Description	Possible Values
Type of Operation	string	Type of operation attempted.	Creation
			Modification
			Deletion
			Execution

Table 14 describes the error properties.

Table 14: Authorization Failure Error Properties

Property	Value	Description
CIMSTATUSCODE	2 (CIM_ERR_ACCESS_DENIED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(A reference to the object to whom access is requested.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.2 Message: Operation Not Supported

Owning Entity: SNIA
Message ID: MP2
Message Format String: <CIM Operation> is not supported.

Table 15 describes the message arguments.

Table 15: Operation Not Supported Message Arguments

Message Argument	Data Type	Description	Possible Values
CIM Operation	string		GetClass
			GetInstance
			DeleteClass
			DeleteInstance
			CreateClass
			CreateInstance
			ModifyClass
			ModifyInstance
			EnumerateClasses
			EnumerateInstances
			EnumerateInstanceNames
			ExecQuery
			Associators
			AssociatorNames
			References
			ReferenceNames
			GetProperty
			SetProperty
			GetQualifier
			SetQualifier
			DeleteQualifier
			EnumerateQualifier

6.5.2.1.3 Message: Property Not Found

Owning Entity: SNIA

Message ID: MP3
Message Format String: <Property Name> property was not found in the <Class name> class.

Table 16 describes the message arguments.

Table 16: Property Not Found Message Arguments

Message Argument	Data Type	Description	Possible Values
Property Name	string	The property name is specified as it was passed by the client.	
Class name	string	The property name is specified as it was passed by the client.	

6.5.2.1.4 Message: Invalid Query

Owning Entity: SNIA
Message ID: MP4
Message Format String: Query language is not supported. The query language supported are <Supported Query Languages>

Table 17 describes the message arguments.

Table 17: Invalid Query Message Arguments

Message Argument	Data Type	Description	Possible Values
Supported Query Languages	string		

6.5.2.1.5 Message: Parameter Error

Owning Entity: SNIA
Message ID: MP5
Message Format String: Parameter <Position> of the <Method Type> method, <Method Name> , is invalid producing <Status Code> . <Additional Status>

Table 18 describes the message arguments.

Table 18: Parameter Error Message Arguments

Message Argument	Data Type	Description	Possible Values
Position	uint16	The position the errant argument appears in the declaration of the method, from left to right.	
Method Type	string		extrinsic
			intrinsic
Method Name	string		
Status Code	string		no
			CIM Status Code: Add status code number after the above
Additional Status	string		parameter value out of range
			invalid combination
			null parameter is not permitted
			non-null value is not permitted

Table 18: Parameter Error Message Arguments

Message Argument	Data Type	Description	Possible Values
			empty string is not permitted
			empty array is not permitted

Table 19 describes the error properties.

Table 19: Parameter Error Properties

Property	Value	Description
CIMSTATUSCODE	4 (CIM_ERR_INVALID_PARAMETER)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.6 Message: Query Syntax Error

Owning Entity: SNIA
Message ID: MP6
Message Format String: Syntactical error on query: <Errant Query Components> <Syntax Errors>

Table 20 describes the message arguments.

Table 20: Query Syntax Error Message Arguments

Message Argument	Data Type	Description	Possible Values
Errant Query Components	string	The parts of the query that are in error with a carrot '^' in front of text that is in error	
Syntax Errors	string	The syntax errors for each of the query components in the previous argument. The two arrays are to match element to element.	

Table 21 describes the error properties.

Table 21: Query Syntax Error Properties

Property	Value	Description
CIMSTATUSCODE	4 (CIM_ERR_INVALID_QUERY)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.7 Message: Query Too Expensive

Owning Entity: SNIA
Message ID: MP7
Message Format String: Query is too expensive because the <Rejection Reason>

Table 22 describes the message arguments.

Table 22: Query Too Expensive Message Arguments

Message Argument	Data Type	Description	Possible Values
Rejection Reason	string		result set will be too big
			query will take too many computing resources to process

Table 23 describes the error properties.

Table 23: Query Too Expensive Error Properties

Property	Value	Description
CIMSTATUSCODE	4 (CIM_ERR_INVALID_QUERY)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.8 Message: Class or Property Invalid in Query

Owning Entity: SNIA
Message ID: MP8
Message Format String: Invalid <Invalid Query Component>

Table 24 describes the message arguments.

Table 24: Class or Property Invalid in Query Message Arguments

Message Argument	Data Type	Description	Possible Values
Invalid Query Component	string	This argument shall contain the 'class name' or 'class name'.'property name'	

Table 25 describes the error properties.

Table 25: Class or Property Invalid in Query Error Properties

Property	Value	Description
CIMSTATUSCODE	4 (CIM_ERR_INVALID_QUERY)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.9 Message: Invalid Join in Query

Owning Entity: SNIA
Message ID: MP9
Message Format String: Invalid join clause: <Invalid Join Clause>

Table 26 describes the message arguments.

Table 26: Invalid Join in Query Message Arguments

Message Argument	Data Type	Description	Possible Values
Invalid Join Clause	string	This argument shall contain the entire join clause that is in error.	

Table 27 describes the error properties.

Table 27: Invalid Join in Query Error Properties

Property	Value	Description
CIMSTATUSCODE	4 (CIM_ERR_INVALID_QUERY)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.10 Message: Unexpected Hardware Fault

Owning Entity: SNIA
Message ID: MP10
Message Format String: Call technical support and report the following error number has occurred, <Hardware Error>

Table 28 describes the message arguments.

Table 28: Unexpected Hardware Fault Message Arguments

Message Argument	Data Type	Description	Possible Values
Hardware Error	sint32	Vendor specific hardware error. Use this error, only when all other standard messages can not cover this condition.	

Table 29 describes the error properties.

Table 29: Unexpected Hardware Fault Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(It is discouraged from specifying any reference here.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.1.11 Message: Too busy to respond

Owning Entity: SNIA
Message ID: MP11

Message Format String: WBEM Server is <Adverse Condition> to respond.

Table 30 describes the message arguments.

Table 30: Too busy to respond Message Arguments

Message Argument	Data Type	Description	Possible Values
Adverse Condition	string		too busy
			initializing

6.5.2.1.12 Message: Shutdown Started

Owning Entity: SNIA
Message ID: MP12
Message Format String: The computer system is shutting down in <seconds to shutdown> seconds.

Table 31 describes the message arguments.

Table 31: Shutdown Started Message Arguments

Message Argument	Data Type	Description	Possible Values
seconds to shutdown	uint32	The number of seconds before the system is shutdown.	

Table 32 describes the alerts that are associated with this message.

Table 32: Shutdown Started Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		The object name must reference the top-most computer system that is shutting down. If the computer system is cluster, then the cluster computer system must be referenced.
ALERT_TYPE	Y	5	Device Alert
PERCEIVED_SEVERITY	Y	4	High

6.5.2.1.13 Message: Component overheat

Owning Entity: SNIA
Message ID: MP13
Message Format String: A component has overheated. <Component Type>

Table 33 describes the message arguments.

Table 33: Component Overheat Message Arguments

Message Argument	Data Type	Description	Possible Values
Component Type	string		The entire device is affected. Device wide failure has already or can be expected shortly.
			Only a single component is affected. Corrective action may be taken.

Table 34 describes the error properties.

Table 34: Component Overheat Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	6 (Environment Error)	Existence is required
ERROR_SOURCE	(The object name must reference the physical element most affected by the over temperature message.)	Existence is required
PERCEIVED_SEVERITY	4 (High)	Existence is required

Table 35 describes the alerts that are associated with this message.

Table 35: Component overheat Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		The object name must reference the physical element most affected by the over temperature message.
ALERT_TYPE	Y	6	Environmental Alert
PERCEIVED_SEVERITY	Y	4	High

6.5.2.1.14 Message: WBEM Management Interface is not available

Owning Entity: SNIA
Message ID: MP14
Message Format String: The management interface for the device is not available.

6.5.2.1.15 Message: Device Failover

Owning Entity: SNIA
Message ID: MP15
Message Format String: Management interface is active on different device at the following URI, <URI>

Table 36 describes the message arguments.

Table 36: Device Failover Message Arguments

Message Argument	Data Type	Description	Possible Values
URI	string		

6.5.2.1.16 Message: Functionality is not licensed

Owning Entity: SNIA
Message ID: MP16
Message Format String: Functionality requested is not licensed. The following license is required, <Required License Name>

Table 37 describes the message arguments.

Table 37: Functionality is Not Licensed Message Arguments

Message Argument	Data Type	Description	Possible Values
Required License Name	string		

Table 38 describes the error properties.

Table 38: Functionality is not licensed Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Reference to top most Computer System.)	Existence is required
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.1.17 Message: Invalid Property Combination during instance creation or modification

Owning Entity: SNIA
Message ID: MP17
Message Format String: The instance contains an invalid combination of properties. The <Errant Property Name> property may not have the value, <Errant Property Value> , when the <Existing Property Name> property has value, <Existing Property Value>

Table 39 describes the message arguments.

Table 39: Invalid Property Combination During Instance Creation or Modification Message Arguments

Message Argument	Data Type	Description	Possible Values
Errant Property Name	string	The name of the property is primary reason for the rejection of this instance.	
Errant Property Value	string	The invalid property value, coerced as a string.	
Existing Property Name	string	The property whose value has to be set in some way before or regardless of the "Errant Property Name" property. For example, property A of value X may be compatible with property B with value Y. But, property B may have had value Y prior to property A having a value or value X. Or, property B may be a key and must logically have a value before any other property set operation is considered.	
Existing Property Value	string		

Table 40 describes the error properties.

Table 40: Invalid Property Combination during instance creation or modification Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Nothing to reference.)	Existence is discouraged
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.1.18 Message: Property Not Found

Owning Entity: SNIA
Message ID: MP18
Message Format String: <Errant Property Name> property was not found in class <Class Name used in Operation>

Table 41describes the message arguments.

Table 41: Property Not Found Message Arguments

Message Argument	Data Type	Description
Errant Property Name	string	The name of the property provided in a instance related CIM Operation that simply does not exist in the class as indicated by the class name.
Class Name used in Operation	string	The class name used in the CIM Operation as stated directly as a method parameters or as part of a CIM Object Name (CIM Object Path).

Table 42 describes the error properties.

Table 42: Property Not Found Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Reference the class in question.)	Existence is required
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.1.19 Message: Proxy Can Not Connect

Owning Entity: SNIA
Message ID: MP19
Message Format String: Proxy CIM provider can not connect. <Reason for Connection Failure>

Table 43 describes the message arguments.

Table 43: Proxy Can Not Connect Message Arguments

Message Argument	Data Type	Description	Possible Values
Reason for Connection Failure	string	The reason for the connection failure.	Authentication Failure
			Authorization Failure
			Communications Failure

Table 44 describes the error properties.

Table 44: Proxy Can Not Connect Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Nothing to reference.)	Existence is discouraged
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.1.20 Message: Not Enough Memory

Owning Entity: SNIA
Message ID: MP20
Message Format String: <Method Type> method <Method Name> can not be completed because of lack of memory.

Table 45 describes the message arguments.

Table 45: Not Enough Memory Message Arguments

Message Argument	Data Type	Description	Possible Values
Method Type	string		intrinsic
			extrinsic
Method Name	string	The method name. If the method is an intrinsic method, provide the CIM Operation Name, e.g., EnumerateInstances. If the method is an extrinsic method, i.e., InvokeMethod, then provide the method name in the class that was invoked.	

Table 46 describes the error properties.

Table 46: Not Enough Memory Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Nothing to reference.)	Existence is discouraged
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.1.21 Message: Object Already Exists

Owning Entity: SNIA
Message ID: MP21
Message Format String: Object already exists.

Table 47 describes the error properties.

Table 47: Object Already Exists Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Reference to the already existing zone element. ()	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2 Storage Messages

6.5.2.2.1 Message: Device Not ready

Owning Entity: SNIA
Message ID: DRM1
Message Format String: Device <Device ID> not ready because of <StateOrStatus> state or status.

Table 48 describes the message arguments.

Table 48: Device Not ready Message Arguments

Message Argument	Data Type	Description	Possible Values
Device ID	string	LogicalDevice.DeviceID, PhysicalElement.Tag, or ComputerSystem.Name	
StateOrStatus	string	Relevant State or Status that explains the reason for the production of this message.	

Table 49 describes the error properties.

Table 49: Device Not ready Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(Object Name for the top-level object for the device, which is typically the computer system instance)	Existence is required
PERCEIVED_SEVERITY	4 (High)	Existence is required

6.5.2.2.2 Message: Internal Bus Error

Owning Entity: SNIA
Message ID: DRM2
Message Format String: Internal Bus Error

Table 50 describes the error properties.

Table 50: Internal Bus Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(Object Name for the top-level object for the device, which is typically the computer system instance)	Existence is required
PERCEIVED_SEVERITY	4 (High)	Existence is required

6.5.2.2.3 Message: DMA Overflow

Owning Entity: SNIA
Message ID: DRM3
Message Format String: DMA Overflow

Table 51 describes the error properties.

Table 51: DMA Overflow Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	4 (High)	Existence is required

6.5.2.2.4 Message: Firmware Logic Error

Owning Entity: SNIA
Message ID: DRM4
Message Format String: Firmware Logic Error

Table 52 describes the error properties.

Table 52: Firmware Logic Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	4 (High)	Existence is required

6.5.2.2.5 Message: Front End Port Error

Owning Entity: SNIA
Message ID: DRM5
Message Format String: Front End Port Error on Device identified by <Device ID>

Table 53 describes the message arguments.

Table 53: Front End Port Error Message Arguments

Message Argument	Data Type	Description	Possible Values
Device ID	string	LogicalDevice.DeviceID	

Table 54describes the alerts that are associated with this message.

Table 54: Front End Port Error Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		Object Name for the top-level object for the device, which is typically the computer system instance
ALERT_TYPE	Y	2	Communications Alert
PERCEIVED_SEVERITY	Y	4	High

6.5.2.2.6 Message: Back End Port Error

Owning Entity: SNIA
Message ID: DRM6
Message Format String: Back End Port Error on Device identified by <Device ID>

Table 55 describes the message arguments.

Table 55: Back End Port Error Message Arguments

Message Argument	Data Type	Description	Possible Values
Device ID	string	LogicalDevice.DeviceID	

Table 56 describes the alerts that are associated with this message.

Table 56: Back End Port Error Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		Object Name for the top-level object for the device, which is typically the computer system instance
ALERT_TYPE	Y	2	Communications Alert
PERCEIVED_SEVERITY	Y	4	High

6.5.2.2.7 Message: Remote Mirror Error

Owning Entity: SNIA
Message ID: DRM7
Message Format String: Error detected associated with remote volume, <Remote Volume Name>

Table 57 describes the message arguments.

Table 57: Remote Mirror Error Message Arguments

Message Argument	Data Type	Description	Possible Values
Remote Volume Name	string	StorageVolume.Name	

Table 58 describes the error properties.

Table 58: Remote Mirror Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(Object Name for the top-level object for the remote block server, which is typically the computer system instance. The implementation will have to implement the Cascading Subprofile.)	Existence is optional
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

Table 59 describes the alerts that are associated with this message.

Table 59: Remote Mirror Error Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	N		Object Name for the top-level object for the remote block server, which is typically the computer system instance. The implementation will have to implement the Cascading Subprofile.
ALERT_TYPE	Y		
PERCEIVED_SEVERITY	Y	3	Medium

6.5.2.2.8 Message: Cache Memory Error

Owning Entity: SNIA
Message ID: DRM8
Message Format String: Cache Memory Error

Table 60 describes the error properties.

Table 60: Cache Memory Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(Object Name for the top-level object for the device, which is typically the computer system instance)	Existence is required
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.2.9 Message: Unable to Access Remote Device

Owning Entity: SNIA
Message ID: DRM9
Message Format String: Unable to Access Remote Device

Table 61 describes the error properties.

Table 61: Unable to Access Remote Device Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	(Object Name for the top-level object for the remote block server, which is typically the computer system instance. The implementation will have to implement the Cascading Subprofile.)	Existence is optional
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.2.10 Message: Error Reading Data

Owning Entity: SNIA
Message ID: DRM10
Message Format String: Error Reading Data

Table 62 describes the alerts that are associated with this message.

Table 62: Error Reading Data Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		Object Name for the top-level object for the device, which is typically the computer system instance
ALERT_TYPE	Y	2	Communications Alert
PERCEIVED_SEVERITY	Y	3	Medium

6.5.2.2.11 Message: Error Writing Data

Owning Entity: SNIA
Message ID: DRM11
Message Format String: Error Writing Data

Table 63 describes the alerts that are associated with this message.

Table 63: Error Writing Data Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		Object Name for the top-level object for the device, which is typically the computer system instance
ALERT_TYPE	Y	2	Communications Alert
PERCEIVED_SEVERITY	Y	3	Medium

6.5.2.2.12 Message: Error Validating Write (CRC)

Owning Entity: SNIA
Message ID: DRM12
Message Format String: Error Validating Write

Table 64 describes the alerts that are associated with this message.

Table 64: Error Validating Write (CRC) Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		Object Name for the top-level object for the device, which is typically the computer system instance
ALERT_TYPE	Y	2	Communications Alert
PERCEIVED_SEVERITY	Y	3	Medium

6.5.2.2.13 Message: Copy Operation Failed

Owning Entity: SNIA
Message ID: DRM13
Message Format String: Copy Operation Failed

Table 65 describes the error properties.

Table 65: Copy Operation Failed Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	()	Existence is discouraged
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.2.14 Message: RAID Operation Failed

Owning Entity: SNIA
Message ID: DRM14
Message Format String: RAID Operation Failed

Table 66 describes the error properties.

Table 66: RAID Operation Failed Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	5 (Hardware Error)	Existence is required
ERROR_SOURCE	()	Existence is discouraged
PERCEIVED_SEVERITY	3 (Medium)	Existence is required

6.5.2.2.15 Message: Invalid RAID Type

Owning Entity: SNIA
Message ID: DRM15
Message Format String: Invalid RAID Type

Table 67 describes the error properties.

Table 67: Invalid RAID Type Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	10 (Unsupported Operation Error)	Existence is required
ERROR_SOURCE	()	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.16 Message: Invalid Storage Element Type

Owning Entity: SNIA
Message ID: DRM16
Message Format String: Invalid Device Type

Table 68 describes the error properties.

Table 68: Invalid Storage Element Type Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	10 (Unsupported Operation Error)	Existence is required
ERROR_SOURCE	()	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.17 Message: Configuration Change Failed

Owning Entity: SNIA
Message ID: DRM17
Message Format String: Configuration Change Failed

Table 69 describes the error properties.

Table 69: Configuration Change Failed Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.18 Message: Buffer Overrun

Owning Entity: SNIA
Message ID: DRM18
Message Format String: Buffer Overrun

Table 70 describes the error properties.

Table 70: Buffer Overrun Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.19 Message: Stolen Capacity

Owning Entity: SNIA
Message ID: DRM19
Message Format String: The capacity requested, <Requested Capacity> , that was requested is no longer available.

Table 71 describes the message arguments.

Table 71: Stolen Capacity Message Arguments

Message Argument	Data Type	Description
Requested Capacity	sint64	Capacity requested in bytes expressed in powers of 10.

Table 72 describes the error properties.

Table 72: Stolen Capacity Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(The pool, volume, or logical disk being modified, or, in the case of element creation the parent pool from which capacity is being drawn.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.20 Message: Invalid Extent passed

Owning Entity: SNIA
Message ID: DRM20
Message Format String: One or more of the extents passed can not be used to create or modify storage elements. <Invalid Extents Array>

Table 73 describes the message arguments.

Table 73: Invalid Extent passed Message Arguments

Message Argument	Data Type	Description	Possible Values
Invalid Extents Array	reference	Array of references to the all Extents that can not be used in the specified manner (ex. CreateOrModifyStroragePool or CreateOrModifyElementFromElements).	

Table 74 describes the error properties.

Table 74: Invalid Extent passed Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(A reference to the storage configuration service instance on which the method was called that caused this error.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.21 Message: Invalid Deletion Attempted

Owning Entity: SNIA
Message ID: DRM21
Message Format String: Existing pool or storage element (StorageVolume or LogicalDisk) may not be deleted because there are existing Storage Extents which relay on it.

Table 75 describes the error properties.

Table 75: Invalid Deletion Attempted Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(A reference to one of the dependent StorageExtents.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.22 Message: Job Failed to Start

Owning Entity: SNIA
Message ID: DRM22
Message Format String: Job failed to start because resources required for method execution are no longer available.

Table 76 describes the error properties.

Table 76: Job Failed to Start Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	8 (Oversubscription Error)	Existence is required
ERROR_SOURCE	(Reference to Job instance which failed to start for this reason if a Job instance was created because of the time required to make this resource assessment. If a Job instance was not created, because the assessment was fast enough, then this property must be NULL.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.23 Message: Job was Halted

Owning Entity: SNIA
Message ID: DRM23
Message Format String: Job was <Reason for Job halt>

Table 77 describes the message arguments.

Table 77: Job was Halted Message Arguments

Message Argument	Data Type	Description	Possible Values
Reason for Job halt	string	A Job may be stopped by a client using the RequestedStateChange method. If the job stopped executing for other reasons, then use a different message.	killed
			terminated

6.5.2.2.24 Message: Invalid State Transition

Owning Entity: SNIA
Message ID: DRM24
Message Format String: An invalid state transition, <Invalid Sync State> , was requested given current state, <Current Sync State>

Table 78 describes the message arguments.

Table 78: Invalid State Transition Message Arguments

Message Argument	Data Type	Description
Invalid Sync State	string	The textual equivalent (Value) for StorageSynchronized.SyncState value requested.
Current Sync State	string	The textual equivalent (Value) for the current StorageSynchronized.SyncState value

Table 79 describes the error properties.

Table 79: Invalid State Transition Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Reference to the StorageSyn-chronized instance in question.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.25 Message: Invalid SAP for Method

Owning Entity: SNIA
Message ID: DRM25
Message Format String: Invalid type of copy services host. The host must be a <Host Type>

Table 80 describes the message arguments.

Table 80: Invalid SAP for Method Message Arguments

Message Argument	Data Type	Description	Possible Values
Host Type	string	The type of copy services on which the method was invoked.	source
			target

Table 81 describes the error properties.

Table 81: Invalid SAP for Method Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Reference to the Computer Sys-tem host which is of the wrong type.)	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.26 Message: Resource Not Available

Owning Entity: SNIA
Message ID: DRM26
Message Format String: <Resource Needed>

Table 82 describes the message arguments.

Table 82: Resource Not Available Message Arguments

Message Argument	Data Type	Description	Possible Values
Resource Needed	string		No replication log available.
			Special replica pool required.

Table 83 describes the error properties.

Table 83: Resource Not Available Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Nothing to reference.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.27 Message: Resource Limit Exceeded

Owning Entity: SNIA
Message ID: DRM27
Message Format String: <Reason>

Table 84 describes the message arguments.

Table 84: Resource Limit Exceeded Message Arguments

Message Argument	Data Type	Description	Possible Values
Reason	string	The reasons for the lack of resources for copy services operation.	Insufficient pool space.
			Maximum replication depth exceeded.
			Maximum replicas exceeded for source element.

Table 85 describes the error properties.

Table 85: Resource Limit Exceeded Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	(Nothing to reference.)	Existence is discouraged
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.2.28

6.5.2.3 Fabric Messages

6.5.2.3.1 Message: Zone Database Changed

Owning Entity: SNIA
Message ID: FC1
Message Format String: Zone database changed for <Fabric Identity Type> named <WWN>

Table 86 describes the message arguments.

Table 86: Zone Database Changed Message Arguments

Message Argument	Data Type	Description	Possible Values
Fabric Identity Type	string	Defines the type of fabric entity names by the following WWN.	fabric
			switch
WWN	string	World Wide name identifier. The required form of the WWN is defined by this regular expression, "^[0123456789ABCDEF]{16}$"	

Table 87 describes the alerts that are associated with this message.

Table 87: Zone Database Changed Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		A reference to the switch or fabric which is named by the WWN.
ALERT_TYPE	Y	6	Environmental Alert
PERCEIVED_SEVERITY	Y	1	Informational

6.5.2.3.2 Message: ZoneSet Activated

Owning Entity: SNIA
Message ID: FC2
Message Format String: ZoneSet <ZoneSet Name> was activated for fabric <WWN>

Table 88 describes the message arguments.

Table 88: ZoneSet Activated Message Arguments

Message Argument	Data Type	Description	Possible Values
ZoneSet Name	string	CIM_ZoneSet.ElementName attribute	
WWN	string	World Wide name identifier. The required form of the WWN is defined by this regular expression, "^[0123456789ABCDEF]{16}$"	

Table 89 describes the alerts that are associated with this message.

Table 89: ZoneSet Activated Alert Information

Name	Req	Value	Description
ALERTING_MANAGED_ELEMENT	Y		A reference to the switch of fabric which is named by the WWN.
ALERT_TYPE	Y	6	Environmental Error
PERCEIVED_SEVERITY	Y	4	High

6.5.2.3.3 Message: Session Locked

Owning Entity: SNIA
Message ID: FC3
Message Format String: Operation blocked by session lock.

Table 90 describes the error properties.

Table 90: Session Locked Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

6.5.2.3.4 Message: Session Aborted

Owning Entity: SNIA
Message ID: FC4
Message Format String: Operation by another client failed causing the session to be aborted. This error may be caused by client aborting, switch aborting the client, or timeout of session lock.

Table 91 describes the error properties.

Table 91: Session Aborted Error Properties

Property	Value	Description
CIMSTATUSCODE	1 (CIM_ERR_FAILED)	Existence is required
ERROR_TYPE	4 (Software Error)	Existence is required
ERROR_SOURCE	Object Name for the top-level object for the device, which is typically the computer system instance ()	Existence is required
PERCEIVED_SEVERITY	2 (Low)	Existence is required

EXPERIMENTAL

6.6 Recipe Overview

6.6.1 Recipe Definition

Recipe: A set of instructions for making something from mixing various ingredients in a particular sequence. The set of ingredients used by a particular recipe is scoped by the particular profile, subprofile or some other well-defined context in which that recipe is defined.

A recipe shall specify an interoperable means for accomplishing a particular task across all conformant implementations. However, a recipe does not necessarily specify the only set of instructions for accomplishing that task. Nor are all tasks that may be accomplished necessarily specified by the set of recipes defined for a particular profile or subprofile.

In order to compress the document, some recipes are implied or assumed. This would include, for instance, that the set of available, interoperable properties are those explicitly defined by a particular profile or subprofile. In general, any CIM intrinsic read methods on profile or subprofile models are implied. However, CIM intrinsic write methods (Create/Delete/Modify) should not be assumed unless explicitly listed in the profile or subprofile definition with a well defined semantic.

For a profile or subprofile, the set of all defined and implied recipes defines the range of behavior across for which interoperability is mandatory for all conformant implementations. Unless specifically defined in a recipe, other sequences of actions (even simple Create/Delete instance requests) are not guaranteed to have the same results across multiple implementations.

Each recipe defines an interoperable series of interactions (between a SMI-S Client and a SMI-S Server) required to manage storage devices or applications. Another goal is to list the operations required for the CIM Client realize functionality. It is not a goal to comprehensively express the programming logic required to implement the recipe in any particular language. In fact, recipes are limited to the expression of CIM or SLP operations, and may simply reference or describe any of the implementation that may be required beyond that.

6.6.2 Recipe Pseudo Code Conventions

6.6.2.1 Overview

A recipe's instructions are written using the pseudo code language defined in this section.

All recipes are prefixed with a summary narrative of the functionality being implemented. This summary may be included explicitly as part of the recipe or reference to the appropriate narrative that can be found elsewhere in the specification.

Note: The use of optional features (profiles or subprofiles) in recipes shall be clearly identified.

CIM Operations and their parameters are taken directly from the *CIM Operations Over HTTP* specification. It is assumed that these methods are being called on the CIM Client API. Arrays grow in size automatically.

6.6.2.2 General Syntax

<condition>	logical statement that evaluates to true (Boolean)
!<condition>	tests for false (Boolean)
<action>	unspecified list of programming logic that is not important to the understanding of the reader for a particular recipe.
<EXIT: *success message*>	Exits the recipe with a success status code. The condition that resulted in the call to exit the recipe was allowable. The implementation subjected to the recipe behaves in accordance to this specification.

<ERROR! *error condition*>	Exits the recipe with a failure status code. The condition that resulted in the call to the exit the recipe was not allowable. The implementation subjected to the recipe does not behave in accordance with this specification.
@{recipe}	logic flow is contained within the specification of the recipe elsewhere in the specification
<variable>	some variable

6.6.2.3 CIM related variable and methods

6.6.2.3.1 CIM Instances and Object Names

$name	represents a single instance (CIMInstance) with a given variable name
$name.property	represents a property in a single instance (CIMInstance)
$name.getObjectPath()	method returns a object name, REF, to the CIM Instance
$name.getNameSpace()	method returns the namespace name for the CIM Instance or Object Name
{value1, value2 ...}	an anonymous array, comprised of selected values of a given type; an anonymous array is an array that is not referable by a variable

Example:

{"Joe", "Fred", "Bob", "Celma"}

$name[]	represents an array of instances (CIMInstances) with a given variable name; array are initialized by constructing an anonymous array.

Example:

Names = {"Joe", "Fred", "Bob", "Celma"}

$name->	represents an object path name (CIMObjectPath)
$name->[]	represents an array of object names of a given name
$name->property	represents a property of object $name
$name[].size()	returns the number of CIM instances in the array
$name->[].length	returns the number of CIM object names in the array
#name[].length	returns the number of variable elements in the array
%name[].length	returns the number of method arguments elements in the array

6.6.2.3.2 Extrinsic method arguments

%name	represents a CIM Argument that can contain any CIM or other variable.
%name[]	represents an array of CIM Arguments

6.6.2.3.3 Other Variables

#name	neither CIM Instance nor Object Name variable. The type may be a string, number or some other special type. Types are defined in the CIM Specification 2.2.
#name[]	a non-CIM variable array
"literal”	some string literal

6.6.2.4 Data Structure

Variables can be collected by an array. The array can be indexed by other variable (see 6.6.2.3.3).

Arguments are always indexed by strings. In other words, the arguments are retrieved from the array by name.

6.6.2.5 Operations

=	assigns right value to left value
==	test for equivalency
!=	test for not equivalency
<	true if the left argument is numerically less than the right argument.
>	true if the left argument is numerically greater than the right argument.
<=	true if the left argument is numerically less than or equal to the right argument.
>=	true if the left argument is numerically greater than or equal to the right argument.
&&	condition A AND condition B
||	condition A OR condition B
+, -, *, /	addition, subtraction, multiplication and division, respectively
++, **--**	increment and decrement a variable, respectively; placement of the operator relative to the variable determines whether the operation is completed before or after evaluation

Example:

```
#i = 1
#names[] = {"A", "B, "C"}
"B" == #names[++#i] is true
2 == #i is true
```

Example:

```
#i = 2
#names[] = {"A", "B, "C"}
"B" == #names[#i++]  is true
3 == #i is true
```

//	comments
nameof	returns an Object Name given a CIM Instance. This unitary operator does nothing in other usages.
ISA	tests for the name of the CIM Instance or object name

Example: if ($SomeName-> ISA CIM_StorageVolume) {

```
<The Object Name is a reference to a CIM_StorageVolume >
}
```

6.6.2.6 Control Operations

The pseudocode used in this specification relies on control operators common to most high-level languages. For example:

- **for**

Example:

```
for #x in <variable array> {
<actions>
}
```

- **if**

Example:

```
if (<condition>) {
<actions>
} ;
if (<condition>) {
<actions>
} else {
<alternate actions>
}
```

- **do/while**

Example:

```
do {
<actions>
} while (<condition>)
```

- continue
  Within a **for** loop: initialize loop variable to next available value and restart loop body. Terminate loop if no more loop variable values available. Within a **do**/**while** loop: transfer control immediately to **while** test.

Example:

```
for #i in <array> {
     if (<some condition>)
        continue;    // process next loop variable
    <alternative>
}
```

- **break**: interrupts the sequence of statement execution within a loop block and exits the loop block altogether. The looping condition is not re-evaluated Statement execution starts at the next statement outside of the loop block.

- **exit**
  Terminate recipe instantly, including termination of any callers.

Example:

```
if (<unexpected condition>)
   exit
```

6.6.2.7 Functions

6.6.2.7.1 Function Declaration

A function definition is of the form *sub functionName(),* followed by the body of the function enclosed in braces. If parameters are to be passed to a function, then are expressed as a comma-separated list of arguments within the parentheses following the function name. Each argument is comprised of a data type and an accompanying argument name.

Functions are declared at the beginning of a recipe.

```
sub functionName(integer nArg1, Class &cArg2) {
<actions>
}
```

6.6.2.7.2 Function Invocation

A function invocation is of the form *&functionName()*. If parameters are to be passed to a function, then are expressed as a comma-separated list within the parentheses following the function name.

```
&functionName(5, pClass)
```

6.6.2.8 Exception Handling

All operations may produce exceptions or errors. The following construct is used to test for particular errors. Once a particular error is caught, then special exception handling logic is processed. Only CIM Errors can be caught.

```
try {
     <actions>
}
     catch (CIM Exception $Exception) {
 <recovery actions>
}
 The error received may also be thrown
 throw $Exception
```

The error response returned from the SMI-S implementation is treated as a exception, a "CIM Exception". The catch condition is expressed in terms of the CIM status code returned (e.g., CIM_ERR_NOT_FOUND) as defined in the CIM Operations specification.

The $Exception variable contains a Error instance. The $Exception CIM Instance may be examined like any other CIM Instance. In this language, the $Exception is never null even if the SMI-S implementation does provide one. In this case, the $Exception CIM Instance is empty with the exception of the CIMStatusCode and CIMStatusCodeDescription properties. This properties are populated with the Status and Description returned in the error response from the SMI-S implementation.

6.6.2.9 Built-in Functions

a) boolean = compare(<variable>, <variable>)

   1) Used to determine if two variables of the same type are equivalent

   2) The variables shall not be CIM instances or object names nor other complex data types or structures

   3) The variables shall be of the same type

b) $instance = newInstance("CIM Classname")

1) Creates a CIM instance, which does not exist in the CIMOM (yet), that can be later filled in with properties and passed to CreateInstance. The namespace is assumed to be the same that the CIM client connected to.

c) $instance - newInstance("CIM Namespace", "CIM Classname")

   1) Variable of the above method that has the namespace name as an argument

d) boolean = contains(<test value>, <variable array>)

   1) Used to test if the variable array contains a value equivalent to the test variable

   2) The array shall be of variables of the same types as the test variable.

   3) If the equivalency is found with at least one value then the function returns true, else false is returned.

   4) If the array is not a simple, or non-CIM, data type, then the test value shall be a CIM property, $SomeInstance.SomeProperty or $SomeObjectname->SomeProperty

e) %Argument = newArgument("Argument Name", <variable>)

   1) Creates a CIM Argument of a given name containing a value, CIM or non-CIM

f) $objectPath-> = newObjectPath("Class name", "NameSpace name")

   1) Returns a new ObjectPath, built from the supplied arguments;

   2) Required to perform the `EnumerateInstances and EnumerateInstanceNames operations`

g) #stringArray[] = #stringVariable.split(#stringParam or "string literal")

   1) Returns an array of strings, built by splitting the string variable around matches of the supplied string parameter

   2) Divides the string into substrings, using the string parameter as a delimiter, returning the substrings in an array in the order in which they occurred in the string variable. If there are no occurrences of the string parameter, then the array returned contains only one string element equal to the original string variable.

h) #intVariable = Integer(#stringVariable)

   1) Returns the integer that the supplied string represents. If the supplied string does not represent an integer, then an error is thrown.

   2) The function will parse and return signed or unsigned integers up to 64-bits in size, and will accept the hyphen '-' character in the 8-bit ASCII-range of UTF-8 as the first character in the string to indicate a negative number.

i) #datetimeVariable = Datetime(#stringVariable)

   1) Returns a variable of Datetime type, as defined by section 2.2.1 the CIM Infrastructure Specification v1.3, that the supplied string represents. If the supplied string does not represent a DateTime object, then an error is thrown.

   2) This function will accept strings of the format described in the CIM Infrastructure Specification, including both timestamps and intervals, zero-padded to 25-characters, and will recognize Datetime strings containing asterisk ("*") characters for fields that are not significant.

6.6.2.10 Extrinsic method calls

```
<variable> = InvokeMethod ($someobjectname->, "Method Name",
 %InArguments[], %OutArguments[])
```

Clause 7: 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.

Table 92: Standards Dependencies for SMI-S

Standard	Version	Organization
CIM Operations over HTTP (DSP0200)	1.2.0	DMTF
CIM Schema	2.11.0	DMTF
CIM Infrastructure Specification (DSP004)	2.3.0	DMTF
CIM Query Language Specification, DSP0202	1.0.0	DMTF
Specification for the Representation of CIM over XML (DSP0201)	2.2	DMTF
WBEM URI Mapping Specification (DSP0207)	1.0.01 (preliminary)	DMTF
UML	1.3	OMG
Securing Block Storage Protocols over IP (RFC 3723)		IETF
Service Location Protocol (SLP), Version 2 (RFC2608)	2	IETF
Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC3289)		IETF
Service Templates and Service: Schemes (RFC2609)		IETF
DHCP Options for Service Location Protocol (RFC2610)		IETF
An API for Service Location (RFC2614)		IETF
Hypertext Transfer Protocol -- HTTP (RFC1945)	1.0	IETF
Hypertext Transfer Protocol -- HTTP /1.1 (RFC2616, RFC2068*)*	1.1	IETF
A String Representation of Distinguished Names (RFC1779)		IETF
HTTP Authentication: Basic and Digest Access Authentication (RFC2617)		IETF
Key words for use in RFCs to Indicate Requirement Levels (RFC2119)		IETF
An Extension to HTTP: Digest Access Authentication (RFC2069)		IETF
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (RFC2045)	November, 1996	IETF
Transport Layer Security (TLS) (RFC 4346, RFC3280)	1.0 (1.1)	IETF
Secure Sockets Layer (SSL)	3.0	Netscape
The Directory: Public-key and attribute certificate frameworks (DER encoded X.509)	May, 2000	ITU-T

Table 92: Standards Dependencies for SMI-S

Standard	Version	Organization
PKCS #12: Personal Information Exchange Syntax	1.0	RSA Laboratories
Storage Management (IS24775-2006)	1.0	SNIA

7.1 Introduction to Profiles

7.1.1 Profile Content

7.1.1.1 Profile Definition

A profile is a specification that defines the CIM model and associated behavior for an autonomous and self-contained management domain. The CIM model includes the CIM Classes, Associations, Indications, Methods and Properties. The management domain is a set of related management tasks. A profile is uniquely identified by the name, organization and version.

In SMI-S, a profile describes the management interfaces for a class of storage subsystem, typically realized as a hardware of software product. For example, SMI-S includes profiles for arrays, FC-Switches, and logical volume manager software. The boundaries chosen for SMI-S profiles are often those of storage products, but some vendors may package things differently. For example, one vendor may choose to package an Array and an FC Switch into a single product; this can be handled in SMI-S by implementing the Array and FC Switch profiles for this product.

A profile may add restrictions to usage and behavior, but cannot change CIM defined characteristics. For example, if a property is required in the CIM model, then it is required in a profile. On the other hand, a profile may specify that a property is required even if it is not required by the general CIM model.

In SMI-S, profiles serve several purposes:

- Specification organization - the SMI-S object model (see Clause 8:, "Object Model") is presented as a set of profiles, each describing a type of storage element or behavior,
- Certification - SMI-S profiles form the basis for CTP certification,
- Discovery- profiles are registered with the CIM Server and advertised to clients as part of the CIM model and using SLP (see Clause 10:, "Service Discovery"). An SMI-S client uses SLP to determine which CIM Servers host profiles it wishes to manage, then uses the CIM model to discover the actual configurations and capabilities.

A subprofile is a profile that specifies a subset of a management domain. A subprofiles's CIM elements are scoped within a containing profile. Multiple profiles may use the same subprofile. A subprofile is uniquely identified by the name, organization and version.

A profile specification may include a list of the subprofiles it uses. The included subprofiles may be optional or mandatory by the scoping profile. The behavior of a profile is specified in this profile and its included subprofiles.

For example, target devices such as RAID arrays and tape libraries may support Fibre Channel or parallel SCSI connectivity. SMI-S includes an FC Target Port Subprofile and an Parallel SCSI Target Port Subprofile that may optionally be supported by profiles representing target devices. The elements defined in the port subprofiles are scoped to the ComputerSystem in the profile. For example, each LogicalPort subclass has a SystemDevice association to the profile's ComputerSystem.

In addition to sharing the purposes of profiles (above), subprofiles have these purposes:

- Optional behavior - a profile may allow, but not require, an implementation to support a subprofile. Although a subprofile does not describe a full product, a subprofile should describe an aspect of a product that is recognizable to an knowledgeable end-user such as a storage administrator,
- Reuse of functionality - some storage management behavior is common across different types of storage elements. For example, block virtualization is managed similarly in RAID arrays and logical volume managers. These common sets of functionality are specified as profiles that are shared by several other profiles.

- Decomposition - certain functionality may not be reused multiple places, but is complicated enough to document as a separate profile. For example, Disk Partition management is only used in the Host Discovered Resources profiles, but is complicated enough that it has been documented as a separate profile.

Terminology

A profile collects included subprofiles and provides the filler needed to define the management interfaces of a particular type of subsystem. Profiles are separated into two groups. *Storage profiles* define the management interfaces for storage subsystems such as arrays or FC switches. *Generic profiles* define management interfaces for generic systems that are related to storage management. Storage and generic profiles are specified the same way in SMI-S, but generic profiles are not certified as free-standing entities, only as a dependency of a storage profile.

A *Package* is a profile that whose implementation is mandatory to comply with the requirements of all of its containing top-level profiles. Since a package is always mandatory, it is not registered with the CIM Server. Packages provide decomposition in the specification.

Profiles may be related by *specialization* - where several profiles (or subprofiles) share many common elements, but are specialized for specific implementations. The SMI-S Security profiles are an example; the specializations (Authorization Profile, Security Resource Ownership Profile,...) share some classes and behavior. Profile specialization is only an artifact of the specification. It saves the reader from reading common aspects in multiple places and help the specification stay consistent across the specialized profiles. There is no information in the CIM model about the relationship between generic and specialized profiles.

7.1.1.2 Format for Profile Specifications

For each profile there is a set of information that is provided to specify the characteristics and requirements of the profile. Subprofiles are also defined using this format, but they are clearly identified as subprofiles.

Each profile or subprofile is defined in subsections that are described in Table 93.

Note: CIM schema diagrams are logically part of a profile description. However, they can be rather involved and cannot be easily depicted in a single diagram. As a result the reader is advised to refer to DMTF characterizations of CIM schema diagrams.

Table 93: Profile Components (Sheet 1 of 3)

Profile Element	Goal
Description	This section provides a description of the profile and model including an overview of the objectives and functionality. Functionality is described in a bullet-form in this section that includes functionality provided by the subprofiles referenced by the profile. If a function is provided by a subprofile, this is indicated, including whether the subprofile is optional or required. Functionality listed in the profile is organized by Levels, and within each Level by FCAPS category, as defined in the SMI-S functionality matrix section <link>. Instance Diagrams: One or more instance diagrams to highlight common implementations that employ this section of the Object Model. Instance diagrams also contain classes and associations but represent a particular configuration; multiple instances of an object may be depicted in an instance diagram. Finally, This section may include supporting text for recipes, properties, and methods as needed.

Table 93: Profile Components (Sheet 2 of 3)

Profile Element	Goal
Health & Fault Management	If a profile provides optional Health & Fault Management capabilities, then this section describes the specifics of these capabilities, including: • A table of the classes that report health information • Tables of possible states of the OperationalStatus and HealthState attributes and descriptions for those elements that report state. • Cause and Effect associations. • Standard Errors produced (including Alert Indications, Errors, CIM Errors, and Health Related Live Cycle Events.
Cascading Considerations	A Profile may be a cascading profile. A **cascading profile** is any Profile that supports the Cascading Subprofile as either a mandatory or recommended subprofile. If the profile is a cascading profile, this section documents cascading considerations in each of the following areas: • Cascaded Resources – Defines the type of resources in the Cascading Profile that are associated to what type of resources in the Leaf Profile and the association. • Ownership Privileges – Identifies the Resource Control Privileges (on leaf resources) that are established by the Cascading Profile. • Limitations on Cascading Subprofile – Identifies any limitations on the Cascading Subprofile that are imposed by the Cascading in effect
Supported Subprofiles and Packages	A list of the names and versions of subprofiles and packages supported by a profile.
Methods of the Profile	This section documents the methods used in this profile. All methods used in recipes shall be documented; optional methods (those not used in recipes) may also be included.
Recipes and Client Considerations	This section documents a set of "recipes" that describe the CIM operations and other steps required to accomplish particular tasks. These recipes do not define the upper bound of what a CIM Server may support, however, they define a lower bound. That is, a CIM Provider implementation shall support these recipes as prescribed to be SMI-S compliant. **Note:** A recipe that is defined as part of a subprofile is only required if the subprofile is implemented. All optional behavior in a profile shall be described in a recipe and shall have a capabilities property a client can test to determine whether the optional behavior is supported. The actual capabilities properties are documented in the Classes Used section.
CIM Server Requirements	A list of requirements on the CIM Server necessary to support the profile and its subprofiles.

Table 93: Profile Components (Sheet 3 of 3)

Profile Element	Goal
CIM Elements	A table listing the classes, associations, subprofile, packages, and indication filters that this profile (or subprofile) supports, and a brief description of each. Everything listed in this section is mandatory for the profile or subprofile. This section shall not list optional elements. Prior to this version of SMI-S, CIM did not have standard language for indication filters; IS24775-2006, *Storage Management* used the proposed WQL query language. The current version of SMI-S uses the CQL standard query language. WQL is also supported for backward compatibility. The Description column for an indication filter specifies whether the filter string is compliant to CQL or WQL. If neither is stated, then the string complies to both CQL and WQL.
Classes Used in the Profile	This section provides one table per class and lists each required and recommended property. For each required or recommended property a brief description on what information is to be encoded is identified. The class tables include a “Flags” column. This can contain “C” (the property is a correlatable name or a format for a name), “D” (the property is a durable name), “M” (the property is modifiable), or “N” (null is a valid value).
Dependencies on Other Standards	A table listing other standards that this profile and its subprofiles are dependent on.

Clause 8: Object Model

8.1 Registry of Profiles and Subprofiles

Each profile and subprofile within the SNIA Storage Management Initiative is identified by a unique name, selected and maintained by the SNIA, to assure that SMI implementers do not encounter any namespace collisions. The registry of these names, and a reference to their definition within this specification, are summarized in Table 94

Table 94: Registry of Profiles and Subprofiles.

Area	Registered Profile Name	Registered Subprofile Names
Server	Server	Object Manager Adapter Subprofile Indication Subprofile
Security	Security	Security Credential Management Subprofile Security Identity Management Subprofile Security Authorization Subprofile
Fabric	Fabric	Zone Control Subprofile Enhanced Zoning and Enhanced Zoning Control Sub-profile FDMI Subprofile Fabric Path Performance Subprofile
	Switch	Blades Subprofile Access Points Subprofile Software Installation Subprofile Multiple Computer System Subprofile Switch Configuration Data Subprofile Physical Package Package Software Package
	Extender	Physical Package Package Software Package
Host	FC HBA	FC Initiator Ports Subprofile
	iSCSI Initiator	iSCSI Initiator Ports Subprofile
	Host Discovered Resources	SCSI Multipath Management Subprofile Disk Partition Subprofile

Table 94: Registry of Profiles and Subprofiles.

Area	Registered Profile Name	Registered Subprofile Names
Storage	Array	Access Points Subprofile Block Server Performance Subprofile Cluster Subprofile Extra Capacity Set Subprofile Disk Drive Subprofile Disk Drive Lite Subprofile Extent Mapping Subprofile Extent Composition Subprofile Location Subprofile Software Subprofile Copy Services Subprofile Job Control Subprofile Pool Manipulation Capabilities and Settings Subprofile LUN Creation Subprofile Device Credentials Subprofile LUN Mapping and Masking Subprofile Masking and Mapping Subprofile SPI Target Ports Subprofile FC Target Ports Subprofile iSCSI Target Ports Subprofile Backend Ports Subprofile Disk Sparing Subprofile FC Initiator Ports Subprofile SPI Initiator Ports Subprofile Block Services Package Physical Package Package Health Package
	Storage Virtualizer	Access Points Subprofile Copy Services Subprofile Job Control Subprofile Location Subprofile Masking and Mapping Subprofile Software Subprofile Multiple Computer System Subprofile FC Initiator Ports Subprofile iSCSI Target Ports Subprofile FC Target Ports Subprofile iSCSI Initiator Ports Subprofile Extent Composition Subprofile Cascading Subprofile Block Services Package Physical Package Package

Table 94: Registry of Profiles and Subprofiles.

Area	Registered Profile Name	Registered Subprofile Names
	Volume Management	Access Points Subprofile Extent Composition Subprofile Location Subprofile Software Subprofile Copy Services Subprofile Disk Sparing Subprofile Multi System Subprofile Job Control Subprofile Cascading Subprofile Block Storage Resource Ownership Subprofile Block Server Performance Subprofile Block Services Package Health Package
	Storage Library	Access Points Subprofile Location Subprofile FC Target Ports Subprofile Software Subprofile Storage Library Limited Access Port Elements Subprofile Storage Library Media Movement Subprofile Storage Library Capacity Subprofile Storage Library Element Counting Subprofile Storage Library InterLibraryPort Connection Subprofile Storage Library Partitioned Library Subprofile Physical Package Package
	NAS Head	Indication Subprofile Cascading Subprofile Access Points Subprofile Multiple Computer System Subprofile Software Subprofile Location Subprofile Extent Composition Subprofile File System Manipulation Subprofile File Export Manipulation Subprofile Job Control Subprofile SPI Initiator Ports Subprofile FC Initiator Ports Subprofile Device Credentials Subprofile Physical Package Package Block Services Package Health Package

Table 94: Registry of Profiles and Subprofiles.

Area	Registered Profile Name	Registered Subprofile Names
	Self-contained NAS System	Indication Subprofile Access Points Subprofile Multiple Computer System Subprofile Software Subprofile Location Subprofile Extent Composition Subprofile File System Manipulation Subprofile File Export Manipulation Subprofile Job Control Subprofile Disk Drive Lite Subprofile SPI Initiator Ports Subprofile FC Initiator Ports Subprofile iSCSI Initiator Ports Subprofile iSCSI Target Ports Subprofile Device Credentials Subprofile Physical Package Package Block Services Package Health Package

8.2 Packages, Subprofiles and Profile

8.2.1 Common Profiles

Common profiles (including subprofiles and packages) are shared by multiple profiles

8.2.1.1 Access Points Subprofile

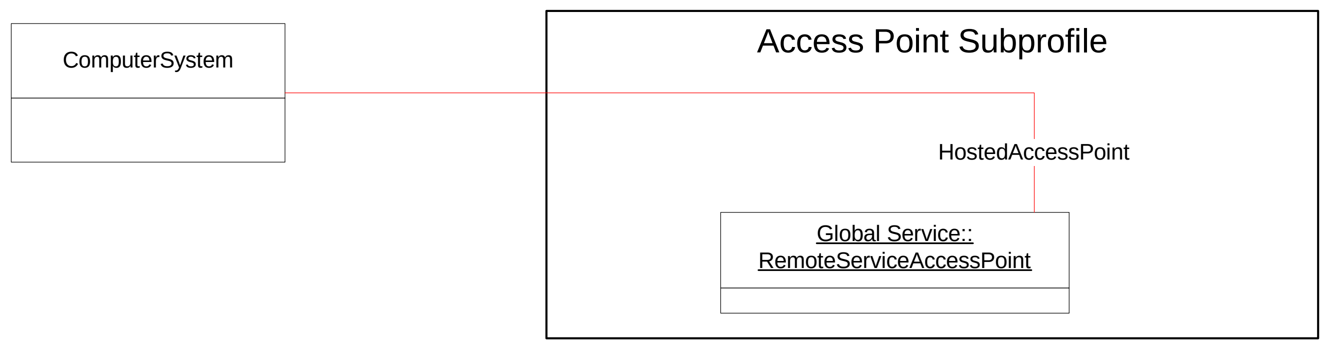
8.2.1.1.1 Description

The Access Points subprofile provides addresses of remote access points for management services.

This is modeled using a RemoteServiceAccessPoint linked to the managed system using a HostedAccessPoint association.

A management service is typically associated with all elements in a system, but in some cases, a management service relates to a subset of elements. The scope of a RemoteServiceAccessPoint may be constrained to a subset of elements using SAPAvailableForElement. If the service referenced in RemoteServiceAccessPoint is not referenced by any SAPAvailableForElement associations, then the service described by RemoteServiceAccessPoint shall apply to all the elements of the system referenced via HostedAccessPoints. This type of system-wide service is depicted in Figure 22: "System-wide Remote Access Point".

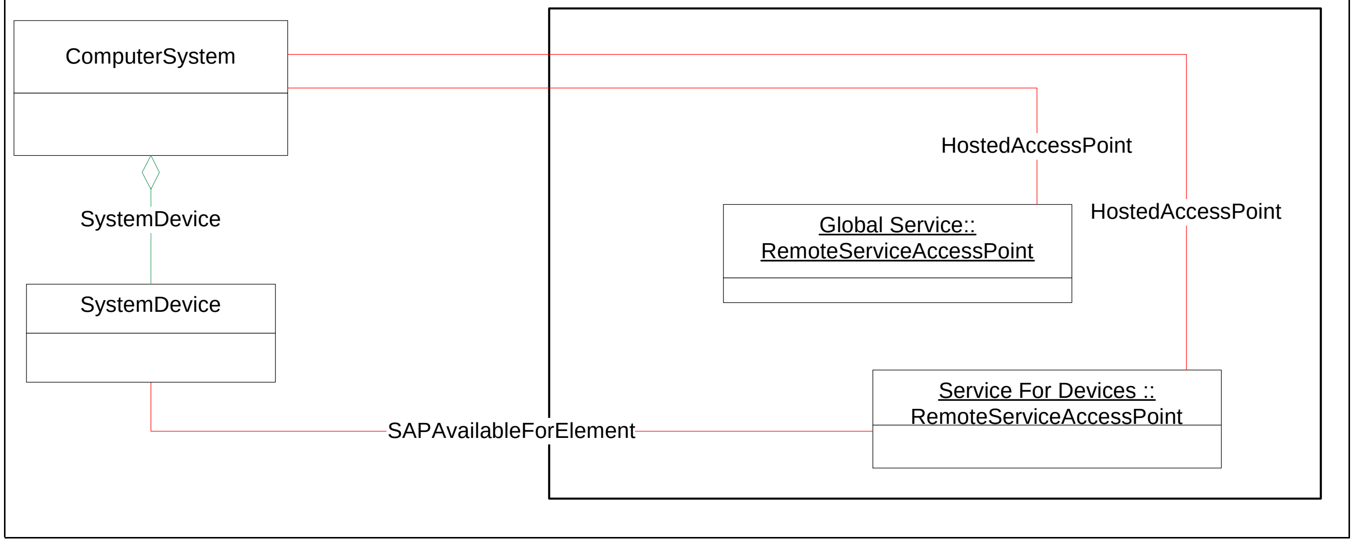
Figure 22: System-wide Remote Access Point



If the service referenced in RemoteServiceAccessPoint is referenced by any SAPAvailableForElement associations, then the service described by RemoteServiceAccessPoint shall apply to the subset of elements referenced via SAPAvailabelForElement associations. The HostedAccessPoint association between RemoteServiceAccessPoint is still mandatory (so the client can readily associate the service to a specific storage system).

Figure 23: "Access Point Instance Diagram" depicts a configuration with two RemoveServiceAccessPoint instances. One represents a system-wide service and the other represents a service that applies just to certain devices.

Figure 23: Access Point Instance Diagram



The exposed management services may represent a web UI that can be launched by a web browser, a telnet interface, or some vendor-specific interface. RemoteServiceAccessPoint InfoFormat property describes the format of the AccessInfo property; valid options include “URL” and FQDN”. In a URL, the text before the “://” is referred to as the “scheme”. A URL with an http or HTTPS scheme is often a web/ HTML page, but HTTP can be used for other purposes. Table 95 specifies the requirements for InfoFormat, AccessInfo, and the scheme subset of a URL AccessInfo.

Table 95: RemoteAccessPoint InfoFormat and AccessInfo Properties

InfoFormat	AccessInfo Scheme	Description
“URL”	“http” or “https”	The references URL shall be a valid web page. It should provide element management for the system or elements referenced by the associated HostedAccessPoint association.
“Other” with OtherInfoFormatDescription = "Non-UI URL"	“http” or” https”	Used for HTTP URLs that do not reference a valid web UI.
“URL”	anything other than “http” and “https”	May be used. No standard behavior is specified.
others from the MOF	n/a	May be used. No standard behavior is specified.

8.2.1.1.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.1.3 Cascading Considerations

Not defined in this standard.

8.2.1.1.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.1.1.5 Methods of this Profile

Not defined in this standard.

8.2.1.1.6 Client Considerations and Recipes

Not defined in this standard.

8.2.1.1.7 Registered Name and Version

Access Points version 1.1.0

8.2.1.1.8 CIM Server Requirements

Table 96: CIM Server Requirements for Access Points

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.1.9 CIM Elements

Table 97: CIM Elements for Access Points

Element Name	Description
Mandatory Classes	
CIM_HostedAccessPoint (8.2.1.1.9.1)	Associate the RemoteServiceAccessPoint to the System on which it is hosted.
CIM_RemoteServiceAccessPoint (8.2.1.1.9.2)	A ServiceAccessPoint for management tools
Optional Classes	
CIM_SAPAvailableForElement (8.2.1.1.9.3)	This association identifies the element that is serviced by the RemoteServiceAccessPoint

8.2.1.1.9.1 CIM_HostedAccessPoint

Associate the RemoteServiceAccessPoint to the System on which it is hosted.
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 98: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System
Dependent		CIM_ServiceAccessPoint	The SAP(s) that are hosted on this System.

8.2.1.1.9.2 CIM_RemoteServiceAccessPoint

A ServiceAccessPoint for management tools
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 99: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
ElementName		string	User-friendly name
AccessInfo		string	Management Address.
InfoFormat		uint16	The format of the Management Address. For interoperability, this shall be URL'(200).'

8.2.1.1.9.3 CIM_SAPAvailableForElement

This association identifies the element that is serviced by the RemoteServiceAccessPoint
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 100: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AvailableSAP		CIM_ServiceAccessPoint	The Service that is available.
ManagedElement		CIM_ManagedElement	The ManagedElement that may use the Service.

8.2.1.1.10 Related Standards

Table 101: Related Standards for Access Points

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.1.2 Cascading Subprofile

8.2.1.2.1 Description

The cascading subprofile defines the set of classes, methods and behavior used to model cross profile dependencies and references. This includes modeling of cross CIM server references when the referenced profile is managed by another CIM server.

Examples of SMI-S Profiles that should support the Cascading Subprofile include Storage Virtualizer, NAS Heads and Volume Managers. However, other profiles may also support the cascading subprofile for cross profile references. For example, if an Array Profile may support the cascading subprofile to effect cross profile references used in “remote copy.”

For ease of documentation, a profile that supports the cascading subprofile is referred to as a **Cascading Profile**. The profile referenced is referred to as a **Leaf Profile**. For example, storage virtualization would support the cascading subprofile and would be a Cascading Profile. It would reference storage volumes in one or more Array profiles. In such configurations, the Array profiles would be referred to as Leaf profiles.

The cascading subprofile defines a common approach to “stitching” resources in the cascading profile to resources in the leaf profiles. While the general mechanism used is common, the specifics may vary depending on the resources that are stitched together. For example, a Storage Virtualization Profile would stitch StorageExtents (in the virtualizer) to StorageVolumes (in arrays). But a Volume Manager would stitch LogicalDisks (in the volume manager) to StorageVolumes (in arrays or virtualizers).

The cascading subprofile defines how to model the relationships between CIM Servers when there are CIM Servers of Leaf profiles that are referenced by a CIM Server of the cascading profile, and how a client manages the interaction between CIM Servers in a cascading configuration (including CIM Server credentials).

In addition to the Cascading subprofile, there are two related subprofiles that may also be supported by the cascading profile or the leaf profiles. They are the Credential Management Subprofile, which defines the classes, methods and behavior for managing the credentials used by a CIM server of the cascading profile when accessing (different) CIM Servers of Leaf profiles. The second is the Security Resource Ownership Subprofile (or a specialization of this subprofile) which defines the classes, methods and behavior of recording ownership in the leaf profiles. The usage of these subprofiles will be referenced in this subprofile, but their definition is contained in separate subprofile specifications.

The Cascading Subprofile provides block-level configuration management in this version of SMI-S.

The Cascading Subprofile defines cascading of resources at the block level. That is, a Cascading Profile uses Block storage resources of the leaf profiles. These are StorageVolumes or LogicalDisks. In this version of SMI-S the model will only be tested in the context of cascading for block storage.

8.2.1.2.1.1 Instance Diagrams

There are three aspects of the cascading subprofile that are illustrated separately:

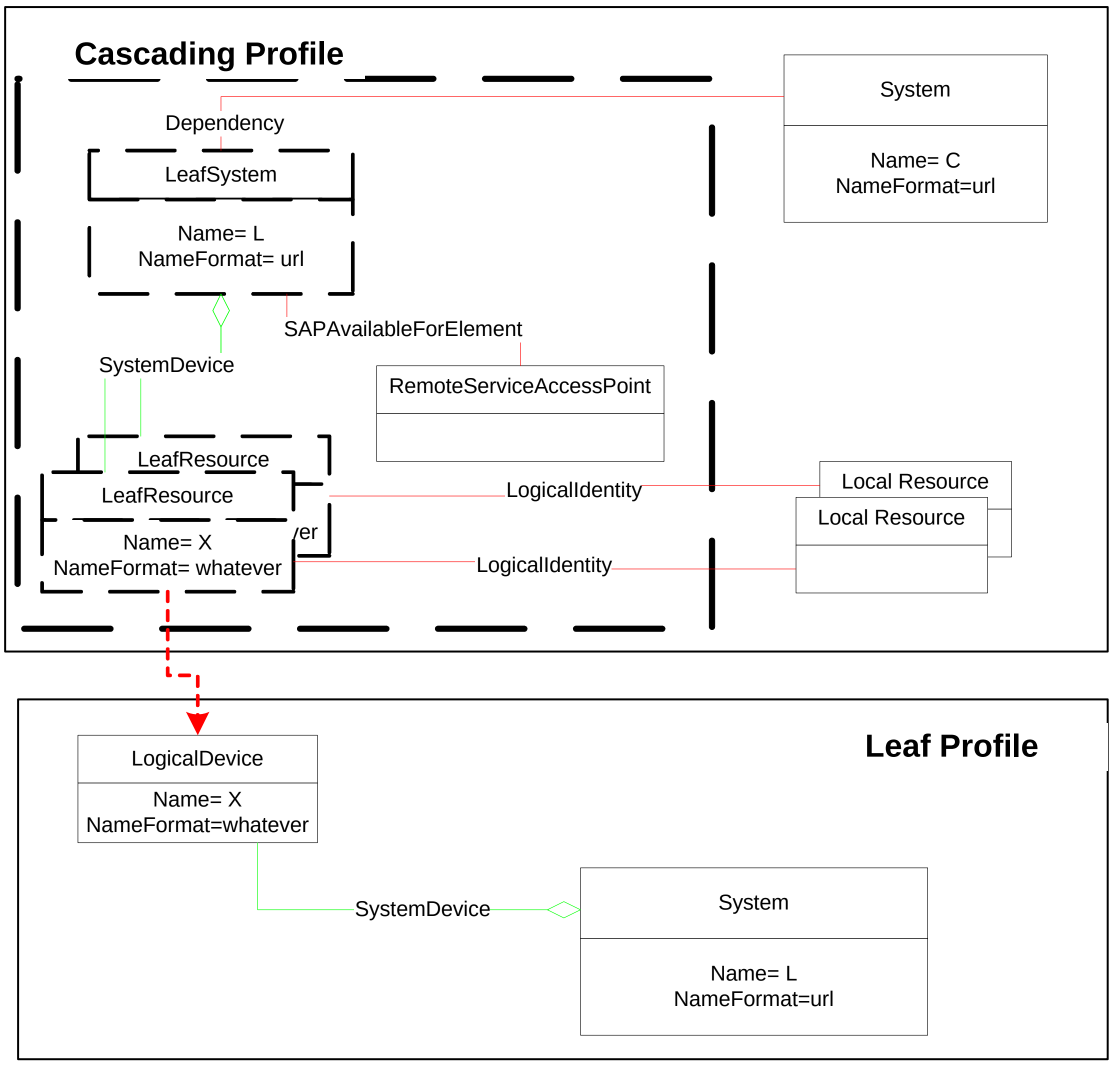
- Logical Topology (usage of leaf resources by cascading profiles)
- Resource Allocation/Deallocation
- CIM Server Topology (usage of CIM Servers by other CIM Servers)

In addition, there are the relationships between the Cascading subprofile and the Security Resource Ownership Subprofile and the Credential Management Subprofile. This relationship will be illustrated, but the details of those subprofiles are documented in their own sections.

Logical Topology

Figure 24: "Instance Diagram for Logical Topology" illustrates the basic constructs for modeling the logical topology represented by cascading profiles. The cascading profile is the top box. The modeling for the cascading subprofile is in the dashed box (in the Cascading Profile). The leaf profile is the lower box. Note that for the basic modeling of the logical topology of cascading, there are no modeling requirements on the leaf profile.

Figure 24: Instance Diagram for Logical Topology



NOTE: The dashed classes in Figure 24: "Instance Diagram for Logical Topology" are instances that are cached in the Cascading Profile. They are redundant with the instances maintained by the Leaf profile. The dashed arrows between the Cacsading Profile and the Leaf Profile signifies “stitching” based on durable names or correlatable ids for the resources represented. The dashed arrows **are not** instantiated associations.

If the Cascading Subprofile is supported by the Cascading Profile, then there will be support for instantiating “leaf” “top level object” (e.g., ComputerSystems) and “leaf” LogicalDevices (e.g., StorageVolumes) in those Leaf Profiles that are “visible” to the Cascading Profile (device). The instances of the “leaf” “top level object” can be found by traversing the CascadingDependency association from the “top level object” of the Cascading Profile.

The leaf resources (logical devices) that are visible to the Cascading Profile have an association (e.g., SystemDevice association) to the “leaf” top level object (e.g., ComputerSystem) that has exposed them to the Cascading Profile.

The top level object, Hosted or SystemDevice association and LogicalDevices mirrors information that is in the Leaf Profile. In some Cascading Profile configurations, the Cascading Profile may want to subscribe to life cycle indications on the devices of interest in the Leaf Profile. However, that is a consideration of the Cascading Profile. It is not required as part of the Cascading Subprofile.

From the top level object (e.g., ComputerSystem) of the Leaf, there may be a SAPAvailableForElement association to a RemoteServiceAccessPoint instance. The RemoteServiceAccessPoint identifies information need for access to the management interface to the Leaf system. This management interface may or may not be a CIM interface.

The expectation is that the model represented in Figure 24: "Instance Diagram for Logical Topology" will be automatically maintained by the Cascading Profile (and providers). There are no methods for client manipulation of this model. In the case of the RemoteServiceAccessPoint instance, the expectation is that discovery of leaf systems would be an automatic process (e.g., SLP discovery of SMI-S Profiles and Servers) and that the provider would record the access information based on its discovery processes.

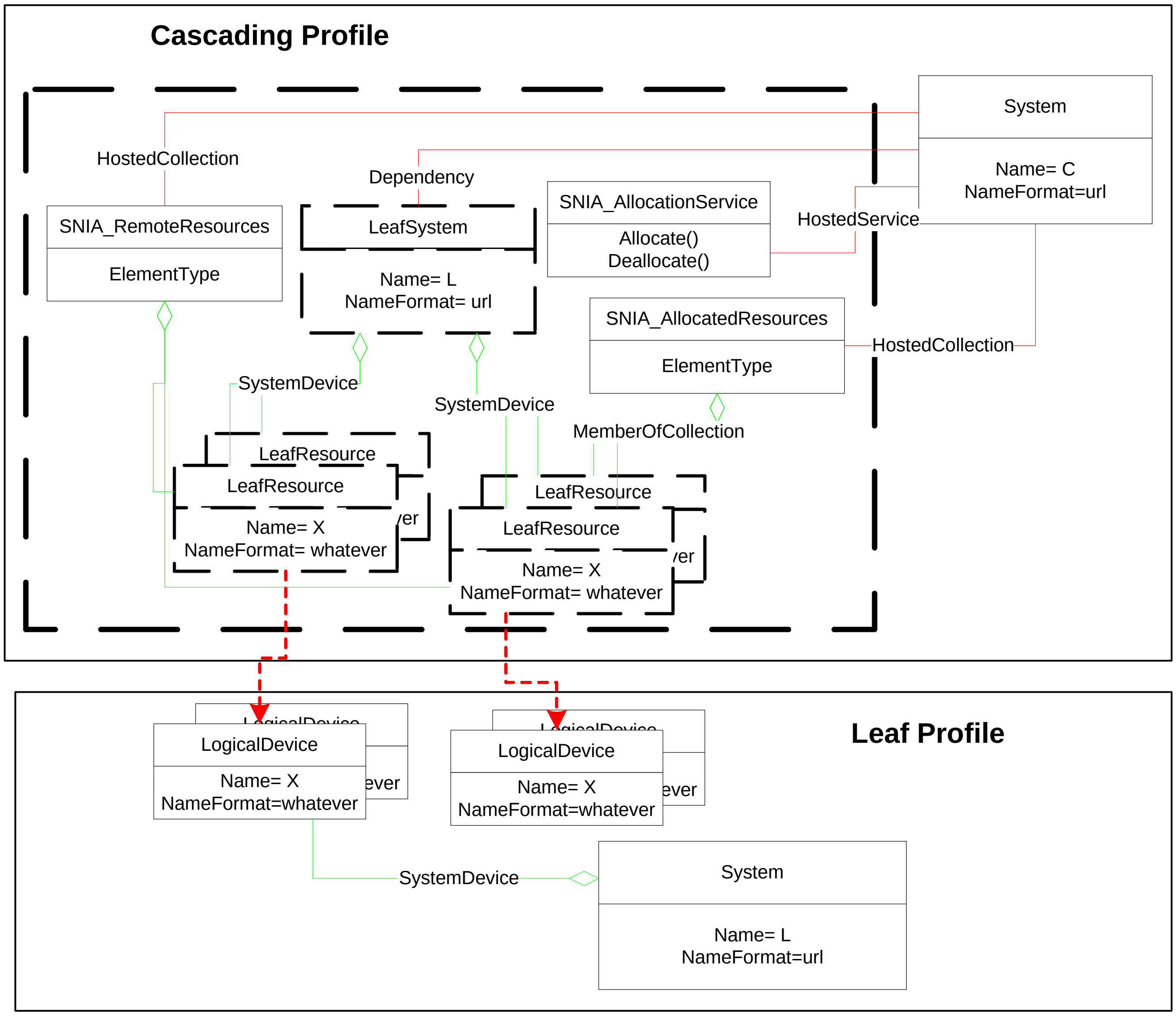
In the simplest form of cascading, this is sufficient to model the logical topology of the cascading. However, many implementations will need to go further (see “Resource Allocation/Deallocation” on page 127 in 8.2.1.2.1.1).

Resource Allocation/Deallocation

In some cascading environments, it is necessary to distinguish between resources that are “visible” to the Cascading Profile from resources that are actually “in use.” For example, a Volume Manager or storage virtualization system may be able to “see” a number of storage volumes (logical units) through its ports. But this does not necessarily mean that is has allocated and is using them. A separate step is required to “prepare” the resources for use. In the case of storage virtualization systems, this step would include assigning the storage to a storage pool in the virtualizer.

To readily discern which storage volumes (logical devices) are “visible” and which volumes are assigned, two collections are defined. The collection of “visible” resources is the “RemoteResources” collection. The collection of assigned resources is the “AllocatedResources” collection. This is illustrated in Figure 25: "Instance Diagram for Resource Allocation/Deallocation"

Figure 25: Instance Diagram for Resource Allocation/Deallocation



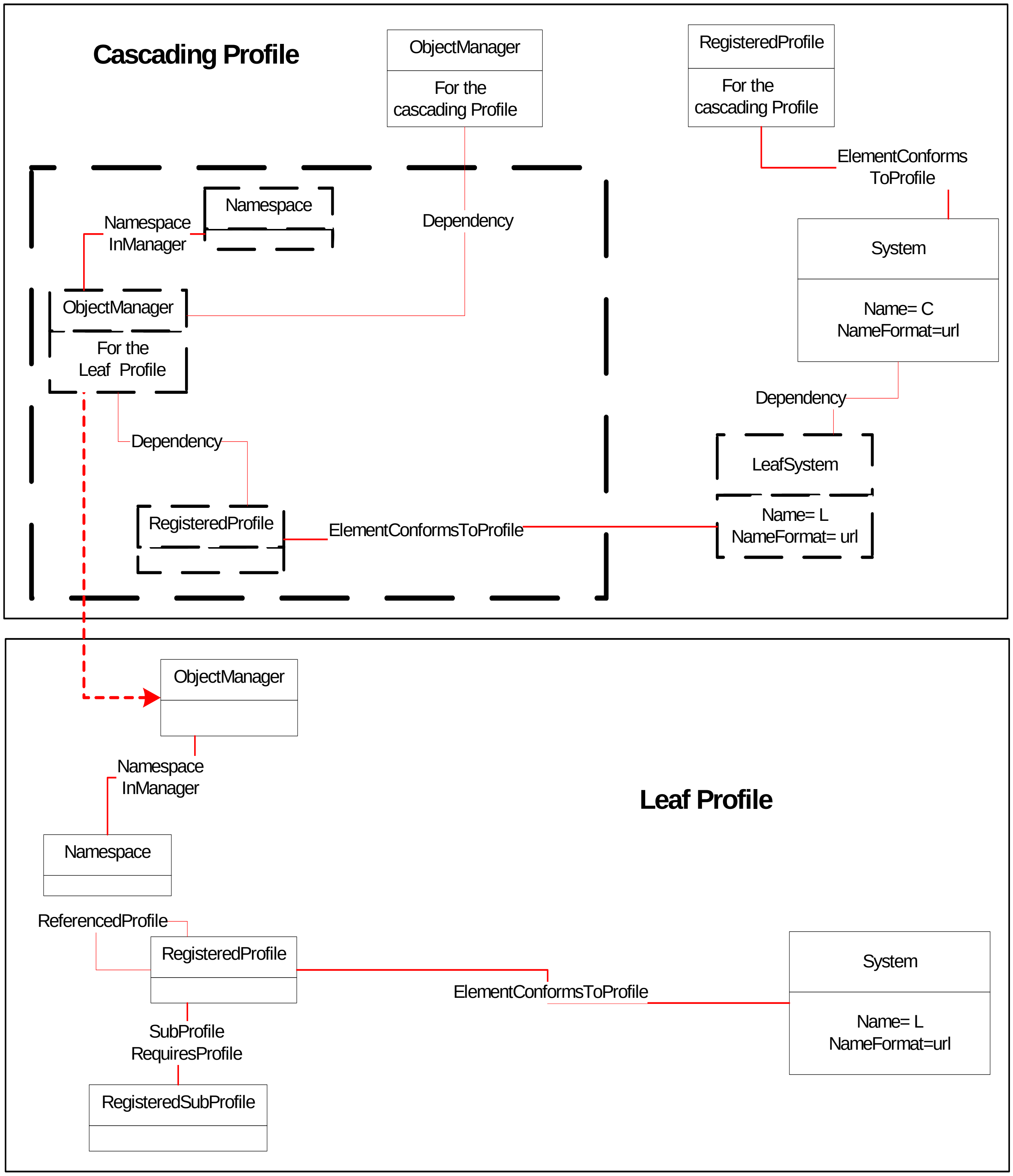
The SNIA_AllocationService may or may not exist. The actual function of Allocation may be implemented as a side effect of other methods. For example, allocating a Leaf StorageVolume may occur as a side effect of CreateOrModifyStoragePool, where the an extent (e.g., leaf StorageVolume) is added to a StoragePool. The semantics of CreateOrModifyStoragePool constructs all the necessary associations for the StorageExtent (and may also have the semantics of an implied allocation of the StorageVolume).

To determine if allocation or deallocation are explicit (via allocate/deallocate method calls) or implicit (side effect of another method), the client should inspect the “AsynchronousMethodsSupported” and “SynchronousMethodsSupported” properties of the SNIA_CascadingCapabilities instance for the System.

CIM Server Topology

In addition to a cascading system using leaf systems and its resources, a cascading profile may also model the dependencies between the CIM Server of the cascading profile and the CIM Servers of the Leaf Profiles. This is illustrated in Figure 26: "Cascading Server Topology".

Figure 26: Cascading Server Topology



As with the logical topology, the server topology is effected by caching Leaf information in the cascading profile. Specifically, the cached instances from the leaf profiles are:

ObjectManager – to allow the dependency between ObjectManagers to be instantiated in the cascading profile.

Namespace – to provide cached information on the namespace of the leaf CIM Server. This would be the Interop Namespace for accessing the Server Profile of the CIM Server.

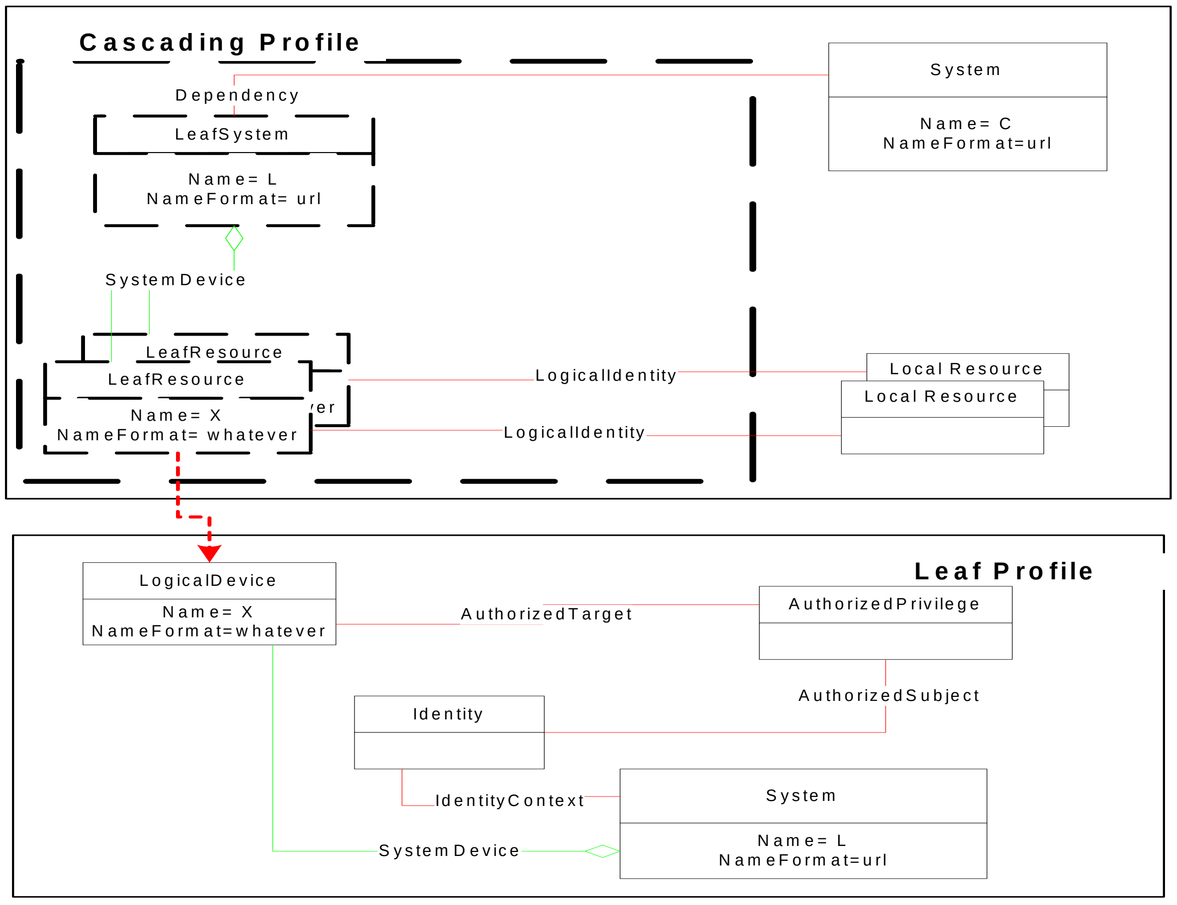
RegisteredProfile – to identify the Profile of the Leaf Profile (e.g., Array or Virtualizer).

In addition, the necessary associations (HostedProfile, NamespaceInManager and ElementConformsToProfile) would be instantiated to connect the relevant instances.

The actual dependence between the CIM Server (ObjectManager) of the Cascading Profile and the CIM Server (ObjectManager) of the Leaf systems is represented by instances of Dependency.

Cascading with the Resource Ownership Subprofile

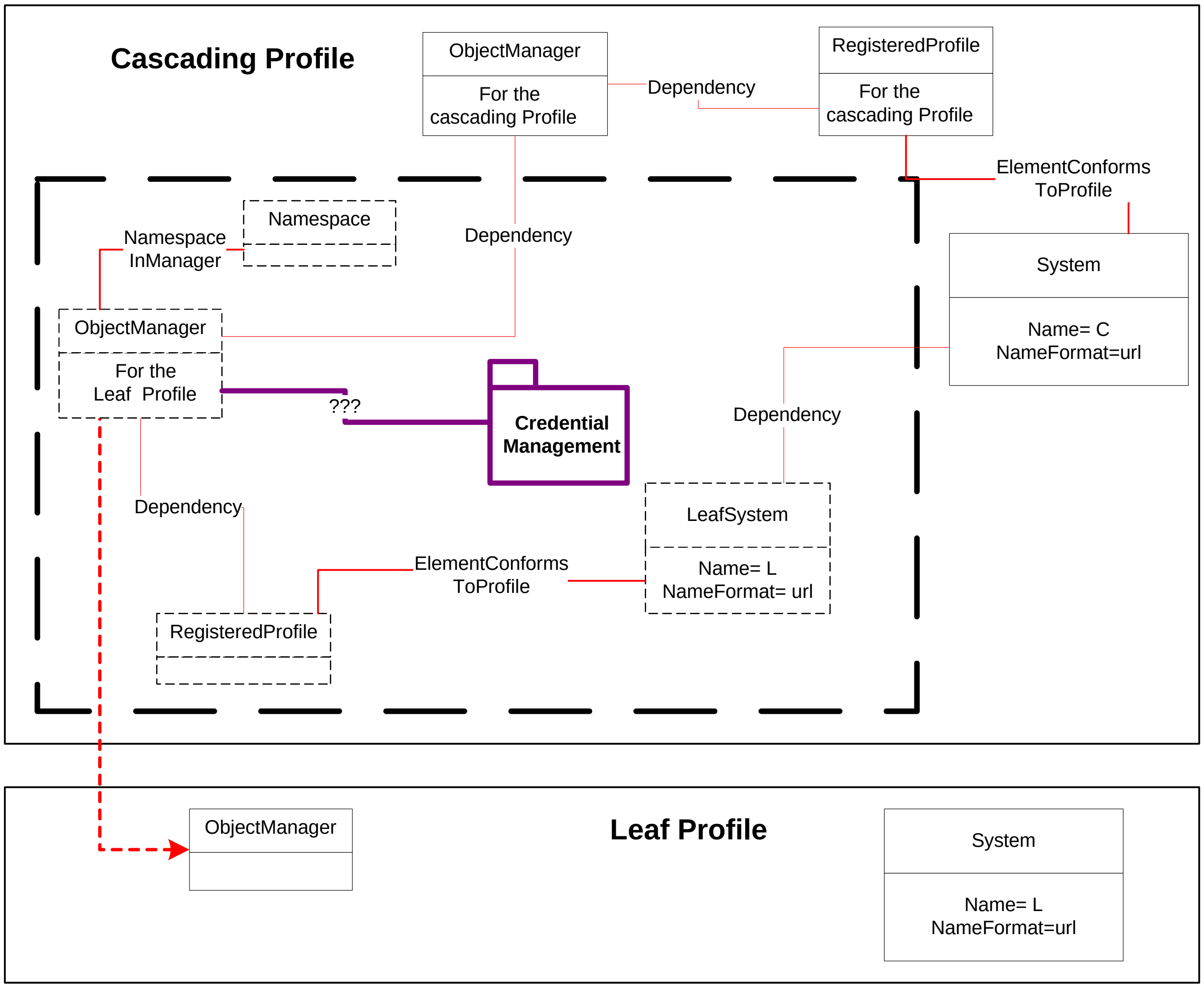
Figure 27: Instance Diagram for Cascading with Resource Ownership



Cascading with the Credentials Management Subprofile

As an extension of the modeling of CIM Server topology, a cascading profile may implement the Credentials Management Subprofile. When this is done it extends the modeling for the Server topology as illustrated in Figure 28: "Instance Diagram for Cascading with Credential Management Subprofile".

Figure 28: Instance Diagram for Cascading with Credential Management Subprofile

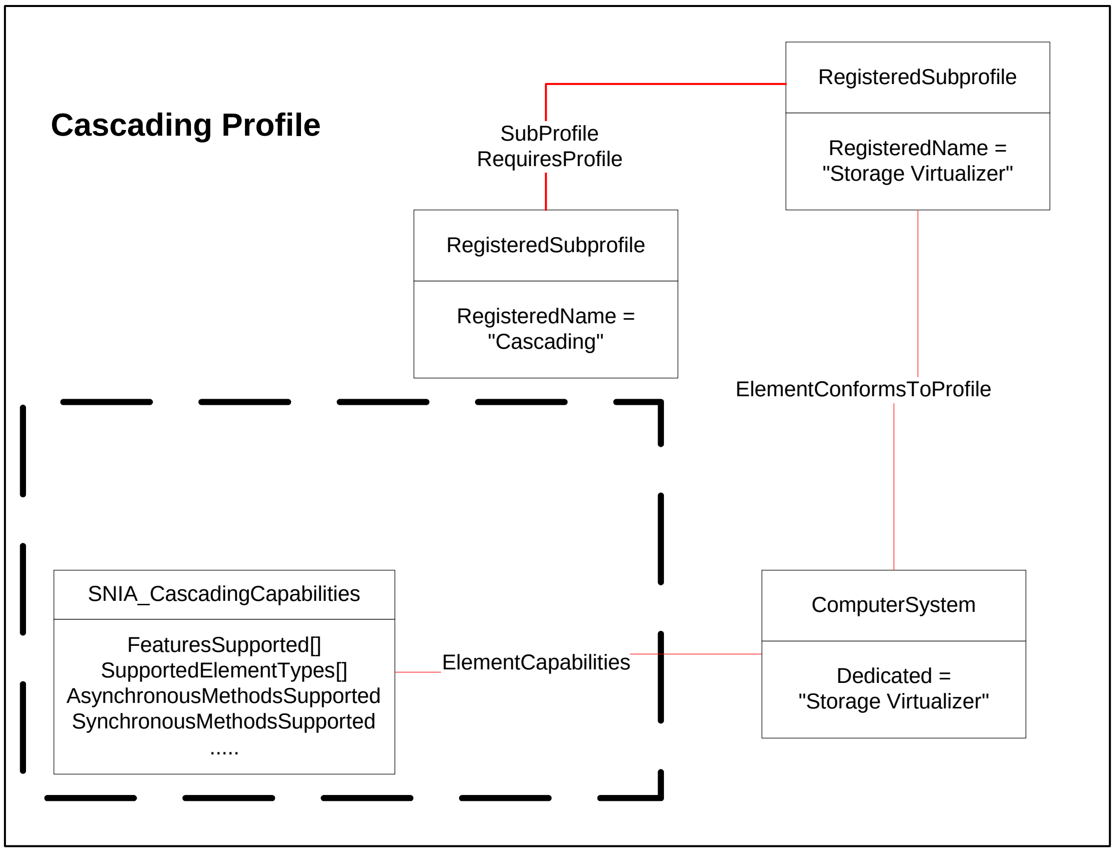


The Credential Management information would be associated with the CIM Server ObjectManager instance for a Leaf system. The Credential Management Subprofile would identify how the cascading system would authenticate itself with the Leaf system.

Modeling for Defining Cascading Capabilities

As indicated in previous discussions, only parts of the Cascading subprofile are mandatory. For a list of what elements are mandatory, see 8.2.1.2.9, "CIM Elements". In order to make it relatively easy for clients to determine what is supported, implementation of the SNIA_CascadingCapabilities class is mandatory if cascading is supported. The modeling for this class is illustrated in Figure 29: "Modeling of Cascading Capabilities".

Figure 29: Modeling of Cascading Capabilities


Cascading Profile
RegisteredSubprofile
RegisteredName = "Storage Virtualizer"
SubProfile RequiresProfile
RegisteredSubprofile
RegisteredName = "Cascading"
ElementConformsToProfile
SNIA_CascadingCapabilities
FeaturesSupported[]
SupportedElementTypes[]
AsynchronousMethodsSupported
SynchronousMethodsSupported
.....
ElementCapabilities
ComputerSystem
Dedicated = "Storage Virtualizer"

The SNIA_CascadingCapabilities instance would be found by doing association traversal from the RegisteredSubprofile for cascading following the ElementCapabilites association.

The properties of SNIA_CascadingCapabilities are defined as follows:

- FeaturesSupported - This is an array that defines the cascading features that are supported by the implementation of the Cascading Profile. The values are "Ownership", "Leaf Credentials", "OM Dependencies" and "Allocation Service".
- SupportedElementTypes - This is an array that defines the type of “Remote Resource” ManagedElements that are supported by the implementation. For this version of SMI-S, only StorageVolumes are supported.
- AsynchronousMethodsSupported – This is an array that defines any asynchronous methods supported for allocation or deallocation of leaf resources. The values are “Allocation” or “Deallocation”.
- SynchronousMethodsSupported – This is an array that defines any synchronous methods supported for allocation or deallocation of leaf resources. The values are “Allocation” or “Deallocation”.

The Cascading subprofile uses durable names of leaf resources for stitching together the Leaf Profile and its resources to the corresponding instances in the Cascading Profile.

The CIM Server of the Cascading Profile may use indications (or provider poll on access) to keep its model accurate.

8.2.1.2.2 Health and Fault Management Considerations

8.2.1.2.2.1 Reporting Health of Leaf Systems, Resources and Object Managers

A Cascading Profile should not report health of leaf resources without verifying the health of those resources (via direct reference to the Leaf Profile). The Cascading Profile may keep health properties in its local copy of the instances for leaf resources for its own purposes, but it should always refer to the leaf profile on requests from clients.

A request for a health property (e.g., OperationalStatus) should result in a request to the underlying leaf resource for the information. If the leaf resource is not available (e.g., the connection to the CIM Server is broken) the Cascading Profile may report health from its local copy of the instance.

8.2.1.2.2.2 Cascading Indications of Health

Given a Cascading Profile is dependent upon leaf resources, the CIM Server of the Cascading Profile may chose to subscribe to health (OperationalStatus) indications on the leaf resources it is actively using (allocated resources). Generally speaking, health problems on leaf resources will translate to health problems on one or more resources in the Cascading Profile. For example, if a StorageVolume in the Array (leaf) profile has an OperationalStatus of “Error”, this may cause one or more StorageVolumes in a Virtualizer that is using the array to either be in error or be degraded.

Health indications should cascade. However, how they cascade will depend on where and how the leaf resources are used.

However a cascading profile discovers a problem with leaf resources, then it may be reflected in operational status of the cascader’s resources.

8.2.1.2.3 Cascading Considerations

Not defined in this standard.

8.2.1.2.4 Supported Subprofiles and Packages

Table 102: Supported Subprofiles for Cascading

Registered Subprofile Names	Mandatory	Version
Security Resource Ownership	No	1.1.0
Credential Management	No	1.1.0

8.2.1.2.5 Methods of this Subprofile

Table 103 summarized the extrinsic methods supported by the Cascading Subprofile.

Table 103: Extrinsic Methods Supported by Cascading Subprofile

Method	Created Instances	Deleted Instances	Modified Instances
Allocate	MemberOfCollection	N/A	N/A
Deallocate	N/A	MemberOfCollection	N/A

8.2.1.2.5.1 Extrinsic Methods of this Profile

There are two extrinsic Methods that may be supported by an implementation of the cascading subprofile:

- Allocate
- Deallocate

8.2.1.2.5.1.1 Allocate

Starts a job to allocate remote resources (from the RemoteResources collection) to the AllocatedResources collection.

Allocate (

[IN, Description (Enumeration indicating the type of element being allocated. This type value shall match the type of the instances.),

ValueMap { "0", "1", "2", "3", "4", "5", "6", "7", "8" },

Values { "Unknown", "Reserved", "Any Type", "StorageVolume", "StorageExtent", "StoragePool", "ComputerSystem", "LogicalDisk", "FileShare" }]

Only “3” (StorageVolume) is supported in this version of SMI-S.

uint16 **ElementType**;

[IN (false), OUT, Description (Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, Description (The reference to the AllocatedResource collection to which Elements are being added.)]

SNIA_AllocatedResources REF **Collection**,

[IN, Description (Array of strings containing representations of references to CIM_ManagedElement instances, that are being allocated to the AllocatedResources Collection.)]

string **InElements[]**);

Error returns are:

{ "Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "In Use", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific" }]

8.2.1.2.5.1.2 Deallocate

Starts a job to remove remote resources (from the AllocatedResources collection) and return them to the RemoteResources collection.

Deallocate (

[IN, Description (Enumeration indicating the type of element being deallocated. This type value shall match the type of the instances.),

ValueMap { "0", "1", "2", "3", "4", "5", "6", "7", "8" }, Values { "Unknown", "Reserved", "Any Type", "StorageVolume", "StorageExtent", "StoragePool", "ComputerSystem", "LogicalDisk", "FileShare" }]

Only “3” (StorageVolume) is supported in this version of SMI-S.

uint16 **ElementType**;

[IN (false), OUT, Description (Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, Description (The reference to the AllocatedResource collection from which Elements are being removed.)]

SNIA_AllocatedResources REF **Collection**,

[IN, Description (Array of strings containing representations of references to CIM_ManagedElement instances, that are being deallocated from the AllocatedResources Collection.")]

string **InElements[]**);

Error returns are:

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

8.2.1.2.5.2 Intrinsic Methods of this Profile

None.

8.2.1.2.6 Client Considerations and Recipes

8.2.1.2.6.1 Recipe MPCP01: Determining Resources used by cascading Profiles

This recipe is not defined in this standard. It will be included in a future revision, based on implementation experience.

8.2.1.2.6.2 Recipe MPCP02: Monitoring the existence of Cascading Profiles

This recipe is not defined in this standard. It will be included in a future revision, based on implementation experience.

8.2.1.2.6.3 OPTIONAL: Recipe MPCP03: Allocation of Leaf Resources

This recipe is not defined in this standard. It will be included in a future revision, based on implementation experience.

8.2.1.2.6.4 OPTIONAL: Recipe MPCP04: Deallocation of Leaf Resources

This recipe is not defined in this standard. It will be included in a future revision, based on implementation experience.

8.2.1.2.6.5 Recipe MPCP05: Monitoring the existence of “Stitching” between Profiles

This recipe is not defined in this standard. It will be included in a future revision, based on implementation experience.

8.2.1.2.6.6 Supported SNIA_CascadingCapabilities Patterns

The SNIA_CascadingCapabilities patterns shown in Table 104: "Cascading Capabilities Patterns" are formally recognized and supported by this version of SMI-S.

Table 104: Cascading Capabilities Patterns

FeaturesSupported	SupportedElementTypes	SynchronousMethods Supported	AsynchronouosMethods Supported
none	StorageVolume	none	none
Ownership, Leaf Credentials, OM Dependencies, Allocation Service	StorageVolume	Allocation Deallocation	Allocation Deallocation
Allocation Service	StorageVolume	Allocation Deallocation	none
Allocation Service	StorageVolume	none	Allocation Deallocation
Ownership, Leaf Credentials, OM Dependencies	StorageVolume	none	none

8.2.1.2.7 Registered Name and Version

Cascading version 1.1.0

8.2.1.2.8 CIM Server Requirements

Table 105: CIM Server Requirements for Cascading

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.2.9 CIM Elements

Table 106: CIM Elements for Cascading

Element Name	Description
Mandatory Classes	
CIM_Dependency (8.2.1.2.9.1)	(System Dependency) This associates the Leaf System to the Cascading System.
CIM_ElementCapabilities (8.2.1.2.9.4)	This associates the CascadingCapabilities to the cascading system (e.g., ComputerSystem).
CIM_HostedCollection (8.2.1.2.9.6)	(AllocatedResources) This would associate the AllocatedResources collection to the top level system for the Cascading Profile (e.g., Storage Virtualizer).
CIM_MemberOfCollection (8.2.1.2.9.9)	(Allocated Resources) This supports collecting leaf resources. This is required to support the AllocatedResources collection.
CIM_ObjectManager (8.2.1.2.9.14)	(Cascading Profile) This is the Object Manager service of the CIM Server.
CIM_SystemDevice (8.2.1.2.9.18)	This association links **LogicalDevice** remote resources to the scoping system. This is used to associate the remote resources with the System that manages them.
SNIA_AllocatedResources (8.2.1.2.9.19)	This is a SystemSpecificCollection for collecting leaf resources that have been deployed for use in the Cascading profile (e.g., StorageVolumes assigned to a virtualizer's StoragePool).
SNIA_CascadingCapabilities (8.2.1.2.9.21)	This defines the cascading capabilities supported by the implementation of the profile.
Optional Classes	
CIM_Dependency (8.2.1.2.9.2)	(ObjectManager Dependency) This associates the Object Manager of the Leaf System to the Object Manager of the Cascading System.
CIM_Dependency (8.2.1.2.9.3)	(RegisteredProfile Dependency) This associates the RegisteredProfile of a leaf system to the Object Manager of the leaf system.

Table 106: CIM Elements for Cascading

Element Name	Description
CIM_ElementConformsToProfile (8.2.1.2.9.5)	(Leaf) This associates the RegisteredProfile of the Leaf Profile to the Leaf system (e.g., ComputerSystem).
CIM_HostedCollection (8.2.1.2.9.7)	(RemoteResources) This would associate the RemoteResources collection to the top level system for the Cascading Profile (e.g., Storage Virtualizer).
CIM_HostedService (8.2.1.2.9.8)	This associates the AllocationService to the system in the cascading profile that hosts the service.
CIM_MemberOfCollection (8.2.1.2.9.10)	(Remote Resources) This supports collecting leaf resources. This is optional when used to support the RemoteResources collection (the RemoteResources collection is optional).
CIM_Namespace (8.2.1.2.9.11)	(Leaf) There would be one for every namespace supported.
CIM_NamespaceInManager (8.2.1.2.9.12)	(Leaf) This associates the namespace to the ObjectManager
CIM_ObjectManager (8.2.1.2.9.13)	(Leaf) This is the Object Manager service of the CIM Server.
CIM_RegisteredProfile (8.2.1.2.9.15)	(Leaf) A registered profile that is supported by the CIM Server
CIM_RemoteServiceAccessPoint (8.2.1.2.9.16)	CIM_RemoteServiceAccessPoint represents the management interface to a leaf system.
CIM_SAPAvailableForElement (8.2.1.2.9.17)	Represents the association between a RemoteServiceAccessPoint and the leaf System to which it provides access.
SNIA_AllocationService (8.2.1.2.9.20)	Optional: This service provides methods for allocating and deallocating leaf resources.
SNIA_RemoteResources (8.2.1.2.9.22)	This is a SystemSpecificCollection for collecting leaf resources that may be allocated by the system of the Cascading profile (e.g., StorageVolumes assigned to a virtualizer's StoragePool).

8.2.1.2.9.1 CIM_Dependency

CIM_Dependency is an association between a Leaf System and the Cascading System (ComputerSystem). The specific nature of the dependency is determined by associations between resources of the cascading system and resources of the leaf system.

CIM_Dependency is not subclassed from anything.

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

Class Mandatory: true

Table 107: SMI Referenced Properties/Methods for CIM_Dependency (System Dependency)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The Cascading System.
Dependent		CIM_ManagedElement	The Leaf System.

8.2.1.2.9.2 CIM_Dependency

CIM_Dependency is an association between an Object Manager of a Leaf System and the Object Manager of the Cascading System (ComputerSystem). If the Leaf System and the Cascading System are supported by the same Object Manager, then no Dependency would exist.

CIM_Dependency is not subclassed from anything.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 108: SMI Referenced Properties/Methods for CIM_Dependency (Object Manager Dependency)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The Object Manager of the Cascading System.
Dependent		CIM_ManagedElement	The Object Manager of the Leaf System.

8.2.1.2.9.3 CIM_Dependency

CIM_Dependency is an association between RegisteredProfile and the Object Manager that provides the management interface.

CIM_Dependency is not subclassed from anything.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 109: SMI Referenced Properties/Methods for CIM_Dependency (RegisteredProfile OM Dependency)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The Object Manager.
Dependent		CIM_ManagedElement	The RegisteredProfile.

8.2.1.2.9.4 CIM_ElementCapabilities

CIM_ElementCapabilities represents the association between ManagedElements (i.e.,ComputerSystem) and their capabilities (e.g., SNIA_CascadingCapabilities).

CIM_ElementCapabilities is not subclassed from anything.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 110: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element (System or System Subclass)
Capabilities		CIM_Capabilities	The CascadingCapabilities instance associated with the element.

8.2.1.2.9.5 CIM_ElementConformsToProfile

CIM_ElementConformsToProfile is the association between the RegisteredProfile of the leaf profile and the system of the leaf (i.e., leaf ComputerSystem).

CIM_ElementConformsToProfile is not subclassed from anything.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 111: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile (Leaf)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	The RegisteredProfile of the leaf system.
ManagedElement		CIM_ManagedElement	The "top level" System instance of the leaf profile.

8.2.1.2.9.6 CIM_HostedCollection

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 Cascading Subprofile, it is used to associate the Allocated Resources to the top level Computer System of the Cascading Profile.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

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

Class Mandatory: true

Table 112: SMI Referenced Properties/Methods for CIM_HostedCollection (AllocatedResources)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The top level ComputerSystem of the profile
Dependent		CIM_SystemSpecificCollection	The AllocatedResources collection

8.2.1.2.9.7 CIM_HostedCollection

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 Cascading Subprofile, it is used to associate the Remote Resources to the top level Computer System of the Cascading Profile.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 113: SMI Referenced Properties/Methods for CIM_HostedCollection (Remote Resources)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The top level ComputerSystem of the profile
Dependent		CIM_SystemSpecificCollection	The RemoteResources collection

8.2.1.2.9.8 CIM_HostedService

CIM_HostedService is an association between a Service (SNIA_AllocationService) and the System (ComputerSystem) on which the functionality resides.

CIM_HostedService is subclassed from CIM_HostedDependency.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 114: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System.
Dependent		CIM_Service	The AllocationService hosted on the System.

8.2.1.2.9.9 CIM_MemberOfCollection

This use of MemberOfCollection is to collect all resource instances (in the AllocatedResources collection). Each association is created as a result of the Allocate method or as a side effect of a cascading profile specific operation.

Created By : ExternalExtrinsic(s):
Modified By : Static
Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 115: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Allocated Resources)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The AllocatedResources collection
Member		CIM_ManagedElement	An individual resource instance that has been allocated.

8.2.1.2.9.10 CIM_MemberOfCollection

This use of MemberOfCollection is to collect all resource instances (in the RemoteResources collection). Each association (and the RemoteResources collection, itself) is created through external means.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 116: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Remote Resources)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The RemoteResources collection
Member		CIM_ManagedElement	An individual resource instance that is or can be allocated.

8.2.1.2.9.11 CIM_Namespace

(Leaf) There would be one for every namespace supported.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 117: SMI Referenced Properties/Methods for CIM_Namespace (Leaf)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	

Table 117: SMI Referenced Properties/Methods for CIM_Namespace (Leaf)

Property	Flags	Type	Description & Notes
ObjectManagerCreationClass-Name		string	
ObjectManagerName		string	
CreationClassName		string	
Name		string	
ClassType		uint16	
DescriptionOfClassType		string	Mandatory if ClassType is set to Other"
Optional Properties/Methods			
ClassInfo		uint16	Deprecated in the MOF, but required for 1.0 compatibility. Not required if all hosted profiles are new in 1.1
DescriptionOfClassInfo		string	Deprecated in the MOF, but mandatory for 1.0 compatibility. Mandatory if ClassInfo is set to Other

8.2.1.2.9.12 CIM_NamespaceInManager

(Leaf) This associates the namespace to the ObjectManager
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 118: SMI Referenced Properties/Methods for CIM_NamespaceInManager (Leaf)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ObjectManager	The ObjectManager containing a Namespace
Dependent		CIM_Namespace	The Namespace in an ObjectManager

8.2.1.2.9.13 CIM_ObjectManager

(Leaf) This is the Object Manager service of the CIM Server.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 119: SMI Referenced Properties/Methods for CIM_ObjectManager (Leaf)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
ElementName		string	

Table 119: SMI Referenced Properties/Methods for CIM_ObjectManager (Leaf)

Property	Flags	Type	Description & Notes
Description		string	
OperationalStatus		uint16[]	
Started		boolean	
Optional Properties/Methods			
StopService()			

8.2.1.2.9.14 CIM_ObjectManager

(Cascading Profile) This is the Object Manager service of the CIM Server.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 120: SMI Referenced Properties/Methods for CIM_ObjectManager (Cascading Profile)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
ElementName		string	
Description		string	
OperationalStatus		uint16[]	
Started		boolean	
Optional Properties/Methods			
StopService()			

8.2.1.2.9.15 CIM_RegisteredProfile

(Leaf) A registered profile that is supported by the CIM Server
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 121: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	This is a unique value for the profile instance.
RegisteredOrganization		uint16	This is the official name of the organization that created the profile. For SMI-S profiles, this would be SNIA.
RegisteredName		string	This is the name assigned by the organization that created the profile.

Table 121: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
RegisteredVersion		string	This is the version number of the organization that defined the profile.
AdvertiseTypes		uint16[]	Defines the advertisement of this profile. If the property is null then no advertisement is defined. A value of 1 is used to indicate other, and a 3 is used to indicate SLP.
Optional Properties/Methods			
OtherRegisteredOrganization		string	
AdvertiseTypeDescriptions		string[]	This shall not be NULL if Other is identified in AdvertiseType'

8.2.1.2.9.16 CIM_RemoteServiceAccessPoint

CIM_RemoteServiceAccessPoint is an instance that provides access information for accessing the actual leaf profile via a management interface.

CIM_RemoteServiceAccessPoint is not subclassed from CIM_ServiceAccessPoint.

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 122: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the management interface.
SystemName		string	The name of the Computer System hosting the management interface.
CreationClassName		string	The CIM Class name of the management interface.
Name		string	The unique name of the management interface.

8.2.1.2.9.17 CIM_SAPAvailableForElement

Represents the association between a RemoteServiceAccessPoint and the leaf System to which it provides access.
Created By : External
Modified By : Static
Deleted By : External

Class Mandatory: false

Table 123: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The System that is made available through a SAP.
AvailableSAP		CIM_ServiceAccessPoint	The Service Access Point that is available to the leaf system.

8.2.1.2.9.18 CIM_SystemDevice

This association links **LogicalDevice** remote resources to the scoping system. This is used to associate the remote resources with the System that manages them.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 124: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that contains this device.
PartComponent		CIM_LogicalDevice	The logical device that is managed by a computer system.

8.2.1.2.9.19 SNIA_AllocatedResources

An instance of a default SNIA_AllocatedResources defines the set of remote (leaf) resources that are allocated and in use by the Cascading Profile.

SNIA_AllocatedResources is subclassed from CIM_SystemSpecificCollection.

At least one instance of the SNIA_AllocatedResources shall exist for a Profile and shall be hosted by one of the ComputerSystems of that Profile.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 125: SMI Referenced Properties/Methods for SNIA_AllocatedResources

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	A user-friendly name for the AllocatedResources collection (e.g., AllocatedVolumes).

Table 125: SMI Referenced Properties/Methods for SNIA_AllocatedResources

Property	Flags	Type	Description & Notes
ElementType		uint16	The type of remote resources collected by the AllocatedResources collection. For this version of SMI-S, the only value supported is "3" (StorageVol-ume).

8.2.1.2.9.20 SNIA_AllocationService

The SNIA_AllocationService class provides methods for allocating and deallocating remote resources for use in the Cascading Profile.

The SNIA_AllocationService class is subclassed from CIM_Service.

There may be an instance of the SNIA_AllocationService if Allocation or Deallocation are supported.

The methods that are supported can be determined from the SynchronousMethodsSupported and AsynchronousMethodsSupported properties of the SNIA_CascadingCapabilities.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 126: SMI Referenced Properties/Methods for SNIA_AllocationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
Optional Properties/Methods			
Allocate()			Support for this method is optional. This method allocates remote (leaf) resources to the AllocatedResources collection.
Deallocate()			Support for this method is optional. This method is used to remove remote (leaf) resources from the Allocate-dResources collection.

8.2.1.2.9.21 SNIA_CascadingCapabilities

An instance of the SNIA_CascadingCapabilities class defines the specific support provided with the implementation of the Cascading Profile.

There would be zero or one instance of this class in a profile. There would be none if the profile did not support the Cascading Subprofile. There would be exactly one instance if the profile did support the Cascading Subprofile.

SNIA_CascadingCapabilities class is subclassed from CIM_Capabilities.

Created By : Static

Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 127: SMI Referenced Properties/Methods for SNIA_CascadingCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
FeaturesSupported		uint16[]	ValueMap { "2", "3", "4", "5" }, Values {"Ownership", "Leaf Credentials", "OM Dependencies", "Allocation Service"}
SupportedElementTypes		uint16[]	For this version of SMI-S, only the value "3" (StorageVolume) is supported. ValueMap { "2", "3", "4", "5", "6", "7", "8" }, Values {"Any Type", "StorageVolume", "StorageExtent", "StoragePool", "ComputerSystem", "LogicalDisk", "FileShare"}
SupportedSynchronousActions		uint16[]	ValueMap { "2", "3" }, Values {"Allocation", "Deallocation"}
SupportedAsynchronousActions		uint16[]	ValueMap { "2", "3" }, Values {"Allocation", "Deallocation"}

8.2.1.2.9.22 SNIA_RemoteResources

An instance of a default SNIA_RemoteResources defines the set of remote (leaf) resources that are available to be used by the Cascading profile.

SNIA_RemoteResources is subclassed from CIM_SystemSpecificCollection.

One instance of the SNIA_RemoteResources would exist for each Element type for a profile and shall be hosted by one of the ComputerSystems of that profile.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 128: SMI Referenced Properties/Methods for SNIA_RemoteResources

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	A user-friendly name for the RemoteResources collection (e.g., RemoteVolumes).

Table 128: SMI Referenced Properties/Methods for SNIA_RemoteResources

Property	Flags	Type	Description & Notes
ElementType		uint16	The type of remote resources collected by the RemoteResources collection. For this version of SMI-S, the only value supported is "3" (StorageVolume).

8.2.1.2.10 Related Standards

Table 129: Related Standards for Cascading

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

DEPRECATED

8.2.1.3 Cluster Subprofile (DEPRECATED)

The functionality of the Cluster Subprofile has been subsumed by 8.2.1.9, "Multiple Computer System Subprofile".

The Cluster Subprofile is defined in *IS24775-2006, Storage Management (*SMI-S 1.0). Any instrumentation that complies to the Multiple Computer System Subprofile defined in this specification may also claim compliance to that version of the Cluster Subprofile and may register as both an SMI-S 1.1 (this version) Multiple Computer System Subprofile and SMI-S 1.0 Cluster Subprofile as defined in IS24775-2006, *Storage Management*.

DEPRECATED

8.2.1.4 Device Credentials Subprofile

8.2.1.4.1 Description

Many devices require a shared secret to be provided to access them. This shared secret is different that the credentials used by the SMI-S Client for authentication with the CIM Server. This Subprofile is used to change this device shared secrets.

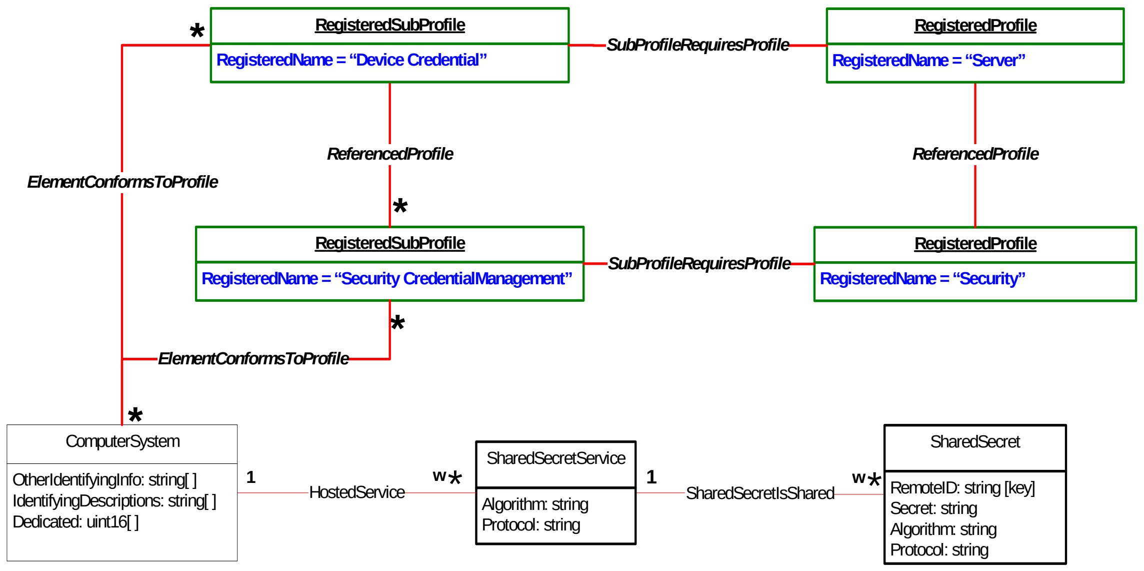
The SMI-S Client shall not be provided with the password, only the principle. The SMI-S Client can use the principle to change the shared secret appropriately.

The device credentials can be exposed throughout the CIM model such that a CIM Client may manipulate them. The credentials are modeled as shared secrets.

Instance Diagram

Figure 30: "Device Credentials Subprofile Model" provides a sample instance diagram.

Figure 30: Device Credentials Subprofile Model



8.2.1.4.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.4.3 Cascading Considerations

Not defined in this standard.

8.2.1.4.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.1.4.5 Extrinsic Methods of this Profile

Not defined in this standard.

8.2.1.4.6 Client Considerations and Recipes

None.

8.2.1.4.7 Registered Name and Version

Device Credentials version 1.1.0

8.2.1.4.8 CIM Server Requirements

Table 130: CIM Server Requirements for Device Credentials

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.4.9 CIM Elements

Table 131: CIM Elements for Device Credentials

Element Name	Description
Mandatory Classes	
CIM_ElementConformsToProfile (8.2.1.4.9.1)	
CIM_HostedService (8.2.1.4.9.2)	
CIM_RegisteredSubProfile (8.2.1.4.9.4)	
CIM_SharedSecret (8.2.1.4.9.5)	
CIM_SharedSecretIsShared (8.2.1.4.9.6)	
CIM_SharedSecretService (8.2.1.4.9.7)	
CIM_SubProfileRequiresProfile (8.2.1.4.9.8)	
Optional Classes	
CIM_RegisteredProfile (8.2.1.4.9.3)	

8.2.1.4.9.1 CIM_ElementConformsToProfile

Class Mandatory: true

Table 132: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.1.4.9.2 CIM_HostedService

Class Mandatory: true

Table 133: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_Service	

8.2.1.4.9.3 CIM_RegisteredProfile

Class Mandatory: false

Table 134: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicates SNIA
RegisteredName	C	string	Parent subprofile

8.2.1.4.9.4 CIM_RegisteredSubProfile

Class Mandatory: true

Table 135: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicates SNIA
RegisteredName	C	string	This subprofile

8.2.1.4.9.5 CIM_SharedSecret

Class Mandatory: true

Table 136: SMI Referenced Properties/Methods for CIM_SharedSecret

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
ServiceCreationClassName		string	
ServiceName		string	
RemoteID		string	
Secret		string	

8.2.1.4.9.6 CIM_SharedSecretIsShared

Class Mandatory: true

Table 137: SMI Referenced Properties/Methods for CIM_SharedSecretIsShared

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SharedSecretService	
Dependent		CIM_SharedSecret	

8.2.1.4.9.7 CIM_SharedSecretService

Class Mandatory: true

Table 138: SMI Referenced Properties/Methods for CIM_SharedSecretService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ElementName		string	

8.2.1.4.9.8 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 139: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.1.4.10 Related Standards

Table 140: Related Standards for Device Credentials

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

DEPRECATED

8.2.1.5 Extra Capacity Set Subprofile (DEPRECATED)

The functionality of the Extra Capacity Set Subprofile has been replaced by 8.2.1.9, "Multiple Computer System Subprofile".

The Extra Capacity Set Subprofile is defined in Annex B of IS24775-2006, *Storage Management*, also known as SMI-S 1.0.

DEPRECATED

8.2.1.6 Health Package

8.2.1.6.1 Description

Failures and abnormal occurrences are a common and expected part of monitoring, controlling, and configuring devices and applications. A SMI-S client needs to be prepared at all times to trap unexpected situations and take appropriate action. This package defines the general mechanisms used in the expression of health in SMI-S. This package does not define the particular way a particular profile, subprofile, or package reports health.

This package builds on the Health and Fault Management (HFM) Clause. In particular, this package defines the basis of all the sections that currently and will exist in this specification or future versions of same.

Error Reporting Mechanism

Error are reports for many reasons. Not all the reasons are directly related to the operation being imposed on the implementation by the client. It is therefore necessary for the client to be able to distinguish between errors that are associated to problems in the formation and invocation of a method, extrinsic or intrinsic, or are related to other conditions.

The client application may need to reform the method call itself, by fixing parameters for example, or the client may need to stop what its attempting. At a basic level, the client needs to know that this operation will succeed at all, given the prevailing conditions on the managed element. A client may also need to notify the end-user of the situation that is preventing the client from fulfilling its function. A HFM application may need to investigate the failure and develop a prognosis.

The types of errors are categorized in the three following types.

a) Errors associated to the method call

b) Errors caused by adverse prevailing conditions in the managed element

c) Errors causes by adverse prevailing conditions in the WBEM Server or related, infrastructural components

Obviously, the method called may not exist. There may be a spelling mistake for the method name. One or more of the parameters may be incorrectly formed, expressed, or otherwise invalid. The first type of error, type a, is designed to inform the client that the operation attempted is still valid, but that the request was faulty. The intent of such an error is to tell the client what is wrong with the method call and allow the method to be invoked again.

On the other hand, the device or application may be in some failure condition which prevents it from honoring this particular or several method calls. This type of error, type b, tells the client that the it is unlikely that the method being attempted will be honored. Specifically, the method execution is blocked by the prevailing condition being described in the error itself. Given the presence of both type a and type b error situations, the implementation should report the type b error. In this case, it does not matter how many fixes are made to the method call, the method call will fail anyway.

The WBEM Service is a separate architectural element from the managed element itself. It can fail, even though the methods and the managed element itself are without error. For example, the WBEM Server may allow only a limited number of concurrent connection or request and reject all others. The server may be shutting down or starting up and thus be unable to process any requests at the time. Unlike type b errors, type c errors are usually transient in nature. Since a failure in the WBEM Server or its components constitutes a communications failure, the reporting of type c errors shall take precedence over all other existing error type conditions.

The WBEM Server returns a error response or a results response to the request, which contains the operation previous mentioned. Errors in WBEM may be reported through two ways. The status code itself provides basic failure information. The number of status codes is very limited. Also on conveyed on the error response, is a Error instance. The Error provides vastly most information than the status code and, as such, is a superior mechanism for reporting errors.

The CIM Error provides attributes to express the categorization and severity of the error. More importantly, the CIM Error and AlertIndication, to be discussed later, contain the exact expression of the nature of the error and additional parameters to that error.

Event Reporting Mechanism

It is not sufficient to simply report the adverse conditions of the device or application through the error reporting mechanism. Many of the adverse conditions that would be reported to a client application attempting control or configuration operations are also of interest to client applications monitoring the very same device or application.

The CIM Event model provides a special class for reporting event conditions, AlertIndication. The AlertIndication is used to report a device or application conditions that may also be represented in one or more other instances. When the implementation detects the presence of a supported condition, it generates an AlertIndication to those listening clients.

It is recommended that the type b and type c errors reported above are also be reported through AlertIndications.

Standard Events

The expression of Error or an Alertindication is not entirely meaningful to the SMI-S client without the standardization. A client can use these classes to determine the category, severity, and some other characteristics of the event, but the client can not determine the exact nature of the event without this standardization.

Standard events are registered and this registry is maintained by some organization or company, like SNIA.

Primary event identification and characterization properties:

- OwningEntity
  This property defines the registration entity for the event. The entities that are in scope for SMI-S are “DMTF” and “SNIA”. If the OwningEntity is neither of these, then this specification provides no meaning for this event.

- MessageID
  This property defines an event identifier that is unique for the OwningEntity. The combination of the OwningEntity and MessageID defines the entry in the registry.

- Message
  This property contains the message that can be forwarded to the end-user. The message is built from using the static, MessageFormatString, and dynamic, MessageArguments, components. This text may be localized. This text is not intended for programmatic processing

- MessageArguments
  This property defines the variable content for the message. The client would programmatically process the arguments to get further details on the nature of the event. For example, the message argument can tell the client which method parameter has a problem and what the problem is.

- MessageFormatString
  This property defines the static component of the message. This property is not included in the event instance itself and is only present in the event registry.

Reporting Health

Many devices or applications can attempt to fix themselves upon encountering some adverse condition. The set of components which the device or application can attempt to fix is called the Fault Region. The set may include part or all of other devices or applications. Having the Fault Regions declared helps a HFM application, acting as a doctor, to do no harm by attempting to interfere and thereby adversely effect the corrective action being attempted.

When components fail or become degraded, they can cause other components to fail or become degraded. For an HFM application to report or attempt to diagnose the problem, the device or application should express what the cause and effect relationships are that define the extent of the components affected by the failure or degradation. The RelatedElementCausingError class provides just such a mechanism.

The cause and effect relationships identified by the RelatedElementCausingError association may be a chain of cause and effect relationships with many levels. Given that devices or applications are sometimes subject to several levels of decomposition, each level of may have its own set of these associations that represent the ranking of cause and effect relationships and their effect on the parent component on the given level.

Computer System Operational Status

For most profiles, the ComputerSystem class is used to define the top or head of the object hierarchy. A profile may allow for partitioning or clustering by having more than one ComputerSystem, but one ComputerSystem often represents the device or application representation. In this role, it is important the summary of the health of the device or application is declared in the ComputerSystem instance.

Table 141: OperationalStatus Details

Primary Operational Status	Subsidiary Operational Status	Description
2 “OK”		The system has a good status
2 “OK”	4 “Stressed”	The system is stressed, for example the temperature is over limit or there is too much IO in progress
2 “OK”	5 “Predictive Failure”	The system will probably fail sometime soon
3 “Degraded”		The system is operational but not at 100% redundancy. A component has suffered a failure or something is running slow
6 “Error”		An error has occurred causing the system to stop. This error may be recoverable with operator intervention.
6 “Error”	7 “Non-recoverable error”	A severe error has occurred. Operator intervention is unlikely to fix it
6 “Error”	16 “Supporting entity in error”	A modeled element has failed
12 “No contact”		The provider knows about the array but has not talked to it since last reboot
13 “Lost communication”		The provider used to be able to communicate with the array, but has now lost contact.
8 “Starting”		The system is starting up
9 “Stopping”		The system is shutting down.
10 “Stopped”		The data path is OK but shut down, the management channel is still working.

OperationalStatus is an array. The primary and subsidiary statuses are both OperationalStatus property, and are summarized in Table 141. If the subsidiary operational status is present in the array, it is intended to provide additional clarification to the primary operational status. The implementation shall report one of the above combinations of statuses. It may also report additional statues beyond the ones defined above.

The operational status combinations listed above that include descriptions about “provider” (i.e., the CIM Provider), are only valid in those cases where the implementation of SMI-S employs a proxy provider.

The above operational statuses shall not be used to report the status of the WBEM Server itself.

EXPERIMENTAL

Event Reporting

The implementation may report Event or AlertIndication instances. The profile, subprofile, or package that includes this package defines whether or not these events are supported and when the events are produced.

If Event or AlertIndication is implemented, then the implementation shall also support the common messages through both Errors and AlertIndications. This means that the implementation shall produce the common event listed in the registry whenever the condition, also described in the registry, is present.

It is mandatory to report error conditions through both AlertIndication or Lifecycle indication and Error in those cases where Error is returned when the method call failed for reasons other than the method call itself. For example, if the device is over heated, then a method call can fail because of this condition. It is expected that the device will report an over heat AlertIndication to listening clients as well.

Fault Region

If the device or application is itself attempting to rectify an adverse condition reported through a standard error, then the implementation shall report what corrective action, if any, it is taking. This is necessary to prevent a HFM application from also trying to rectify the very same condition. An HFM application should avoid a interfering with ongoing corrective action taken by the device or application itself.

The corrective action may be a process, like hardware diagnostics or volume rebuild. In which case, the above requirement is fulfilled by expressing the instances representing the process.

The corrective action may be a state change, like reboot. In which case, the above requirement is fulfilled by expressing the state change in some CIM Instances.

In all cases, the profile, subprofile, or package that includes this package defines the standard events included and the associated, possible corrective actions taken in response to these events.

EXPERIMENTAL

RelatedElementCausingError

This package provides a mechanism by which the effect of a component failure on other components can be reported. The RelatedElementCausingError association defines what components are causing a particular component to fail or become degraded.

Some effects are more central to a given failure or degradation than others. This association provides a mechanism for ranking related effects, easing the identification of primary and secondary causes. The implementing shall provide the EffectCorrelation property, but it recommended that the implementation also provide the FailureRelationshipInitiated and Ranking properties

If there are these cause and effect relationships, the RelatedElementCausingError association should be implemented to report the causes of the failure or degradation.

HealthState

The HealthState property in LogicalDevice defines the state for a particular component. The OperationalStatus defines operational status. For example, a disk or port may be taken off-line for service. The component’s health may still be OK or not OK. The two properties, when used in combination, disambiguate the health of the component. For example, a OperationStatus of 10 “Stopped” and a HealthState of 30 “Major Failure” means that the component is off-line and has failed. While a OperationalStatus of 10 “Stopped” and a HealthState of 5 “OK” for the very same component means that although the component is off-line, the component is still in good working order.

The HealthState of a component should not represent the health of any other component as well by way of a summary or aggregate health state. However, if the component is itself relies on other components for its health, because the component itself is an aggregate of components, then the HealthState may represent a summary HealthState by side-effect.

HealthState is a mandatory for all system device logical devices that are defined by the profile or subprofile that includes this package. It is recommended that HealthState is something other than 0 “Unknown”. However, a component may report “Unknown” after it has reported one of the other HealthStates. When HealthState changes from 5 “OK”, it is mandatory that a LogicalDevice report some other HealthState (e.g., 30 “Major Failure”) before reporting 0 “Unknown”. Such a requirement is necessary, so that the client can notice the adverse state change via polling or indication before the component is no longer responding.

8.2.1.6.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.6.3 Cascading Considerations

Not defined in this standard.

8.2.1.6.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.1.6.5 Methods of this Profile

Not defined in this standard.

8.2.1.6.6 Client Considerations and Recipes

Not defined in this standard.

8.2.1.6.7 Registered Name and Version

Health version 1.1.0

8.2.1.6.8 CIM Server Requirements

Table 142: CIM Server Requirements for Health

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.6.9 CIM Elements

Table 143: CIM Elements for Health

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.1.6.9.1)	
CIM_LogicalDevice (8.2.1.6.9.2)	
Optional Classes	
CIM_RelatedElementCausingError (8.2.1.6.9.3)	
Mandatory Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	CQL - Operational Status change of the device and application.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDevice AND SourceInstance.CIM_LogicalDevice::HealthState <> PreviousInstance.CIM_LogicalDevice::HealthState	CQL - Health State change of the logical component.

8.2.1.6.9.1 CIM_ComputerSystem

Class Mandatory: true

Table 144: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	
OperationalStatus		uint16[]	Overall status of the Host

8.2.1.6.9.2 CIM_LogicalDevice

Class Mandatory: true

Table 145: SMI Referenced Properties/Methods for CIM_LogicalDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
HealthState		uint16	Reports the health of the component beyond the operational status.

8.2.1.6.9.3 CIM_RelatedElementCausingError

Class Mandatory: false

Table 146: SMI Referenced Properties/Methods for CIM_RelatedElementCausingError

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ManagedElement	
Antecedent		CIM_ManagedElement	
EffectCorrelation		uint16	Describes the general nature of the cause and effect correlation.
Optional Properties/Methods			
FailureRelationshipInitiated		datetime	Reports the date and time when this cause and effect was created. The population of this property is recommended.
Ranking		uint16	Describes the order of effect from 1, the highest effect, on. If there is only one of these associations between two elements, the ranking shall be 1. Once more associations are added, then it Recommend that the implementation assist the client by stating which of the cause and effect relationship should be reviewed and addressed first. This property assists a client in accomplishing a triage of known problems.

8.2.1.6.10 Related Standards

Table 147: Related Standards for Health

Specification	Revision	Organization
CIM Infrastructure	2.3	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.1.7 Job Control Subprofile

8.2.1.7.1 Description

In some profiles, some or all of the methods described may take some time to execute (longer than an HTTP time-out). In this case, a mechanism is needed to handle asynchronous execution of the method as a 'Job'.

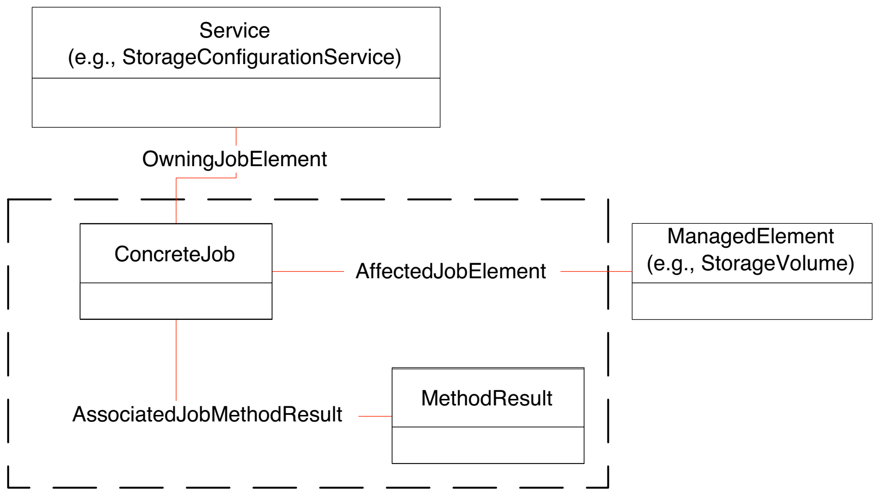
This subprofile defines the constructs and behavior for job control for SNIA profiles that make use of the subprofile.

Note: The subprofile describes a specific use of the constructs and properties involved. The actual CIM capability may be more, but this specification clearly states what clients may depend on in SNIA profiles that implement the Job Control subprofile.

Instance Diagram

A normal instance diagram is provided in Figure 31: "Job Control Subprofile Model".

Figure 31: Job Control Subprofile Model



When the Job Control Subprofile is implemented and a client executes a method that executes asynchronously, a reference to an instance of ConcreteJob is returned and the return value for the method is set to “Method parameters checked - job started”.

The ConcreteJob instance allows the progress of the method to be checked, and instance Indications can be used to subscribe for Job completion.

The associations OwningJobElement and AffectedJobElement are used to indicate the service whose method created the job by side-effect and the element being affected by the job. The job itself may create, modify and/or delete many elements during its execution. The nature of this affect is the creation or deletion of the instances or associations or the modification of instance properties. These elements, albeit regular instances or associations, are said to be *affected* by the job. The elements linked by AffectedJobElement may change through the execution of the job, and in addition, the job may be associated to more than one Input and/or Output elements or other elements affected by side-effect. Input and Output elements are those referenced by method parameters of the same type, input and output parameters respectively.

EXPERIMENTAL

The following set of rules defines the nature of the AffectedJobElement associations for a given job in terms of the references passed as parameters to the service method that spawned the job. Obviously, the distinction of Input element from Output element in the following rules only makes sense if these parameters are not both Input and Output elements.

- If all Elements created by the method exists immediately upon the return from the method, then AffectedJobElement shall reference the Output Element.
- If the Output Element, one or more, does not exist until the job has completed, the AffectedJobElement shall reference the Input Element until the job completes, at which time AffectedJobElement shall then reference the Output Element instead.
- In the event the job fails and the Output Element created during the job and referenced by AffectedJobElement is no longer available, AffectedJobElement shall revert to referencing the Input Element.
- If the method affects elements without referencing elements as Output parameters, then the AffectedJobElement Association shall reference the Input element, one or more.
- If the method only modifies the elements referenced with method parameters, then the AffectedJobElement association references the modified elements. Elements modified by the job shall be reference by this association.
- If the method affects elements but references no elements as either Input or Output parameters or the only Input elements referenced are those of the elements to be deleted, then AffectJobElement associations shall exist to other elements that are affected by the job.
- Other elements whose references are not used in the method invocation, but that are created or modified by side-effect of the job's execution shall be associated to the job via the AffectJobElement association, but may cease to be associated once the job has finished execution.

The lifetime of a completed job instance, and thus the AffectedJobElement association to the appropriate Element is currently implementation dependent. However, the set of AffectedJobElement associations to Input and Output element present when the job finishes execution shall remain until the job is deleted.

<u>MethodResult</u>

Jobs are produced by side effect of the invocation of an extrinsic method. Reporting the resulting Job is the purpose of this subprofile. The MethodResult class is used to report the extrinsic method called and the parameters passed to the method. In this way, third party observers of a CIMOM can tell what the job is and what it is doing. A MethodResult instance contains the LifeCycle indications that have been or would have been produced as the result of the extrinsic method invocation. That is, the instance contains the indications whether or not there were the appropriate indication subscription at the time the indication were produced.

A client may fetch the method lifecycle indication produced when the method was called from the PreCallIndication attribute. This indication, an instance of InstMethodCall, contains the input parameters provided by the client that called the method.

A client may fetch the method lifecycle indication produced once the method execution was completed from the PostCallIndication. This indication contains the input parameters provided by the client that called the method and output parameters returned by the method implementation. Parameters that are both input and output parameters will contain the output parameter provided by the method implementation.

EXPERIMENTAL

OperationalStatus for Jobs

The OperationalStatus property is used to communicate that status of the job that is created. As such, it is critical that implementations are consistent in how this property is set. The values that shall be supported consistently are:

- 2 “OK” - combined with 17 “Completed” to indicate that the job completed with no error.
- 6 “Error” - combined with 17 “Completed” to indicate that the job did not complete normally and that an error occurred.
- 10 “Stopped” implies a clean and orderly stop.
- 17 “Completed” indicates the Job has completed its operation. This value should be combined with either 2 “OK” or 6 “Error, so that a client can tell if the complete operation passed (Completed with OK), and failure (Completed with Error).

JobState for Jobs

The JobState property is used to communicate Job specific states and statuses.

- 2 “New” - Job was created but has not yet started
- 3 “Starting” - Job has started
- 4 “Running” - Job is current executing
- 5 “Suspended” - Job has been suspended. The Job may be suspended for many reasons like it has been usurped by a higher priority or a client has suspended it (not described within this subprofile).
- 6 “Shutting Down” - Job is completing its work, has been terminated, or has been killed. The Job may be cleaning up after only having completed some of its work.
- 7 “Completed” - Job has completed normally, its work has been completed successfully.
- 8 “Terminated” - Job has been terminated
- 9 “Killed” - Job has been aborted. The Job may not cleanup after itself.
- 10 “Exception” - Job failed and is in some abnormal state. The client may fetch the error conditions from the job. See Getting Error Conditions from Jobs () in 8.2.1.7.1, "Description".

Table 148 maps the standard mapping between the OperationalStatus and JobState properties on ConcreteJob. The actual values of the properties are listed in Table 148 with the associated value from

Table 148: OperationalStatus to Job State Mapping

OperationalStatus	JobState	Job is
2 “OK”, 17 “Completed”	7 “Completed”	Completed normally
6 “Error”, 17 “Completed”	10 “Exception”	Completed abnormally
10 “Stopped”	7 “Terminated”	Terminated
6 “Error”	9 “Killed”	Aborted / Killed
2 “OK”	4 “Running”	Executing
15 “Dormant”	2 “New”	Created but not yet executing
2 “OK”, 8 “Starting”	3 “Starting”	Starting up
2 “OK”	5 “Suspended”	Suspended
2 “OK”, 9 “Stopping”	6 “Shutting Down”	Terminated and potentially cleaning up
6 “Error”	6 “Shutting Down”	Killed and is aborting

the property’s ValueMap qualifier.

<u>Determining How Long a Job Remains after Execution</u>

The Job shall report how long it will remain after it has finished executing, fails on its own, is terminated, or is killed. The TimeBeforeRemoval attribute reports a datetime offset.

The TimeBeforeRemoval and DeleteOnCompletion attributes are related. If the DeleteOnCompletion is FALSE, then the Job shall remain until is it explicitly deleted. If the DeleteOnCompletion is TRUE, then the Job shall exist for the length of time specified in the TimeBeforeRemoval attribute. An implementation may not support the setting of the DeleteOnCompletion attribute because it does not support the client modifying the Job instance.

The amount of time specified in the TimeBeforeRemoval should be five or more minutes. This amount of time allows a client to recognize that the Job has failed and retrieve the Error.

8.2.1.7.2 Health and Fault Management

The implementation should report CIM Errors from the ConcreteJob.GetError() method. See 8.2.1.6, "Health Package" for details.

EXPERIMENTAL

The standards messages specific to this profile are listed in Table 149. See 6.5, "Standard Messages" for a description of standard messages and the list all standard messages

Table 149: Standard Message for Job Control Subprofile

Message ID	Message Name
DRM22	Job failed to start
DRM23	Job was halted

EXPERIMENTAL

8.2.1.7.3 Cascading Considerations

Not defined in this standard.

8.2.1.7.4 Support Subprofiles and Packages

Not defined in this standard.

8.2.1.7.5 Methods of the Profile

Job Modification

A Job instance may be modified. The DeleteOnCompletion and TimeBeforeRemoval properties are writable. If the intrinsic ModifyInstance method is supported, then the setting of both attributes shall be supported.

EXPERIMENTAL

Getting Error Conditions from Jobs

```
uint32 GetError(
            [Out, EmbeddedObject] string Error);
```

This method is used to fetch the reason for the job failure. The type of failure being report is when a Job stops executing on its own. That is, the Job was not killed or terminated. An Embedded Object, encoded in a string, shall returned if the method is both supported and the job has failed. The Job shall report the 10 “Exception” status when the Job has failed on its own.

The GetError method should be supported.

The Error string contains a Error instance. See 8.2.1.6, "Health Package" for details on how to process this CIM Instance.

EXPERIMENTAL

Suspending, Killing or Terminating a Job

A Job may be suspended, terminated or killed. Suspending a Job means that the Job will not be executing and be suspended until it is resumed. Terminating a job means to request that the Job stop executing and that the Job clean-up its state prior to completing. Killing a job means to request that the Job abort executing, usually meaning there is little or no clean-up of Job state.

```
uint32 RequestStateChange(
                    [In] RequestedState,
                    [In] TimeoutPeriod);
```

A client may request a state change on the Job.

- RequestedState - The standard states that can requested are “Start”, “Suspend”, “Terminate”, “Kill”, “Service”. A new Job may be started. A suspended Job may be resumed, using the “Started” requested status. A executing Job may be suspended, terminated, or killed. A new or executing Job may be put into the “Service” state. The “Service” state is vendor specific. An implementation can indicate what state transitions are supported by not returning the 4 098 “Invalid State Transition” return code
- TimeoutPeriod - The client the state transition to occur within the specified amount of time. The implementation may support the method but not this parameter.

Return codes:

- 0 “Completed with No Error”

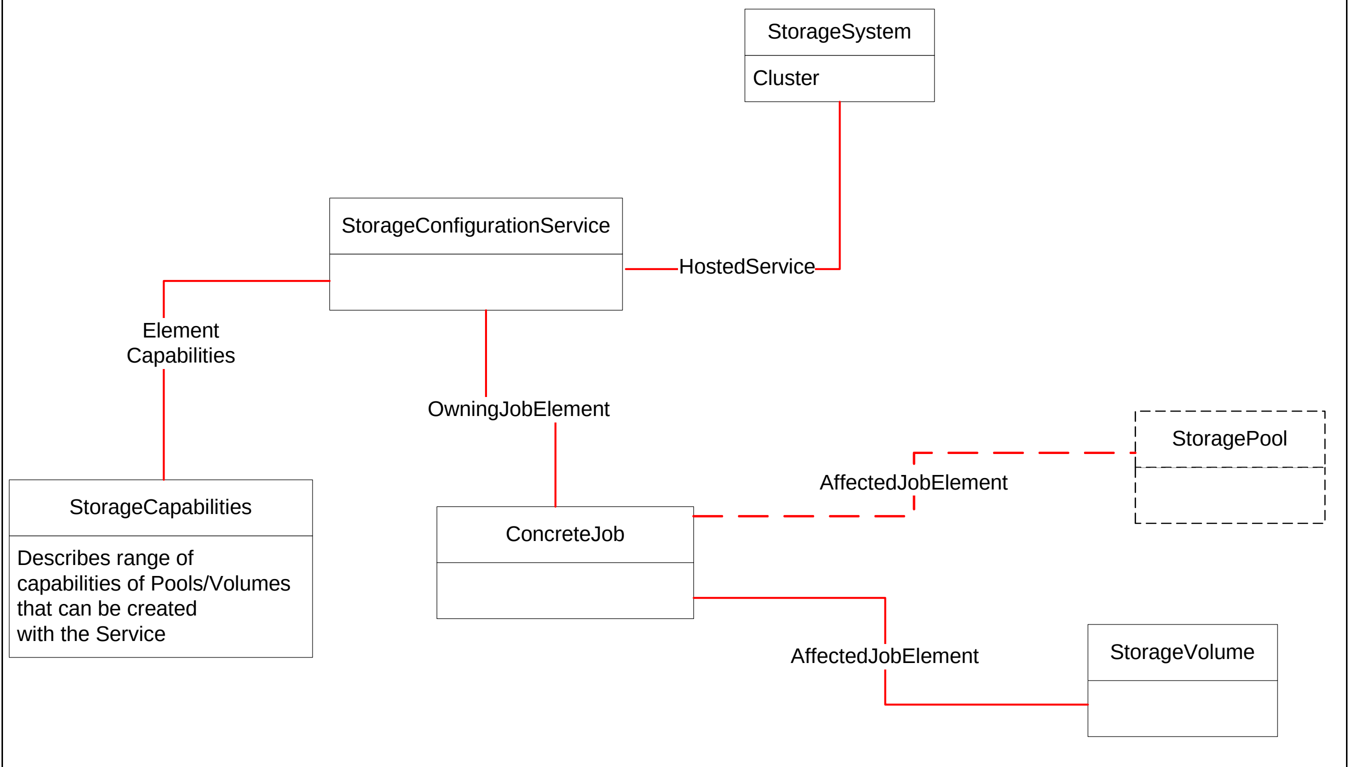
- 1 “Not Supported” - The method is not supported
- 2 “Unknown/UnSpecified Error” - Failure for some vendor specific reason
- 3 “Can not complete within Timeout Period” - The requested amount of time is less than how long the requested state transition takes
- 4 “Failed”
- 5 “Invalid Parameters” - The parameters are incorrect
- 6 “In Use” - Another client has requested a state change that has not completed
- 4 096 “Method Parameters Checked - Transition Started” - The method can return before the state transition completes. This error code tells that calling that this situation has occurred
- 4 097 “Invalid State Transition” - The state change requested is invalid for the current state. 4 098 “Use of Timeout Parameter Not Supported” - This implementation does not support the TimeoutPeriod parameter. A client may pass a NULL for the TimeoutPeriod and try again. There is no mechanism to determine what state changes are supported by a particular implementation. Such a mechanism is planned for a future version of this specification.
- 4 099 “Busy” - A state change is underway in the Job and, as such, the state can not be changed. An implementation may use this return code to indicate the job can not be suspended, killed, or terminated at all or in the current phase of execution

8.2.1.7.6 Client Considerations and Recipes

If the operation will take a while (longer than an HTTP timeout), a handle to a newly minted ConcreteJob is returned. This allows the job to continue in the background. Note a few things:

- The job is associated to the Service via OwningJobElement and is also linked to the object being modified/created via AffectedJobElement. For example, a job to create a StorageVolume may start off pointing to a Pool until the Volume is instantiated at which point the association would change to the StorageVolume.
- These jobs do not have to get instantiated. If the method completes quickly, a null can be returned as a handle, as illustrated in Figure 32: "Storage Configuration".
- It may take some time before the Job starts.
- A Job may be terminated or killed.
- Jobs may be modified.
- Jobs may be restarted.

Figure 32: Storage Configuration



8.2.1.7.7 Registered Name and Version

Job Control version 1.1.0

8.2.1.7.8 CIM Server Requirements

Table 150: CIM Server Requirements for Job Control

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.7.9 CIM Elements

Table 151: CIM Elements for Job Control

Element Name	Description
Mandatory Classes	
CIM_AffectedJobElement (8.2.1.7.9.1)	
CIM_AssociatedJobMethodResult (8.2.1.7.9.2)	
CIM_ConcreteJob (8.2.1.7.9.3)	
CIM_MethodResult (8.2.1.7.9.4)	
CIM_OwningJobElement (8.2.1.7.9.5)	
Mandatory Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.PercentComplete <> PreviousInstance.PercentComplete	Deprecated WQL - Modification of Percentage Complete for a Concrete Job
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.OperationalStatus[*] = 17 AND SourceInstance.OperationalStatus[*] = 2	Deprecated WQL - Modification of Operational Status for a Concrete Job to 'Complete' and 'OK'
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.OperationalStatus[*] = 17 AND SourceInstance.OperationalStatus[*] = 6	Deprecated WQL - Modification of Operational Status for a Concrete Job to 'Complete' and 'Error'
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.CIM_ConcreteJob::PercentComplete <> PreviousInstance.CIM_ConcreteJob::PercentComplete	CQL - Modification of Percentage Complete for a Concrete Job
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND ANY SourceInstance.CIM_ConcreteJob::OperationalStatus[*] = 17 AND ANY SourceInstance.CIM_ConcreteJob::OperationalStatus[*] = 2	CQL - Modification of Operational Status for a Concrete Job to 'Complete' and 'OK'
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND ANY SourceInstance.CIM_ConcreteJob::OperationalStatus[*] = 17 AND ANY SourceInstance.CIM_ConcreteJob::OperationalStatus[*] = 6	CQL - Modification of Operational Status for a Concrete Job to 'Complete' and 'Error'
Optional Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.JobStatus <> PreviousInstance.JobStatus	Deprecated WQL - Modification of Job Status for a Concrete Job. Deprecated
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.CIM_ConcreteJob::JobStatus <> PreviousInstance.CIM_ConcreteJob::JobStatus	CQL - Modification of Job Status for a Concrete Job. Deprecated

Table 151: CIM Elements for Job Control

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.JobState <> PreviousInstance.JobState	Deprecated WQL - Modification of Job State for a Concrete Job.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ConcreteJob AND SourceInstance.CIM_ConcreteJob::JobState <> PreviousInstance.CIM_ConcreteJob::JobState	CQL - Modification of Job State for a Concrete Job.

8.2.1.7.9.1 CIM_AffectedJobElement

Class Mandatory: true

Table 152: SMI Referenced Properties/Methods for CIM_AffectedJobElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AffectedElement		CIM_ManagedElement	The ManagedElement affected by the execution of the Job.
AffectingElement		CIM_Job	The Job that is affecting the ManagedElement.

8.2.1.7.9.2 CIM_AssociatedJobMethodResult

Class Mandatory: true

Table 153: SMI Referenced Properties/Methods for CIM_AssociatedJobMethodResult

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Job		CIM_ConcreteJob	The Job that has parameters.
JobParameters		CIM_MethodResult	The parameters for the method which by side-effect created the Job.

8.2.1.7.9.3 CIM_ConcreteJob

Created By : External
Deleted By : External
Class Mandatory: true

Table 154: SMI Referenced Properties/Methods for CIM_ConcreteJob

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Name		string	The user-friendly name for this instance of Job. In addition, the user-friendly name can be used as a property for a search or query. (Note: Name does not have to be unique within a namespace.)"

Table 154: SMI Referenced Properties/Methods for CIM_ConcreteJob

Property	Flags	Type	Description & Notes
OperationalStatus		uint16[]	Describes whether the Job is running or not.
JobStatus		string	Add additional detail beyond the OperationalStaus about the runtime status of the Job. This property is free-form and vendor-specific
PercentComplete		uint16	The percentage of the job that has completed at the time that this value is requested. Optimally, the percentage should reflect the amount of work accomplished in relation to the amount of work left to be done. 0 percent complete means that the job has not started and 100 percent complete means the job has finished all its work. However, in the degenerate case, 50 percent complete means that the job is running and may remain that way until the job completes.
DeleteOnCompletion		boolean	Indicates whether or not the job should be automatically deleted upon completion. If this property is set to false and the job completes, then the extrinsic method DeleteInstance shall be used to delete the job versus updating this property. Even if the Job is set to delete on completion, the job shall remain for some period of time, see GetError() method.
TimeBeforeRemoval		datetime	The amount of time the job will exist after the execution of the Job if DeleteOnCompletion is set to FALSE. Jobs that complete successfully or fail shall remain for at least this period of time before being removed from the model (CIMOM).
JobState		uint16	Add additional detail beyond the OperationalStaus about the runtime state of the Job.

Table 154: SMI Referenced Properties/Methods for CIM_ConcreteJob

Property	Flags	Type	Description & Notes
GetError()			This method is used to retrieve the error that caused the Job to fail. The Job shall remain in the model long enough to allow client to a) notice that the job was stopped executing and b) to retrieve the error using this method. There are not requirements for how long the job must remain; however, it is suggested that the Job remain for at least five minutes. JobStatus=10 "Exception" tell the client that the job failed and this method can be called to retrieve the reason why embedded in the CIM_Error, see GetError() method.
Optional Properties/Methods			
ElapsedTime		datetime	The time interval that the Job has been executing or the total execution time if the Job is complete.
ErrorCode		uint16	A vendor specific error code. This is set to zero if the job completed without error.
ErrorDescription		string	A free form string containing the vendor error description.
RequestStateChange()			This method changes the state of the job. The client may suspend, terminate, or shutdown the job. To terminate a job means to request a clean shutdown of the job, have it finish some portion of it's work and terminate or to roll back the changes done by the job to date. The implement can make the choice which behavior. To kill a job means to abort the job, perhaps leaving some element of the work partially done and in an unknown state.

8.2.1.7.9.4 CIM_MethodResult

Class Mandatory: true

Table 155: SMI Referenced Properties/Methods for CIM_MethodResult

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
PrecallIndication		string	Contains a copy of the CIM_InstMethodCall produced when the configuration or control change method was called. This EmbeddedInstance contains the configuration or control change extrinsic method name (MethodName) and parameters (MethodParameters).
PostCallIndication		string	Contains a copy of the CIM_InstMethodCall produced when the configuration or control change method has completed execution and control was returned to the client. This EmbeddedInstance contains the configuration or control change extrinsic method name (MethodName) and parameters (MethodParameters).

8.2.1.7.9.5 CIM_OwningJobElement

Class Mandatory: true

Table 156: SMI Referenced Properties/Methods for CIM_OwningJobElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
OwningElement		CIM_ManagedElement	The ManagedElement responsible for the creation of the Job. (e.g., StorageConfigurationService)
OwnedElement		CIM_Job	The Job created by the ManagedElement.

8.2.1.7.10 Related Standards

Table 157: Related Standards for Job Control

Specification	Revision	Organization
CIM Infrastructure	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.1.8 Location Subprofile

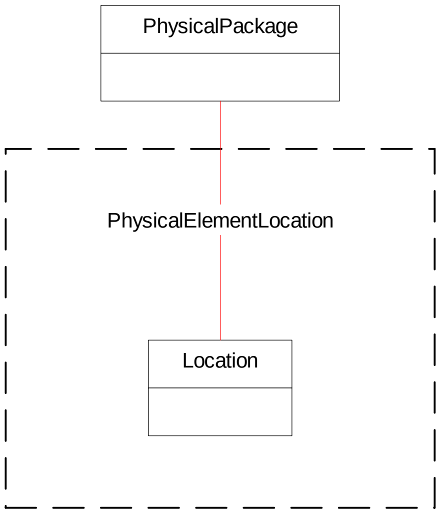
8.2.1.8.1 Description

Associated with product information, a PhysicalPackage may also have a location. This is indicated using an instance of a Location class and the PhysicalElementLocation association.

<u>Instance Diagram</u>

Figure 33: "Location Instance" illustrates a typical instance diagram.

Figure 33: Location Instance



8.2.1.8.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.8.3 Cascading Considerations

Not defined in this standard.

8.2.1.8.4 Supported Subprofiles and Packages

None.

8.2.1.8.5 Methods of the Profile

None.

8.2.1.8.6 Client Considerations and Recipes

None

8.2.1.8.7 Registered Name and Version

Location version 1.1.0

8.2.1.8.8 CIM Server Requirements

Table 158: CIM Server Requirements for Location

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.8.9 CIM Elements

Table 159: CIM Elements for Location

Element Name	Description
Mandatory Classes	
CIM_Location (8.2.1.8.9.1)	
CIM_PhysicalElementLocation (8.2.1.8.9.2)	Associates the location to product

8.2.1.8.9.1 CIM_Location

Class Mandatory: true

Table 160: SMI Referenced Properties/Methods for CIM_Location

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	A free-form string defining a label for the Location.
PhysicalPosition		string	A free-form string indicating the placement of a PhysicalElement.
Optional Properties/Methods			
ElementName		string	User-friendly name.
Address		string	A free-form string indicating a street, building or other type of address for the PhysicalElementsLocation.'

8.2.1.8.9.2 CIM_PhysicalElementLocation

Associates the location to product
Class Mandatory: true

Table 161: SMI Referenced Properties/Methods for CIM_PhysicalElementLocation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Element		CIM_PhysicalElement	The PhysicalElement whose Location is specified.
PhysicalLocation		CIM_Location	The PhysicalElementsLocation.'

8.2.1.8.10 Related Standards

Table 162: Related Standards for Location

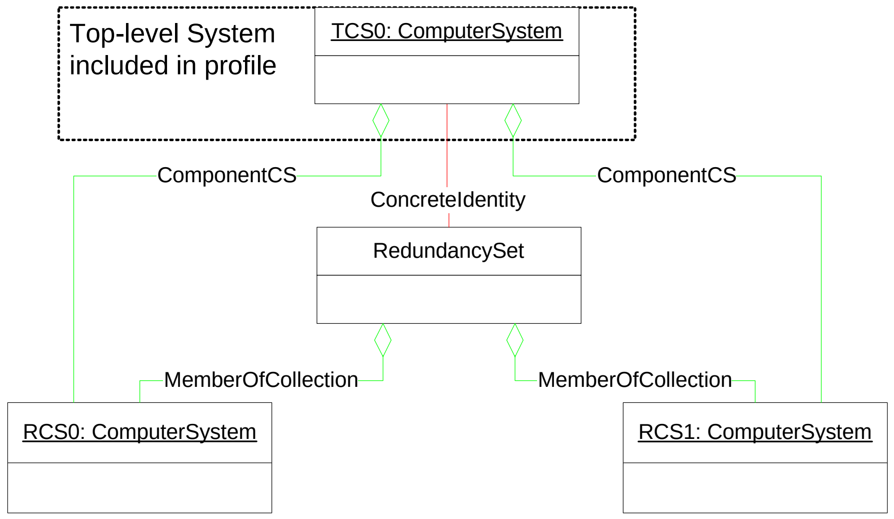
Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.1.9 Multiple Computer System Subprofile

8.2.1.9.1 Description

The Multiple Computer System Subprofile models multiple systems that cooperate to present a “virtual” computer system with additional capabilities or redundancy. This virtual aggregate system is sometimes referred to as a cluster. and is illustrated in Figure 34:, "Two Redundant Systems Instance Diagram".,

Figure 34: Two Redundant Systems Instance Diagram



The general pattern for the redundancy aspect of Multiple Systems uses an instance of RedundancySet to aggregate multiple “real” ComputerSystem instances (labeled RCS0 and RCS1 in the diagram). Another ComputerSystem instance (TCS0) is associated to the RedundancySet instance using a ConcreteIdentity association and is associated to the real ComputerSystems using ComponentCS.

Top Level System

The top (“virtual”) system in this diagram (labeled TCS0) is referred to as the Top Level System. Note that for single-system configurations, the top-level system is the only system. Top-level systems have characteristics different from the underlying ComputerSystem instances.

The Top Level System is associated to the registered profile described in 8.2.4.1, "Server Profile". Other elements such as LogicalDevices (ports, volumes), ServiceAccessPoints, and Services are associated to the top-level system if these elements are supported by multiple underlying systems (for example, the underlying systems provide failover and/or load balancing). Alternatively, elements can be associated to an underlying system if that system is a single point of failure. For example, a RAID array may associate StorageVolume instances to a top-level system since these are available when one underlying system (RAID controller) fails, all the port elements are associated to one underlying system because the ports become unavailable when this system fails.

The Dedicated property is required for top-level systems. Each profile defines the values that are appropriate for Dedicated.

Non-Top-Level Systems

Each ComputerSystem instance shall have a unique Name property. For non-top-level systems, Name may be vendor-unique; in which case, NameFormat shall be set to “Other”.

ComputerSystem.Dedicated should not be used in non-top-level systems.

Non-top-level systems shall not be associated to registered profiles or subprofiles.

Each non-top-level ComputerSystem shall be associated to the top-level system using ComponentCS. Note that non-top-level systems may not be members of a RedundancySet. For example, a top-level system may be associated to a RedundancySet with two systems as described in Figure 34: "Two Redundant Systems Instance Diagram" and also associated via ComponentCS to another Computer (not a member of a RedundancySet) representing a service processor.

Types of RedundancySets

The TypeOfSet property of RedundancySet is a list describing the types of redundancy. its values are summarized in Table 163.

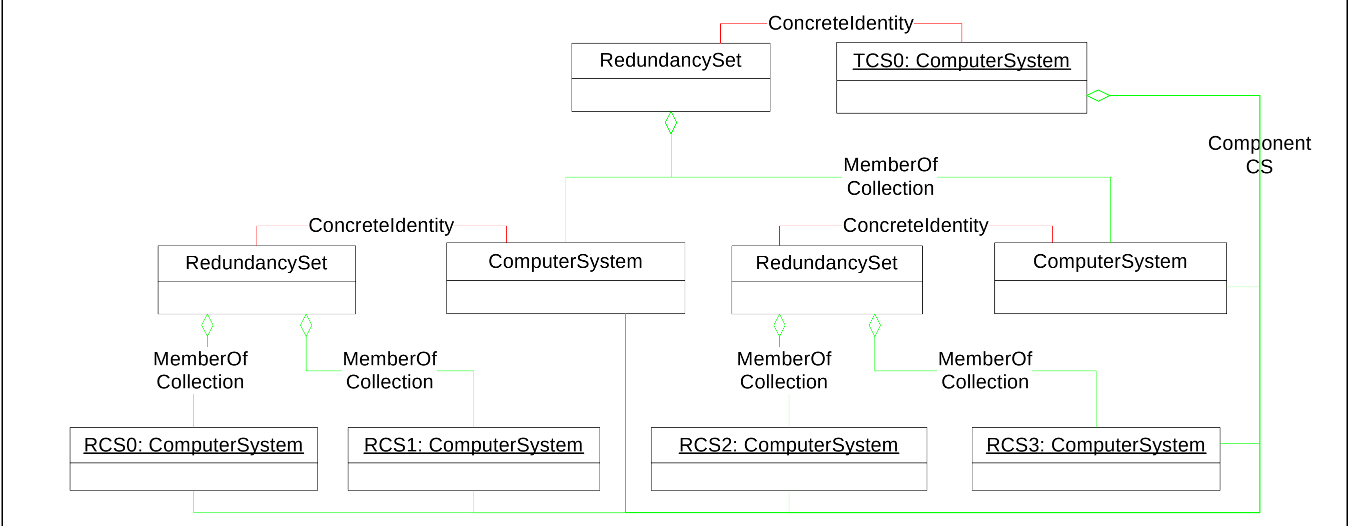
Table 163: Redundancy Type

Redundancy Type	Description
N+1	All ComputerSystems are active, are unaware and function independent of one another. However, there exists at least one extra ComputerSystem to achieve functionality.
Load Balanced	All computer systems are active. However, their functionality is not independent of each other. Their functioning is determined by some sort of load balancing algorithm (implemented in hardware and/or software). 'Sparing' is implied (i.e., each computer system can be a spare for the other(s).
Sparing	All computer systems are active and are aware of each other. However, their functionality is independent until failover. Each computer system can be a spare for the other(s).
Limited Sparing	All members are active, and they may or may not be aware of each and they are not spares for each other. Instead, their redundancy is indicated by the IsSpare relationship.
Other/Unspecified	The relationship between the computer systems is not specified.

Multiple Tiers of Systems

The diagram above describes two tiers of systems; the real systems (labeled RCS0 and RCS1) in the lower tier are aggregated into a top-level system (TCS0) in the upper tier. There may be more than two tiers, as depicted in Figure 35: "Multiple Redundancy Tier Instance Diagram".

Figure 35: Multiple Redundancy Tier Instance Diagram



The systems in the bottom tier (RCS0-RCS3) represent "real" systems.

RedundancySet.TypeOfSet can be used as part of multiple tier configurations to describe different types of redundancy at different tiers. For example, a virtualization system has four controllers that operate in pairwise redundancy. This could be modeled using the model in the diagram above and setting TypeOfSet in the top RedundancySet to “N+1” and setting TypeOfSet to “LoadBalancing” in the lower two RedundacySets.

Associations between ComputerSystems and other Logical Elements

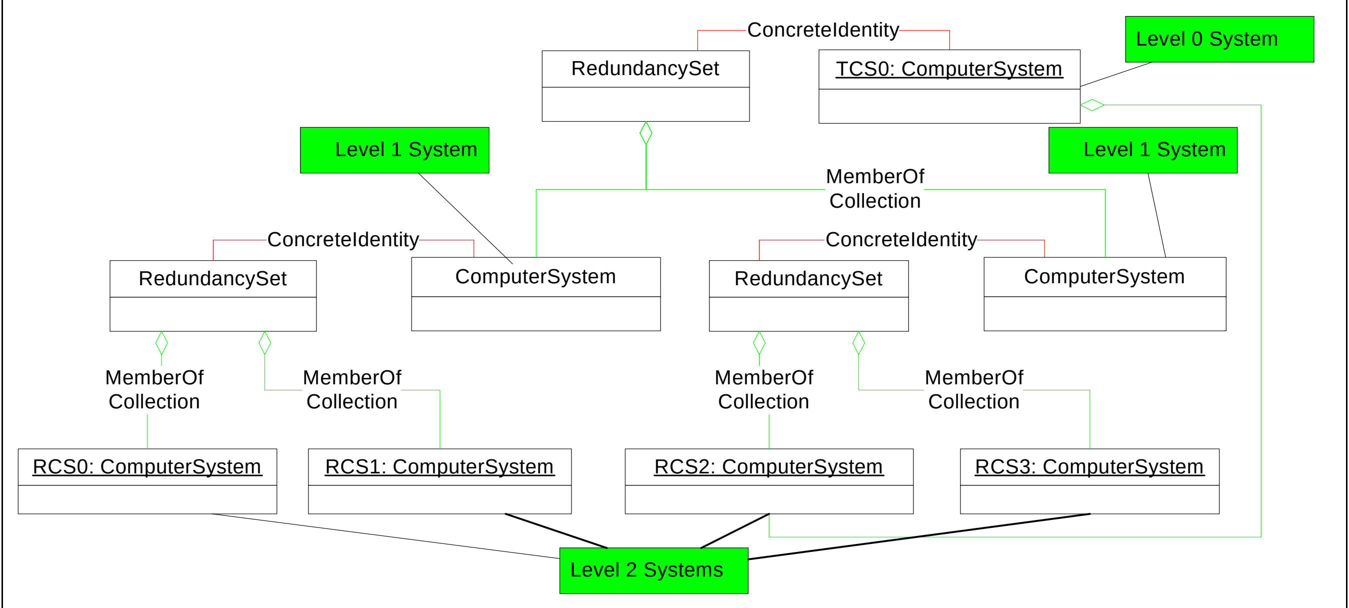
SystemDevice associates device (subclasses of LogicalDevice such as LogicalPort or StorageVolume) and ComputerSystem instances. The cardinality of SystemDevice is one-to-many; a LogicalDevice may be associated with one and only one ComputerSystem. If the device availability is equivalent to that of the top-level system, it shall be associated to the top-level system via SystemDevice. If the device may become unavailable while the system as a whole remains available, the device shall be associated to a non-top-level system that has availability equivalent to the device. This system could be a real system or a system in an intermediate tier (representing some redundancy less than full redundancy).

This same approach shall be used for all other logical CIM elements with associations to systems. For example, HostedService and HostedAccessPoint shall associate elements (services, access points, and protocol endpoints) to the ComputerSystem with availability to the element.

Based on the arrangement of systems in Figure 35, associations from systems to service and capabilities classes shall not be lower than associations to other classes. For the purpose of formally stating this rule, each ComputerSystem is assigned a level number. The profile's top-level ComputerSystem has level number 0. The ComputerSystem instances that are members of RedundancySets associated via ConcreteIdentity to the top-level system have level number 1. The members of redundancy sets associated to the level number 1 systems via ConcreteIdentity have level number 2. In general, the ComputerSystem members of redundancy sets associated to the level number n systems via ConcreteIdentity have level number n+1. The level of non-system objects is the level of the ComputerSystem instance associated to the object via associations such as SystemDevice, HostedAccessPoint, HostedService, or ElementCapabilities.

Figure 36: "System Level Numbers" demonstrates these system level numbers using the same configuration from Figure 35: "Multiple Redundancy Tier Instance Diagram". Note that ComponentCS diagrams are omitted from this diagram to avoid clutter.

Figure 36: System Level Numbers



All subclasses of CIM_Service and CIM_Capabilities shall have a level number less than or equal to the level number of storage classes (ports, volumes, etc.) that are influenced by the properties and methods of the Service and Capabilities classes. In some cases, different storage classes are influenced by different Service or Capabilities classes; the “level number less than or equal to” requirement may apply differently to different Service/Capabilities classes. It is always valid to associate Service and Capabilities classes to the top-level ComputerSystem since the level number of the top-level system (0) is always less than or equal to the level number of any other system.

Example 1 - An array with two controllers is modeled as a top-level ComputerSystem with real systems representing the controllers. The system’s storage volumes remain available when one controller fails, but each LogicalPort becomes unavailable when a controller fails. The StorageVolumes should be associated to the top-level ComputerSystem and the LogicalPorts should be associated to one of the real ComputerSystems.

Example 2 - An array with four pair-wise redundant controllers. Each LogicalPort is associated with a pair of controllers - if one controller in a pair fails, the port is still accessible through the alternate controller. This corresponds to Figure 35: "Multiple Redundancy Tier Instance Diagram"; the ports should be associated with one of the ComputerSystems in the middle tier.

A provider shall delete and create associations between ComputerSystems and logical elements (e.g., ports, logical devices) during failover or failback to represent changes in availability. This includes SystemDevice, HostedAccessPoint, HostedService, or HostedFileSystem associations (and other associations weak to systems). The effect of the creation and deletion of associations is to switch these elements from one ComputerSystem to another. The profiles that include Multiple Computer System Subprofile shall specify the affected associations and indications for creation and deletion of these associations.

Associations between ComputerSystems and PhysicalPackages and Products

The relationship between ComputerSystems, PhysicalPackages, and Products is defined in the Physical Package Package (see 8.2.1.10, "Physical Package Package") which may be required by the profile including the Multiple Computer System Subprofile. Typically, the top-level system is associated to a PhysicalPackage which is associated to a Product. Non-top-level systems may also be associated to PhysicalPackage and indirectly to a Product. If all underlying ComputerSystems share the same physical package, a single PhysicalPackage should be associated to the upper ComputerSystem.

The relationships between ComputerSystems, redundancy sets, and CIM logical elements serve as a redundancy topology - informing the client of the availability of subsets of logical elements. The relationships between PhysicalPackages and logical elements serve as a physical topology. These two topologies need not be equivalent. Consider these examples:

Example 1: a RAID array with a single controller (no redundancy); the controller and all backend disks are housed in a single chassis. This is modeled as a single ComputerSystem, no RedundancySets, no ComponentCS associations, and a single PhysicalPackage with a single associated Product.

Example 2: a RAID array with two redundant controllers; both controllers and all backend disks share a single chassis. In this case, the redundancy topology matches Figure 34: "Two Redundant Systems Instance Diagram". The top-level ComputerSystem is associated to a PhysicalPackage with a single associated Product.

Example 3: two arrays described in example 1 are assembled as part of common rack and sold as a single product. Note that although there are two controllers, there is no redundancy - the two controllers act completely independently. This is modeled as two top-level computer systems attached to separate PhysicalPackages (representing the two internal chassis); These two PhysicalPackages have a Container association to third PhysicalPackage representing the assembly - which has an association to a Product.

Example 4: two arrays described in Example 1 are assembled as part of a common rack and also share a high-speed trunk and a mutual failover capability. This failover capability means the two controllers share a RedundancySet and common top-level system. The result is similar to example 2, but each real ComputerSystem is now associated to separate PhysicalPackages which have Contiainer associations to a common PhysicalPackage.

Storage Systems without Multiple Systems

In configurations where the instrumentation does not model multiple ComputerSystem instances, all the associations described above reference the one and only ComputerSystem.

Durable Names and Correlatable IDs of the Subprofile

This subprofile does not impose any requirements on names. The requirements for ComputerSystem names are defined in the profiles that depend on Multiple Computer System Subprofile and in 6.2.4, "Correlatable and Durable Names". Clients should not expect that a network name or IP address is exposed as a ComputerSystem property. The Access Points subprofile should be used to model a network access point.

8.2.1.9.2 Health and Fault Management Considerations

The requirements for OperationalStatus of a ComputerSystem are discussed in 8.2.1.6, "Health Package".

8.2.1.9.3 Cascading Considerations

None

8.2.1.9.4 Supported Subprofiles and Packages

None.

8.2.1.9.5 Methods of the Profile

This subprofile does not include any extrinsic methods. A client may use this subprofile to discover information about the topology of computer systems, but cannot change the topology.

8.2.1.9.6 Client Considerations and Recipes

A client cannot generally, interoperably navigate the redundancy topology using ComponentCS because some Component CS associations may not parallel RedundancySet associations. But a client may use ComponentCS selectively to speed up certain tasks. In particular, a client may locate the top-level system from other ComputerSystems using ComponentCS.

8.2.1.9.6.1 Find Top-level Computer Systems

See 8.2.4.1.5.3, "Identify the ManagedElement Defined by a Profile". Top-level systems are the only objects in SMI-S associated to RegisteredProfile via ElementConformsToProfile.

8.2.1.9.6.2 Find the Top-level Computer System for any LogicalDevice

```
/
// DESCRIPTION:
// Find the Top-level Computer System for any CIM_LogicalDevice
//
// Preconditions:
//  $Device - Reference the LogicalDevice
//
// Find Systems associated to $Device
$Systems->[] = AssociatorNames($Device->,   // ObjectName
        “CIM_SystemDevice”,             // AssocClass
        “CIM_System”,                   // ResultClass
        “PartComponent”,                // Role
        “GroupComponent”)               // ResultRole
if ($Systems == null || $Systems->[].size != 1) {
    <ERROR! must be exactly one ComputerSystem Associated via
         SystemDevice to each LogicalDevice instance>
}

// System->[0] is the associated system; see if it’s the
// top-level system for the scoping profile.  All ComponentCS
// association GroupComponent references must refer to the
// profile’s top-level system.
$UpperSystems->[] = AssociatorNames($System->[0],
     “CIM_ComponentCS”,// AssocClass
     “CIM_ComputerSystem”,// ResultClass
     “PartComponent”,// Role
     “GroupComponent”)   // ResultRole
if ($UpperSystems != null && $UpperSystems->[].size > 1) {
// The restriction below is a characteristic of this subprofile
// and matches the DMTF Partinion white paper.
    <ERROR! must be no more than one ComputerSystem Associated
         via ComponentCS to each LogicalDevice instance>
}
// If an upper system was found, it must be the top-level
// system; if not, then the system associated to the device
// must be the top-level system
if ($UpperSystems->[].size == 1) {
  $TopLevelSystem =  $UpperSystems->[0]
} else {
  $TopLevelSystem = $System->[0]
}

// The remaining steps are not needed to locate the top-level
// system, but validate the classes and associations.
//
// The system associated to the device may also be part of a RedundancySet.
// If so, follow a chain from that system to the RedundancySet, then
// follow ConcreteIdentity to a system - then check to see if it has
// ConponentCS to the top-level system.  Keep iterating till no more
// RedundancySets - this must be the same system as TopLevelSystem.
do {
    // Get the RedundancySet that $System->[0] is a member of
    $RedundancySets->[] = AssociatorNames($System->[0],
               “CIM_MemberOfCollection”,
               “CIM_RedundancySet”,
               “Member”,
               “Collection”)
    if ($RedundancySets == null || $RedundancySets->[].size ==0) {
        #InARedundancySet = false
    } else {
        #InARedundancySet = true
     // Error is more than one RedundancySet
     if ($RedundancySets->[].size != 1) {
         <ERROR: A system cannot be the member of multiple RedundancySets>
     }
        $Systems->[] = AssociatorNames($RedundancySets->[0],   // ObjectName
            “CIM_ConcreteIdentity”,          // AssocClass
            “CIM_System”,                   // ResultClass
            “SameElement”,                  // Role
            “SystemElement”)                // ResultRole
        if ($Systems == null || $Systemss->[].size != 1) {
            <ERROR: There must be exactly one System associated to each
             RedundancySet>
     }
     // if System->[0] is not the TopLevelSystem, it must have ComponentCS
     if ($System->[0] != $TopLevelSystem) {
            $UpperSystems->[] = AssociatorNames($System->[0],
                “CIM_ComponentCS”,// AssocClass
             “CIM_ComputerSystem”,// ResultClass
             “PartComponent”,// Role
                “GroupComponent”)   // ResultRole
            if ($UpperSystems == null && $UpperSystems->[].size != 1) {
                <ERROR: must be no more than one ComputerSystem Associated
                   via ComponentCS to each LogicalDevice instance>
         }
            if ($UpperSystems->[0] != $TopLevelSystem) {
               <ERROR: The one end of every ComponentCS must be the Top Level
              system>
            }
     }
    }
} while (#InARedundancySet)
```

8.2.1.9.7 Registered Name and Version

Multiple Computer System version 1.1.0

8.2.1.9.8 CIM Server Requirements

Table 164: CIM Server Requirements for Multiple Computer System

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.9.9 CIM Elements

Table 165: CIM Elements for Multiple Computer System

Element Name	Description
Mandatory Classes	
CIM_ComponentCS (8.2.1.9.9.1)	Associates non-top-level systems to the top-level system
CIM_ComputerSystem (8.2.1.9.9.2)	Non-Top-level System
CIM_ConcreteIdentity (8.2.1.9.9.3)	Associates aggregate (possibly top-level) ComputerSystem and RedundancySet
CIM_MemberOfCollection (8.2.1.9.9.5)	Associates RedundancySet and its member ComputerSystems
CIM_RedundancySet (8.2.1.9.9.6)	
Optional Classes	
CIM_IsSpare (8.2.1.9.9.4)	optional
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Creation of a ComputerSystem instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Deletion of a ComputerSystem instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Change of Operational Status of a ComputerSystem instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	CQL - Change of Operational Status of a ComputerSystem instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_RedundancySet AND SourceInstance.RedundancyStatus <> PreviousInstance.RedundancyStatus	Deprecated WQL - Change of redundancy status

Table 165: CIM Elements for Multiple Computer System

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_RedundancySet AND SourceInstance.CIM_RedundancySet::RedundancyStatus <> PreviousInstance.CIM_RedundancySet::RedundancyStatus	CQL - Change of redundancy status

8.2.1.9.9.1 CIM_ComponentCS

Associates non-top-level systems to the top-level system
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 166: SMI Referenced Properties/Methods for CIM_ComponentCS

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ComputerSystem	The Top-Level ComputerSystem; must be associated to a RegisteredProfile
PartComponent		CIM_ComputerSystem	The contained (Sub)ComputerSystem

8.2.1.9.9.2 CIM_ComputerSystem

Non-Top-level system

Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 167: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	
NameFormat		string	Non-top-level system names are not correlatable, any format is valid
ElementName		string	
OperationalStatus		uint16[]	

8.2.1.9.9.3 CIM_ConcreteIdentity

Associates aggregate (possibly top-level) ComputerSystem and RedundancySet
Created By : Static or External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 168: SMI Referenced Properties/Methods for CIM_ConcreteIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemElement		CIM_ManagedElement	REF to the Computer System
SameElement		CIM_ManagedElement	REF to the RedundancySet

8.2.1.9.9.4 CIM_IsSpare

Associates the ComputerSystem that may be used as a spare to the RedundancySet of ActiveComputerSystem.

Class Mandatory: false

Table 169: SMI Referenced Properties/Methods for CIM_IsSpare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The spare system
Dependent		CIM_RedundancySet	The RedundancySet
SpareStatus		uint16	
FailoverSupported		uint16	

8.2.1.9.9.5 CIM_MemberOfCollection

Associates RedundancySet and its member ComputerSystems
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 170: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Member		CIM_ManagedElement	
Collection		CIM_Collection	

8.2.1.9.9.6 CIM_RedundancySet

Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 171: SMI Referenced Properties/Methods for CIM_RedundancySet

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	

Table 171: SMI Referenced Properties/Methods for CIM_RedundancySet

Property	Flags	Type	Description & Notes
RedundancyStatus		uint16	The redundancy status shall be either 'Unknown' 0, 'Redundant' 2, or 'Redundancy Lost'. 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 RedundancySet. It should report 3 when there are no more spares (via IsSpare association) per the RedundancySet.
TypeOfSet		uint16[]	
Optional Properties/Methods			
ElementName		string	

8.2.1.9.10 Related Standards

Table 172: Related Standards for Multiple Computer System

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

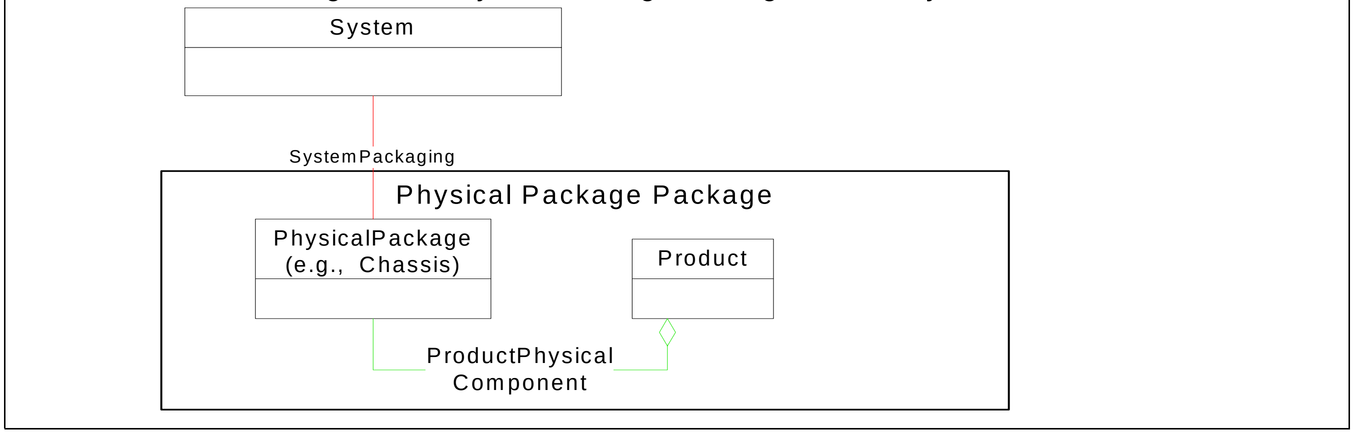
8.2.1.10 Physical Package Package

8.2.1.10.1 Description

Physical Package Package models information about a storage system’s physical package and optionally about internal sub-packages. A System is 'realized' using a SystemPackaging association to a PhysicalPackage (or a subclasses such as Chassis). The physical containment model can then be built up using Container associations and subclasses (such as PackageInChassis).

Physical elements are described as products using the Product class and ProductPhysicalComponent associations. The Product instances may be built up into a hierarchy using the ProductParentChild association. The Product class holds information such as vendor name, serial number and version.

Figure 37: Physical Package Package Mandatory Classes



Well Defined Subcomponents

In addition to defining physical packages at the “System” level, PhysicalPackage may also be defined at a lower, subcomponent level. For example, PhysicalPackage is used in the Disk Drive Lite Subprofile and for devices supported by storage media libraries (e.g., TapeDrive and ChangerDevice). If the subcomponents are supported by the Profile, they shall model their physical packaging. When subcomponents are modeled, there shall be a container relationship between their physical package and the containing package (e.g., the System level physical package). In addition, there shall be a ProductParentChild association between the subcomponent Product and the parent Product.

The Physical Package constructs may also be used to model other aspects of the environment. However, this is not mandatory. Note that each controller is realized by a card. The cards are contained in a controller chassis.

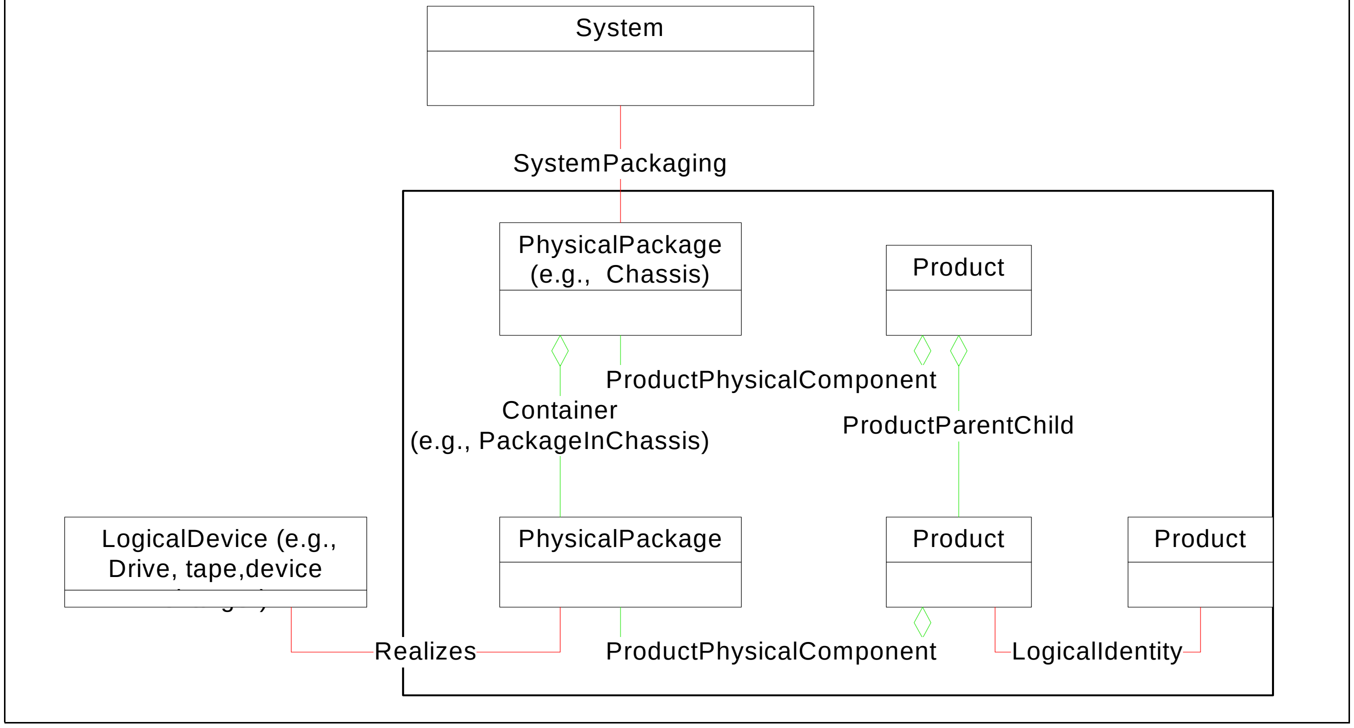
When establishing physical packages for subcomponents (e.g., disk drives, changers, etc.) the provider shall populate both Container and Realizes associations. Similarly, when establishing the Product instances for the packages the provider shall populate the ProductParentChild association to the parent product.

Multiple Product Identities

Instrumentation may optionally describe multiple product identities for a physical package, for example, product information for both an OEM and vendor. This information should be modeled as multiple instances of CIM_Product associated with the LogicalIdentity association. The Product instance that clients should treat as primary is directly associated with PhysicalPackage via ProductPhysicalComponent. Additional product instances are associated with the primary product using the LogicalIdentity association.

Figure 38: "Physical Package Package with Optional Classes" shows an example of the use of mandatory and optional physical package classes.

Figure 38: Physical Package Package with Optional Classes



8.2.1.10.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.10.3 Cascading Considerations

Not defined in this standard.

8.2.1.10.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.1.10.5 Methods of this Profile

Not defined in this standard.

8.2.1.10.6 Client Considerations and Recipes

Find Asset Information

Information about a system is modeled in PhysicalPackage. PhysicalPackage may be subclassed to Chassis; the more general PhysicalPackage is used here to accommodate device implementations that are deployed in multiple chassis. PhysicalPackage has an associated Product with physical asset information such as Vendor and Version.

Finding Product information

To locate product information (Vendor, Serial number and product versions) information about a device that is conforms to the profile, you would start with the “top-level” computer system and traverse the SystemPackaging to the PhysicalPackage (e.g., a Chassis). From the PhysicalPackage, the client would then traverse the ProductPhysicalComponent association to locate the Product instance. The primary Vendor, Serial Number and version for the device is in the Product instance associated with the

PhysicalPackage. Additional product identities may be associated with the primary Product using the LogicalIdentity association.

Finding Asset information

There are certain subcomponents of a device that a client may be interested in locating. For example, disk drives in an array or changer devices in a library. To locate the asset information of these subcomponents, the client would follow the ProductParentChild association from the system Product to lower level Products.

Alternatively, if the client is starting from a LogicalDevice, it can locate the PhysicalPackage by following the Realizes association from the LogicalDevice. From the PhysicalPackage, the client can find the Product information by traversing the ProductPhysicalComponent association.

8.2.1.10.7 Registered Name and Version

Physical Package version 1.1.0

8.2.1.10.8 CIM Server Requirements

Table 173: CIM Server Requirements for Physical Package

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.10.9 CIM Elements

Table 174: CIM Elements for Physical Package

Element Name	Description
Mandatory Classes	
CIM_PhysicalPackage (8.2.1.10.9.7)	
CIM_Product (8.2.1.10.9.8)	
CIM_ProductPhysicalComponent (8.2.1.10.9.10)	
CIM_SystemPackaging (8.2.1.10.9.12)	This association implement the 'realizes' relationship between a system and it's physical components. The LibraryPackage subclass should be used for Storage Media Libraries and the ComputerSystemPackage should be used for other profiles.
Optional Classes	
CIM_Card (8.2.1.10.9.1)	A subclass of PhysicalPackage which may be used to appropriately model a specific implementation
CIM_Chassis (8.2.1.10.9.2)	A subclass of PhysicalPackage which may be used to appropriately model a specific implementation
CIM_Container (8.2.1.10.9.3)	This associates a PhysicalPackage to its component physical packages (e.g., Drives in a Storage System). This may be subclassed (e.g., PackageInChassis), but only the Container properties are required
CIM_LogicalIdentity (8.2.1.10.9.4)	Required by the presence of CIM_Card
CIM_PackageInChassis (8.2.1.10.9.5)	Provided to allow component hierarchies
CIM_PhysicalConnector (8.2.1.10.9.6)	Required by the presence of CIM_Card
CIM_ProductParentChild (8.2.1.10.9.9)	If more than one product comprises a system, this association should be used to indicate the 'parent' product
CIM_Realizes (8.2.1.10.9.11)	Required by the presence of CIM_Card

8.2.1.10.9.1 CIM_Card

A subclass of PhysicalPackage which may be used to appropriately model a specific implementation
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.2 CIM_Chassis

A subclass of PhysicalPackage which may be used to appropriately model a specific implementation
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.3 CIM_Container

This associates a PhysicalPackage to its component physical packages (e.g., Drives in a Storage System). This may be subclassed (e.g., PackageInChassis), but only the Container properties are required

Class Mandatory: false

Table 175: SMI Referenced Properties/Methods for CIM_Container

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PhysicalPackage	
PartComponent		CIM_PhysicalElement	

8.2.1.10.9.4 CIM_LogicalIdentity

Required by the presence of CIM_Card
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.5 CIM_PackageInChassis

Provided to allow component hierarchies
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.6 CIM_PhysicalConnector

Required by the presence of CIM_Card
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.7 CIM_PhysicalPackage

Class Mandatory: true

Table 176: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
Manufacturer		string	
Model		string	
Optional Properties/Methods			
ElementName		string	
Name		string	
SerialNumber		string	
Version		string	
PartNumber		string	

8.2.1.10.9.8 CIM_Product

Class Mandatory: true

Table 177: SMI Referenced Properties/Methods for CIM_Product

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
IdentifyingNumber		string	
Vendor		string	
Version		string	
ElementName		string	

8.2.1.10.9.9 CIM_ProductParentChild

If more than one product comprises a system, this association should be used to indicate the 'parent' product
Class Mandatory: false

Table 178: SMI Referenced Properties/Methods for CIM_ProductParentChild

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Parent		CIM_Product	
Child		CIM_Product	

8.2.1.10.9.10 CIM_ProductPhysicalComponent

Class Mandatory: true

Table 179: SMI Referenced Properties/Methods for CIM_ProductPhysicalComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_Product	
PartComponent		CIM_PhysicalElement	

8.2.1.10.9.11 CIM_Realizes

Required by the presence of CIM_Card
Class Mandatory: false
No specified properties or methods.

8.2.1.10.9.12 CIM_SystemPackaging

This association implement the 'realizes' relationship between a system and it's physical components. The Library-Package subclass should be used for Storage Media Libraries and the ComputerSystemPackage should be used for other profiles.
Class Mandatory: true

Table 180: SMI Referenced Properties/Methods for CIM_SystemPackaging

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	

Table 180: SMI Referenced Properties/Methods for CIM_SystemPackaging

Property	Flags	Type	Description & Notes
Dependent		CIM_System	

8.2.1.10.10 Related Standards

Table 181: Related Standards for Physical Package

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.1.11 Policy Package

8.2.1.11.1 Description

The Policy Package would be deployed by any profile or subprofile that provides Policy management capability. Any profile or subprofile that supports the Policy Package is referred to as a "**Policy based**" profile or subprofile. In this version of SMI-S, there is no profile defined for a "**Global Policy Manager**" that provides policy management for a variety of other SMI-S profiles. The intent of this version of the SMI-S Policy Package is to support policy mechanisms "inside" Arrays, Storage Virtualizers, Volume Management, NAS, Storage Libraries, and Fabric components of a storage network. As a result, there are some limitations in this version of the Policy Package and there are some simplifying assumptions that can be made about the Policy mechanisms. For example, most arrays today don't support providing a general policy mechanism for a storage network. The policies and the context of the execution of the policies are confined to the array.

There are, however, some complications that will be dealt with. In particular, cascading profiles, such as Volume Management, Storage Virtualizers and NAS Heads will have to deal with policies that derive context from other profiles (e.g., arrays and/or fabric). Note: In the future, the Policy Package will be expanded to support a Specific Policy Profile as implemented in a Global Policy Manager and this may raise additional requirements.

Note: This Package covers "Policy-Based" support. That is, it only covers implementation of Policy constructs (classes and associations) in the policy based profile or subprofile. It does not cover requirements on underlying profiles that may be used by the policy based profile or subprofile.

It is important to understand the limitations of the Policy Package in this version of SMI-S. While one could argue that a host based volume manager has a broad view of the storage network and could, in theory, perform policy based SAN management, there is no expectation that a volume manager will (or should) be the vehicle for SAN management. The policies that would be supported by a Volume Management Profile would be policies for automating certain administrative functions of the volume manager.

8.2.1.11.1.1 Instance Diagrams

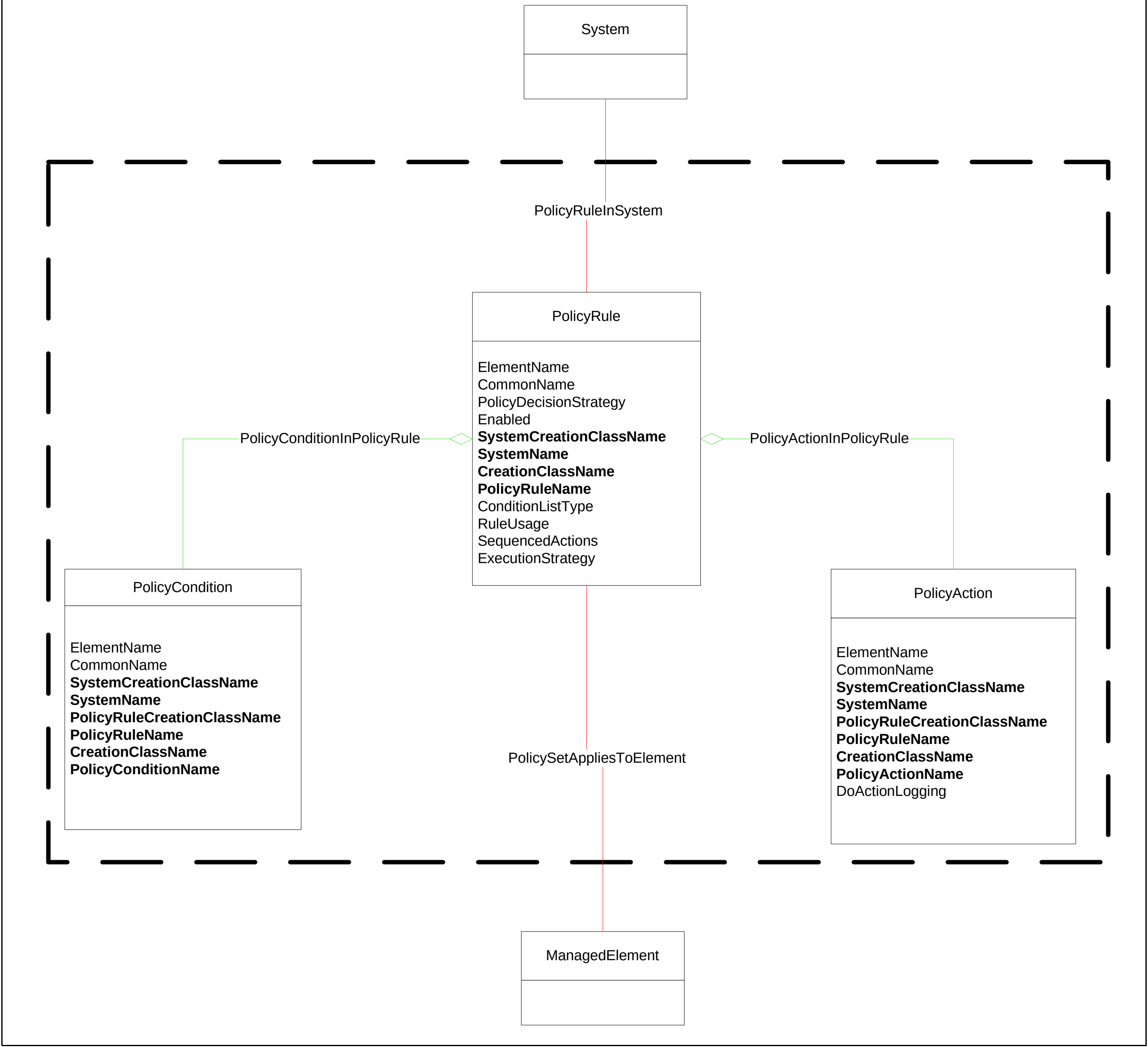
Support for the Policy Package entails support for a number of constructs and the methods to support them. Any given implementation may support only a subset of the constructs and methods, based on how flexible their support is. This will be discussed in more detail in Table 185, "SMI-S Supported PolicyCapabilities Patterns".

Policy constructs will be discussed in the following sections, starting with the basics and then building on those basics to describe more complicated functions and constructs.

Basics of Policy Support

The basic constructs used by the Policy Package are illustrated in Figure 39: "Basic Policy Package Instance Diagram"

Figure 39: Basic Policy Package Instance Diagram



There are five basic constructs that define a policy:

PolicyRule – This defines a policy to be applied. Specifically, it collects a number of other constructs that compose the policy.

PolicyCondition – A condition to be evaluated at the time the Policy Rule is checked. The PolicyCondition would be subclassed to a specific condition (e.g., QueryCondition) that can be evaluated in the context of the policy based profile or subprofile.

PolicyAction – An action to be executed based on conditions of the policy rule. The PolicyAction would be subclassed to a specific PolicyAction (e.g., a MethodAction) supported by the policy based profile or subprofile.

PolicySetAppliesToElement – An association that may be referenced by a PolicyCondition or PolicyAction (e.g., used as part of the query string in QueryConditions or MethodActions) to constrain the application of the PolicyRule. The “ManagedElement” would generally be any ManagedElement within the profile of the policy based profile or subprofile.

Note: In the case of a Policy-based cascading profile, the ManagedElement could be a reference to a ManagedElement in a leaf profile (see 8.2.1.11.3, "Cascading Considerations")

PolicyRuleInSystem – An association that is used to define the System scope of the PolicyRule. For policy based profiles or subprofiles, the system in question would be the “top level” system for the profile.

Note: In the case where a Policy-based cascading profile cascades to a Policy-based leaf profile, it is possible for a PolicyRule to be defined at the leaf and referenced by the cascading profile (i.e., cascading policy rules). See 8.2.1.11.3, "Cascading Considerations" for more information on this case.

In addition there are associations to define what Policy conditions are used in what Policy Rules (the PolicyConditionInPolicyRule association) and what Policy Actions are used by what Policy Rules (the PolicyActionInPolicyRule association).

A PolicyRule is the central class used for representing the 'If Condition then Action' semantics of a policy rule. A PolicyRule condition, in the most general sense, is represented as either an OR’ed set of AND’ed conditions (Disjunctive Normal Form, or DNF) or an AND’ed set of OR’ed conditions (Conjunctive Normal Form, or CNF). Individual conditions may either be negated (not C) or unnegated (C). The actions specified by a PolicyRule are to be performed if and only if the PolicyRule condition (whether it is represented in DNF or CNF) evaluates to TRUE.

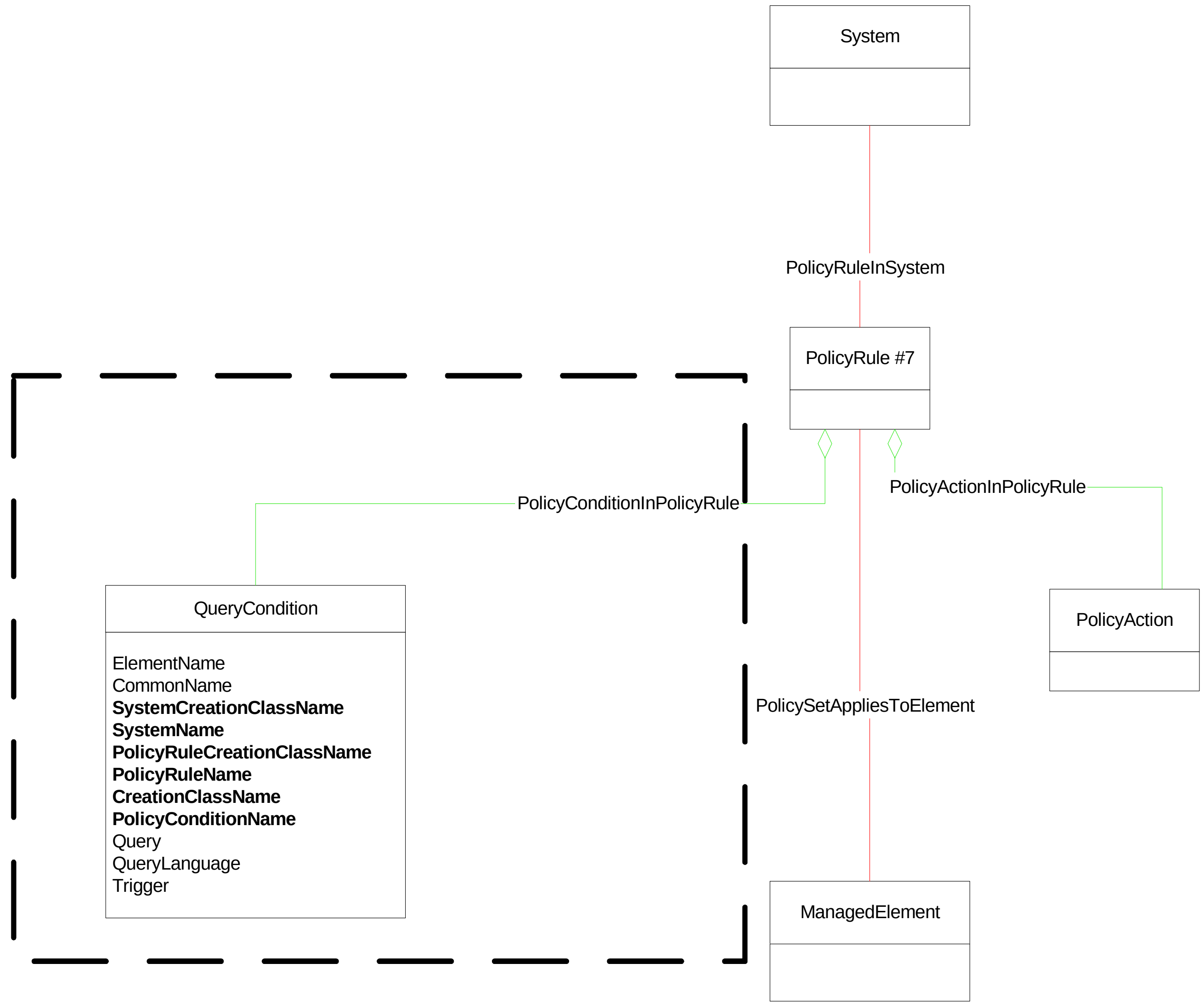
The conditions and actions associated with a PolicyRule are modeled, respectively, with instances of PolicyCondition and PolicyAction. These condition and action objects are tied to instances of PolicyRule by the PolicyConditionInPolicyRule and PolicyActionInPolicyRule aggregations.

The PolicyRule class uses the property ConditionListType, to indicate whether the conditions for the rule are in DNF (disjunctive normal form), CNF (conjunctive normal form) or, in the case of a rule with no conditions, as an UnconditionalRule. The PolicyConditionInPolicyRule aggregation contains two additional properties to complete the representation of the Rule's conditional expression. The first of these properties is an integer to partition the referenced PolicyConditions into one or more groups, and the second is a Boolean to indicate whether a referenced Condition is negated.

Query Conditions

The basic constructs used by QueryConditions are illustrated in Figure 40: "Policy Package QueryCondition Support Instance Diagram"

Figure 40: Policy Package QueryCondition Support Instance Diagram



A QueryCondition is a subclass of PolicyCondition that defines the criteria for generating a set of instances that result from the contained query. If there are no instances returned from the query, then the result is false; otherwise, true.

Note: A QueryCondition instance has a Trigger property. This property indicates whether or not the query is to be used to trigger evaluation of all QueryConditions of the PolicyRule. If the Trigger Boolean is set to TRUE, then the QueryCondition is a trigger. When the QueryCondition evaluates to TRUE, then all the QueryConditions are evaluated.

Note: None, some or all query conditions in a PolicyRule may have the Trigger Boolean set to TRUE. If no Trigger property is set to TRUE, then the conditions are to be periodically evaluated (with the period selected by the policy based profile or subprofile). See “Trigger Conditions” on page 215 in 8.2.1.11.1.1.

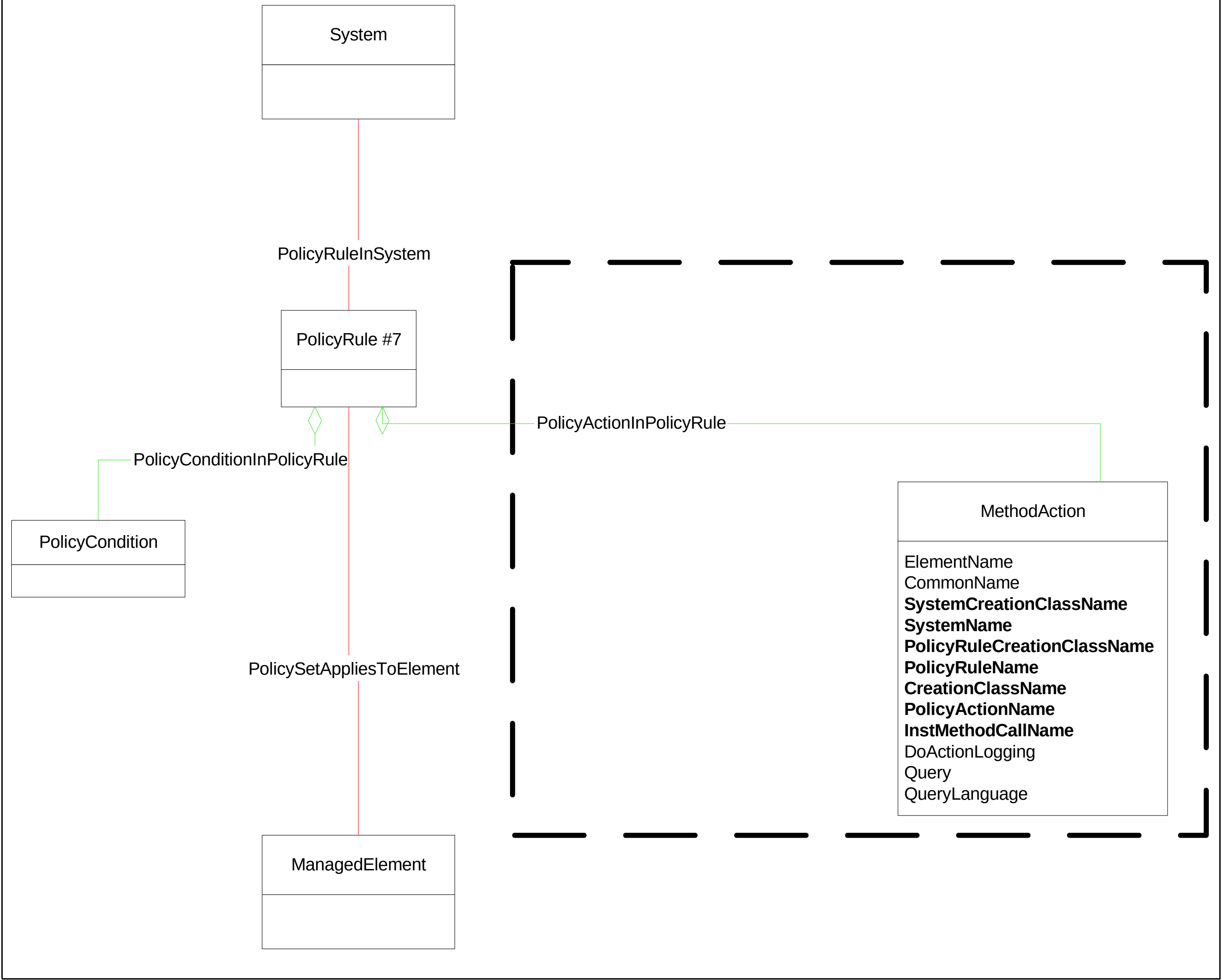
The following query is an example of a QueryCondition query that might be used:

```
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/primordial.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(primoridal) = hostedpool.PartComponent
    and OBJECTPATH(system) = hostedpool.GroupComponent
    and OBJECTPATH(system) = hosted.Antecedent
    and OBJECTPATH(service) = hosted.Dependent
    and service ISA "CIM_StorageConfigurationService"
```

MethodActions

The basic constructs for MethodActions of the Policy Package are illustrated in Figure 41: "Policy Package MethodAction Support Instance Diagram"

Figure 41: Policy Package MethodAction Support Instance Diagram



A MethodAction is a PolicyAction that is a method 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 ManagedElement. 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 result instance from a QueryCondition query, a MethodAction query, or other instances.

The following query is an example of a MethodAction query that might be used:

```
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
```

PolicySetAppliesToElement

PolicySetAppliesToElement makes explicit which PolicyRules are currently applied to a particular Element. This association indicates that the PolicyRules that are appropriate for a ManagedElement (specified using the PolicyRoleCollection aggregation) have actually been implemented 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 PolicyRule does not make use of the association. Note that if the named Element refers to a Collection, then the PolicyRule is assumed to be applied to all the members of the Collection.

PolicyRules are defined in the context of the System in which they apply. For policy based profiles or subprofiles, this is the “top level” system of the profile. The top level system can have many PolicyRules. A priority may be assigned to these rules using the Priority property of the PolicyRuleInSystem association.

Note: See 8.2.1.11.3, "Cascading Considerations" for a variation of this that involves cascading policy rules.

Context Passing

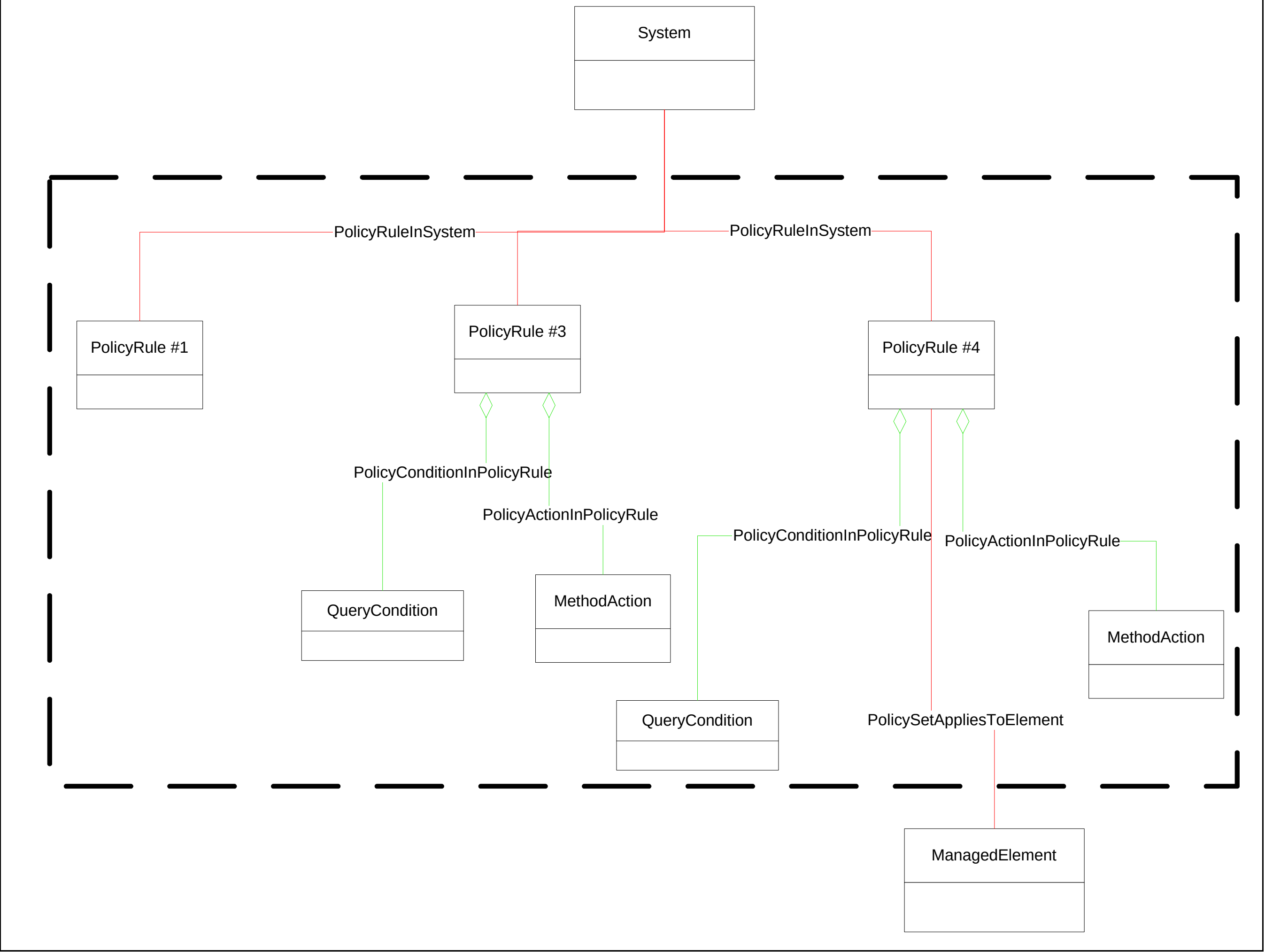
The execution of a PolicyRule involves establishing and naming the results of Query execution in QueryConditions and Queries associated with MethodActions. These Query results are transient instances that only exist in the context of the PolicyRule. The QueryResultName is a Property of QueryCondition that identifies the output of the query in the QueryCondition instance. The InstMethodCallName is a Property of a MethodAction that identifies the output of the query in the MethodAction instance.

Static Rules Support

A policy based profile or subprofile may support a set of “Static” PolicyRules. These are PolicyRules that cannot be modified by a client (except for enabling or disabling the rule or defining a

PolicySetAppliesToElement association). The constructs used for this are illustrated in Figure 42: "Policy Package for Static Rules Instance Diagram".

Figure 42: Policy Package for Static Rules Instance Diagram



The figure shows 3 static rules (PolicyRules #1, #3 and #4). These illustrate four distinct types of Static policy rules.

The first PolicyRule (PolicyRule #1) has no condition(s) and action(s) (or PolicySetAppliesToElement association). It merely names a specified policy rule. The only aspect of the PolicyRule that may (or may not) be changed is the “Enabled” property of the PolicyRule. This type of static policy rule is used to identify a behavior supported by the policy based profile. For example, Arrays might define a PolicyRule named “Controller Failover Type 1” or “Controller Failover Type 2” to indicate how controller failover works. Any particular Array Profile implementation would only support one of these PolicyRules. The client would determine behavior of failover by inspecting which PolicyRule is followed. But the actual behavior is not actually modeled in CIM. It is merely referenced using this simple form of static policy rules.

The second PolicyRule (PolicyRule #3) is has condition(s) and action(s), but is not referenced by any PolicySetAppliesToElement association. It behaves exactly like any other PolicyRule, except the QueryCondition(s) and MethodAction(s) are fixed and cannot be changed. The only aspects of the PolicyRule that may (or may not) be changed is the “Enabled” property of the PolicyRule. This type of static policy rule is more descriptive than the first, in that it models conditions that are evaluated and actions that are taken.

The third PolicyRule (PolicyRule #4) has condition(s) and action(s), and is referenced by a PolicySetAppliesToElement association. It behaves exactly like any other PolicyRule, except the QueryCondition(s) and MethodAction(s) are fixed and cannot be changed. The only aspects of the PolicyRule that may be changed are the "Enabled" property of the PolicyRule and the PolicySetAppliesToElement association (to identify the managed element in which to apply the rule). In this case, the Query Condition or MethodAction refers to the PolicySetAppliesToElement association to constrain where or how the policy rule is applied. This type of static Policy Rule can be applied to specific managed elements in the profile. For example, an Array PolicyRule might define a policy for automatic extension of a StoragePool. The application of this policy to specific StoragePools would be governed by use of the PolicySetAppliesToElement.

Note: All PolicyRules have a PolicyRuleInSystem association to the System in which the PolicyRule is evaluated. In most cases, this will be the Top Level Object (System) for the policy based profile (i.e., the RegisteredProfile that a specific Policy RegisteredSubprofile supports). In order for the execution of the Policy to be constrained to the profile in question the QueryConditions and MethodActions should include a reference to PolicyRuleInSystem.

EXPERIMENTAL

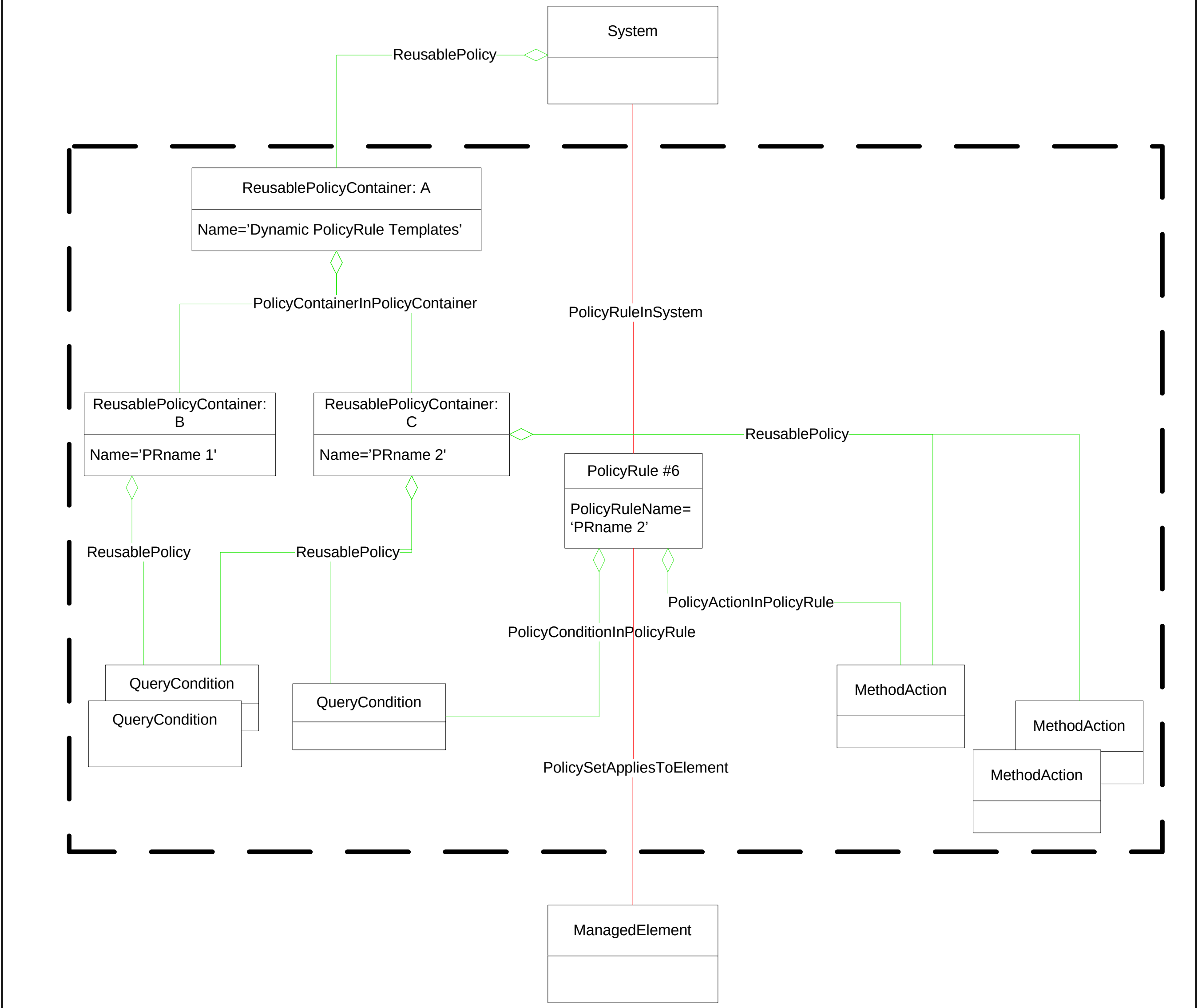
If any of these types of "Static Rules" are supported by a specific Policy Subprofile implementation then the PolicyFeaturesSupported array property of the PolicyCapabilities shall be set to include the "Static Rules" value.

EXPERIMENTAL

<u>Static Conditions and Actions</u>

In addition to Static Rules, there are “Dynamic” PolicyRules that can be constructed using static conditions and static actions. The constructs used for this are illustrated in Figure 43: "Policy Package Support for Static Conditions and Actions Instance Diagram".

Figure 43: Policy Package Support for Static Conditions and Actions Instance Diagram



Dynamic PolicyRules are constructed out of PolicyRule templates. In Figure 43: "Policy Package Support for Static Conditions and Actions Instance Diagram", PolicyContainer C is a template, and PolicyRule #6 is the policy rule constructed from the template. The PolicyContainer C merely collects all the “static” Conditions and “Static” actions that may be used to construct the PolicyRule. The ReusablePolicy associations are what connects the QueryConditions and MethodActions to the ReusablePolicyContainer (template). Note that a QueryCondition or MethodAction may appear in multiple ReusablePolicyContainers (e.g., ReusablePolicyContainer B and ReusablePolicyContainer C share a common QueryCondition).

To construct PolicyRule #6, the client would need to create PolicyRule #6 (giving it a client defined name) and creating the associations to the conditions and actions that are desired.

Note: Creation of the PolicyRule and the associations to QueryConditions and MethodActions are done using the CreateInstance intrinsic. Until all associations are in place and correctly configured, the “Enabled” property of the PolicyRule should be “disabled.” Once everything is in place and correct, the client may enable the rule).

EXPERIMENTAL

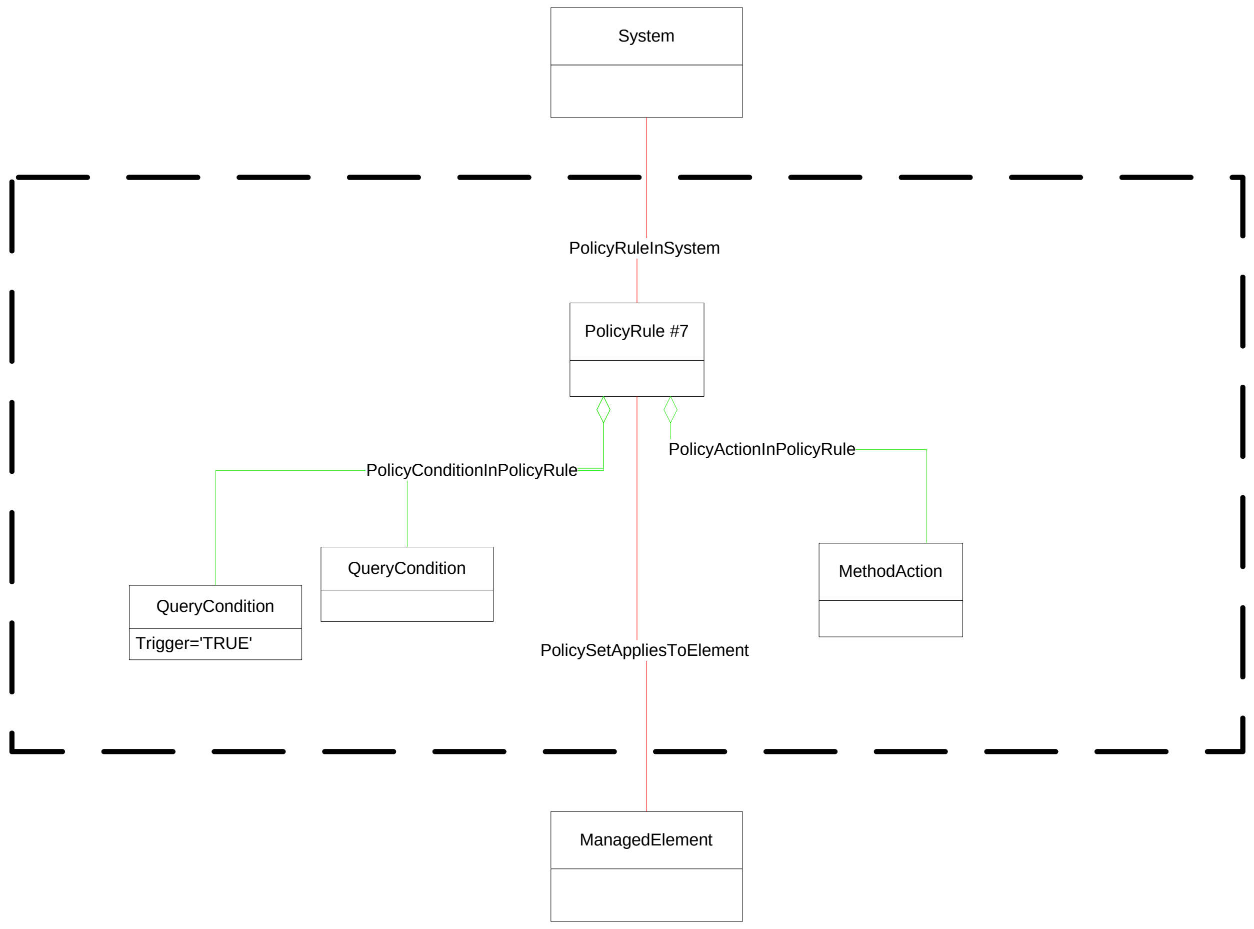
If any of these types of “Dynamic Rules” are supported by a specific Policy Subprofile implementation then the PolicyFeaturesSupported array property of the PolicyCapabilities shall be set to include the “Dynamic Rules” value.

EXPERIMENTAL

Dynamic Conditions and Actions

The most general policy support includes support for dynamic conditions and actions. The constructs used for this are the basic policy constructs as illustrated in Figure 44: "Policy Package support for Dynamic Conditions and Actions Instance Diagram".

Figure 44: Policy Package support for Dynamic Conditions and Actions Instance Diagram



In the dynamic conditions and actions case, all constructs are built using CreateInstance. The client would first create (and name) the PolicyRule, setting the Enabled property to ‘disabled’. Then the client would create the QueryConditions and MethodActions, and associate them to the PolicyRule.

Note: At least one QueryCondition should have a Trigger property of TRUE. If all the QueryConditions have a Trigger property of FALSE, the conditions will be evaluated at the convenience of the CIM server.

SMI-S only recognizes CQL Query statements in the QueryConditions. An implementation may support other QueryLanguages, but these would not be covered by SMI-S.

CQL defines “levels” of support. These levels are recognized for the purposes of Policy definitions. The CQL levels shall be identified in the CQLFeatures property of the QueryCapabilities instance associated to a specific Policy Subprofile (See Policy (and Query) Capabilities on Page 221 in 8.2.1.11.1.1.

EXPERIMENTAL

If this types of “Client defined rules” are supported by a specific Policy Subprofile implementation then the PolicyFeaturesSupported array property of the PolicyCapabilities shall be set to include the “Client Defined Rules” value.

EXPERIMENTAL

Trigger Conditions

Trigger Conditions are QueryConditions that, when TRUE, cause evaluation of all conditions in the Policy Rule. A trigger condition is a QueryCondition with the Trigger property set to TRUE. This is illustrated in Figure 45: "Policy Package support for Trigger Conditions Instance Diagram".

Figure 45: Policy Package support for Trigger Conditions Instance Diagram

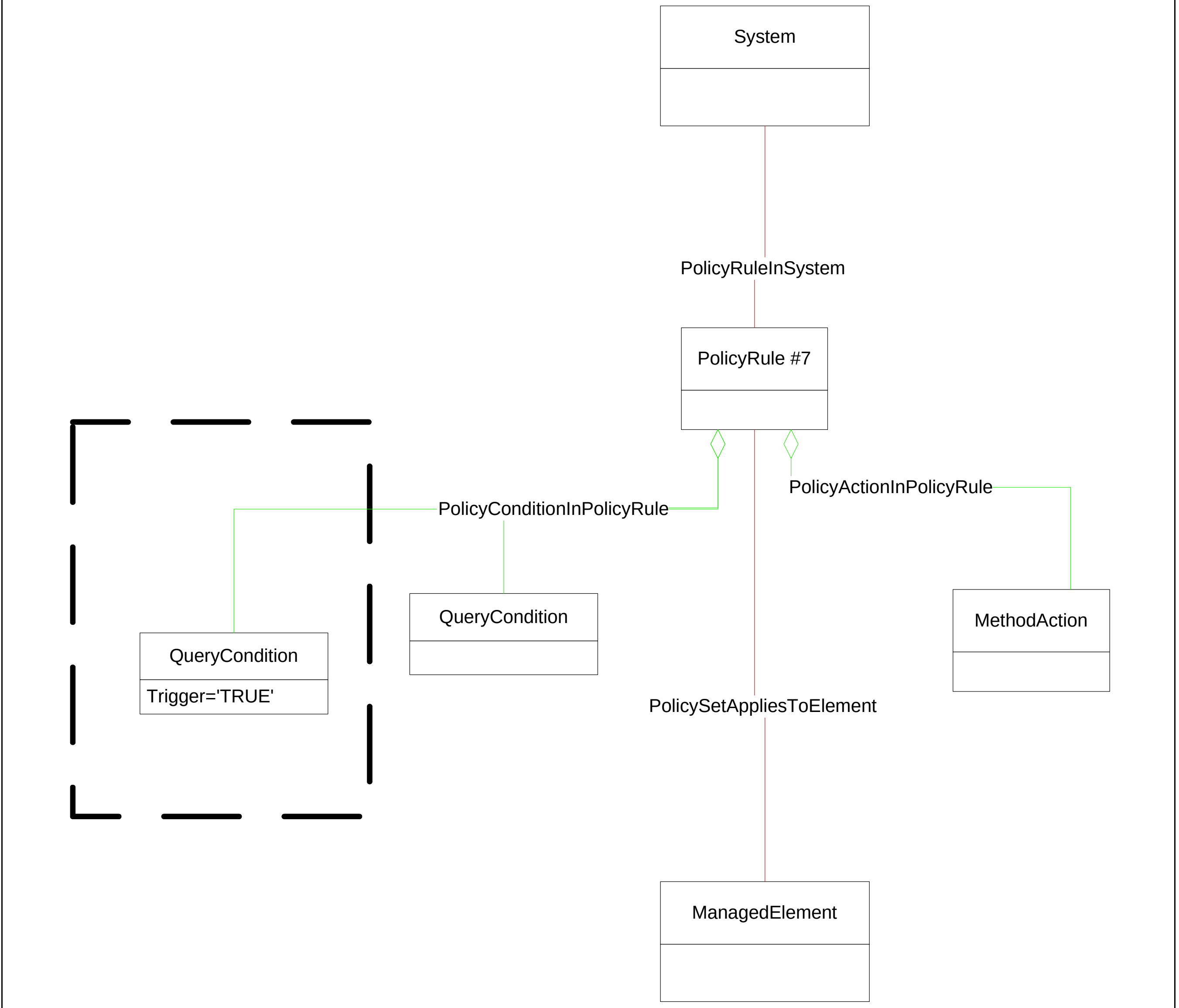
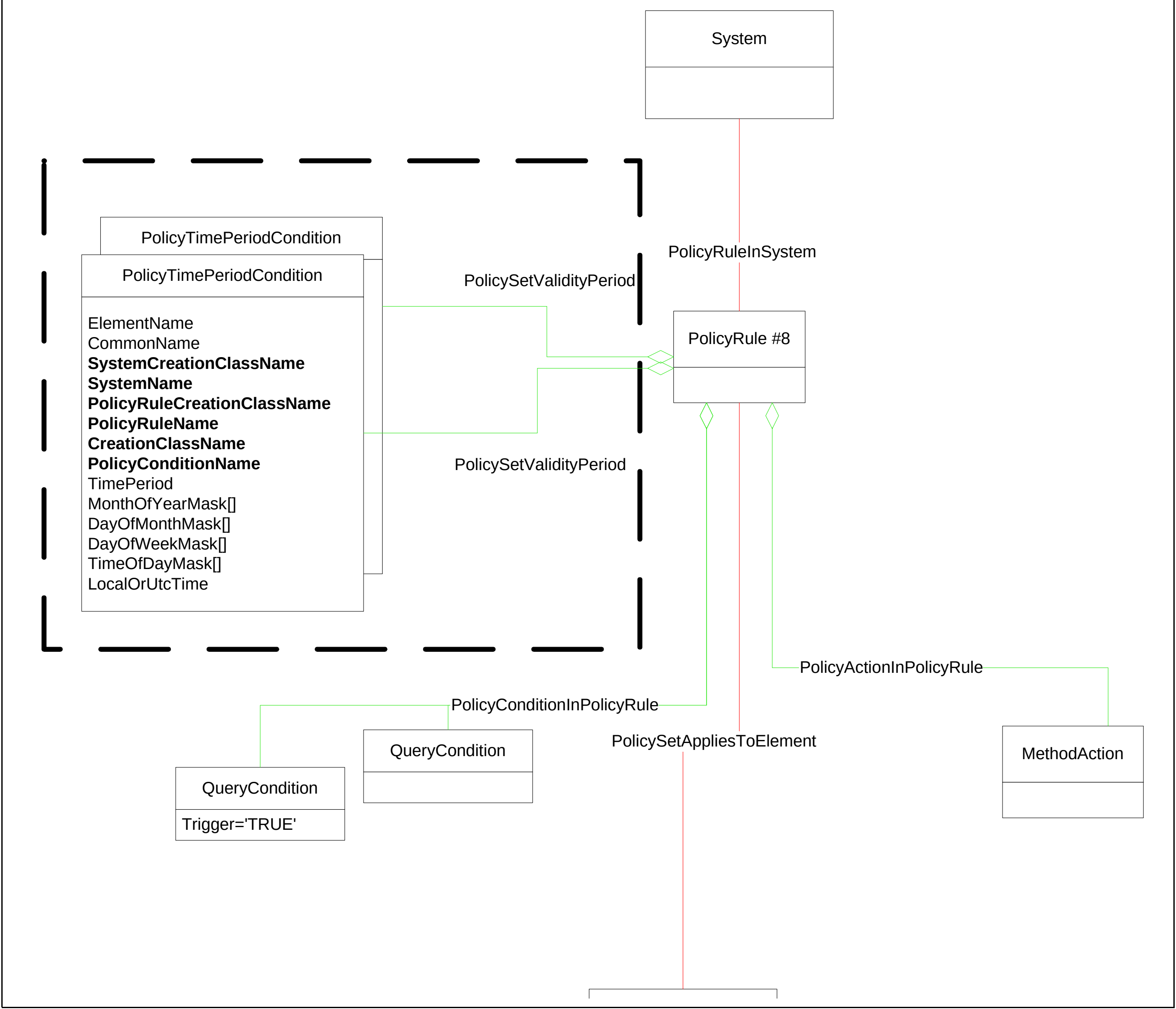


Figure 45: "Policy Package support for Trigger Conditions Instance Diagram" shows a PolicyRule with three QueryConditions. Two of the QueryConditions have Trigger set to TRUE. In the third, the Trigger property is set to FALSE. If either of the first two QueryConditions are true the third is evaluated.

TimePeriod Conditions

PolicyRules may be constrained by one or more time periods that define when the PolicyRule is to be active. The constructs used for this are illustrated inFigure 46: "Policy Package support for Time Periods Instance Diagram" .

Figure 46: Policy Package support for Time Periods Instance Diagram



A PolicyRule may also be associated with one or more policy time periods, indicating the schedule according to which the policy rule is active and inactive. In this case it is the PolicySetValidityPeriod aggregation that provides this linkage.

Evaluation of Policy conditions may be consider to be done in the following sequence:

1) Trigger Conditions - triggers are treated like indications to initiate evaluation of other conditions

2) TimePeriod Conditions - to determine if the remaining conditions need to be evaluated

3) Non-Trigger Conditions - the remaining Policy Conditions.

When there are compound conditions, the evaluation of each compound condition is evaluated independently. And the evaluation of a compound condition would follow the logical sequence described above.

When there are multiple PolicyTimePeriodConditions in a PolicyRule, then all shall evaluate to true. If there are no PolicyTimePeriodConditions specified in a PolicyRule, then all times are valid.

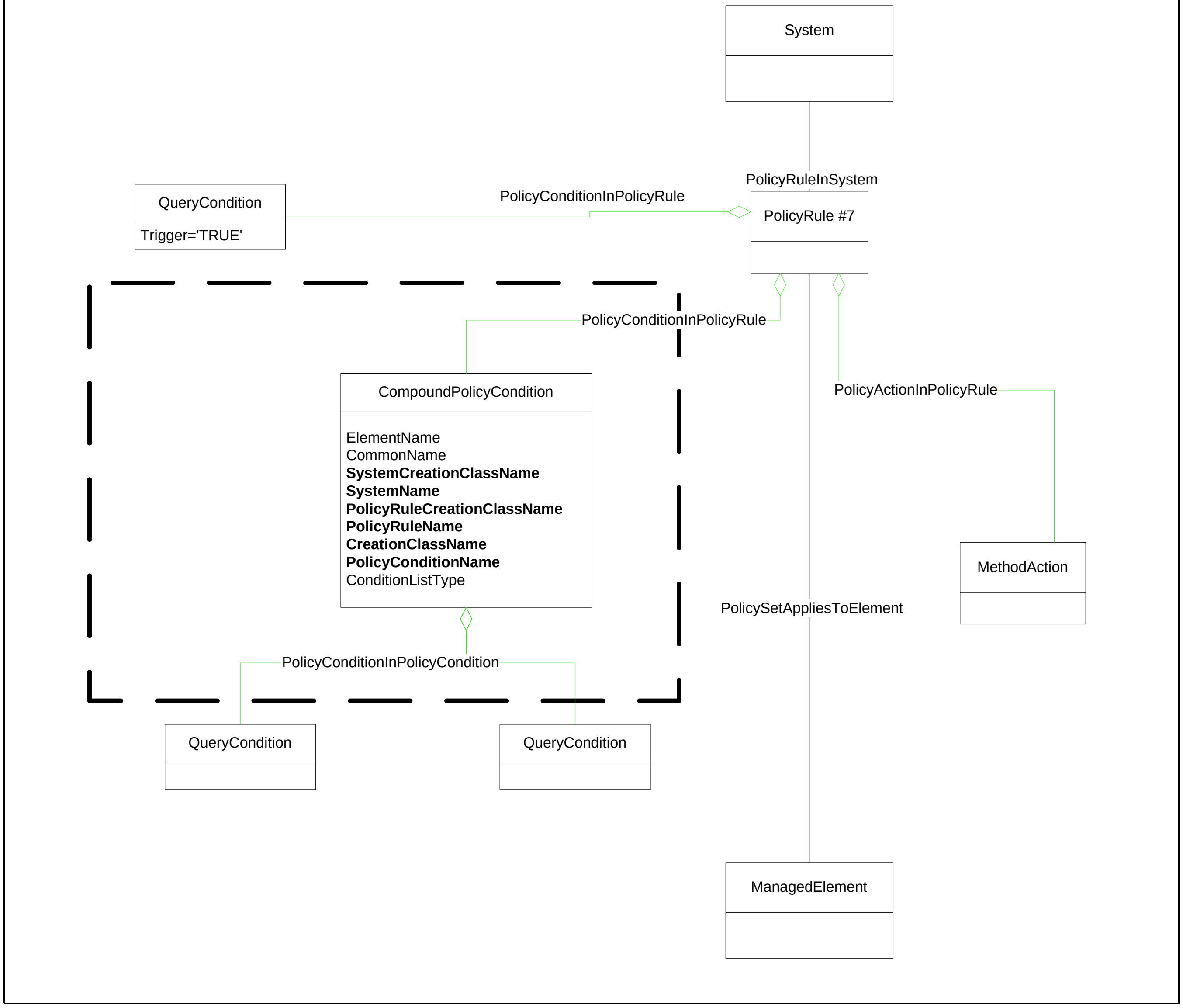
There are also two special cases in which one of the date/time strings is replaced with a special string defined in RFC 2445.

- If the first date/time is replaced with the string 'THISANDPRIOR', then the property indicates that a PolicyRule is valid [from now] until the date/time that appears after the '/'.
- If the second date/time is replaced with the string 'THISANDFUTURE', then the property indicates that a PolicyRule becomes valid on the date/time that appears before the '/', and remains valid from that point on.

Compound Conditions

QueryConditions may be aggregated into rules and into compound conditions. The constructs used for this are illustrated inFigure 47: "Policy Package support for Compound Conditions Instance Diagram"

Figure 47: Policy Package support for Compound Conditions Instance Diagram



A PolicyRule aggregates zero or more instances of the QueryCondition 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 enabled.

QueryConditions may be aggregated into rules and into compound conditions. PolicyConditionStructure is the abstract aggregation class for the structuring of policy conditions.

The Conditions aggregated by a PolicyRule or CompoundPolicyCondition are grouped into two levels of lists: either an OR'ed set of AND'ed sets of conditions (DNF, the default) or an AND'ed set of OR'ed sets of conditions (CNF). Individual QueryConditions in these lists may be negated. The property ConditionListType specifies which of these two grouping schemes applies to a particular PolicyRule or CompoundPolicyCondition instance.

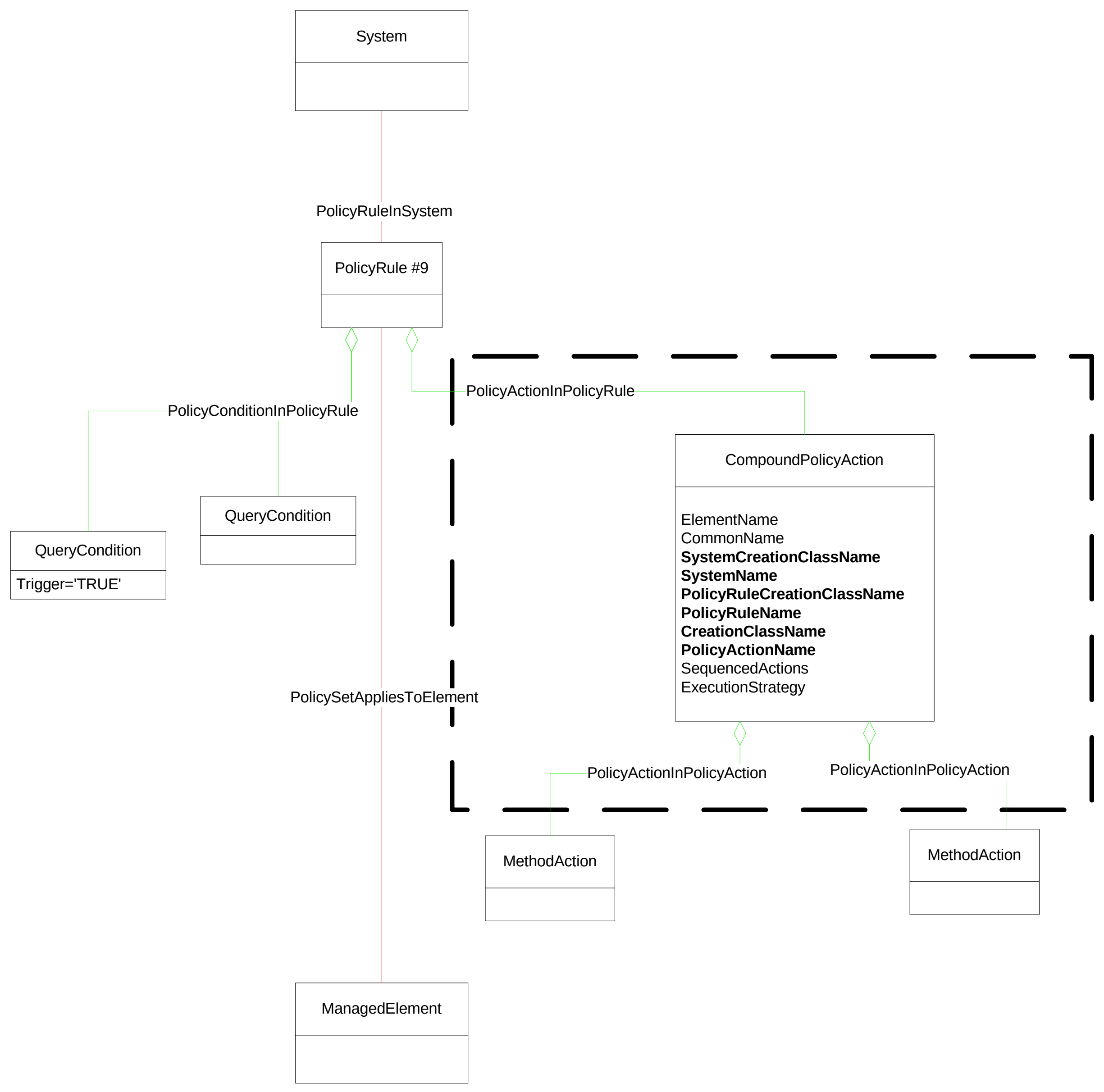
One or more PolicyTimePeriodConditions may be among the conditions associated with a PolicyRule or CompoundPolicyCondition via the PolicyConditionStructure subclass association. In this case, the time periods are simply additional Conditions to be evaluated along with any others that are specified.

A CompoundPolicyCondition aggregates zero or more instances of the QueryCondition class, via the PolicyConditionInPolicyCondition association. A CompoundPolicyCondition that aggregates zero Conditions is not valid; it may, however, be in the process of being defined. Note that a CompoundPolicyCondition should have no effect until it is valid.

Compound Actions

PolicyActions may be aggregated into rules and into compound actions. The constructs used for this are illustrated in Figure 48: "Policy Package support for Compound Actions Instance Diagram"

Figure 48: Policy Package support for Compound Actions Instance Diagram



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.

A series of examples will make ordering of PolicyActions clearer: 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. When 'n' is a positive integer, it indicates a place in the sequence of actions to be performed, with smaller integers indicating earlier positions in the sequence. The special value '0' indicates 'don't care'. If two or more PolicyActions have the same non-zero sequence number, they may be performed in any order, but they shall all be performed at the appropriate place in the overall action sequence.

If all actions have the same sequence number, regardless of whether it is '0' or non-zero, any order is acceptable.

The values:

1:ACTION A

2:ACTION B

1:ACTION C

3:ACTION D

indicate two acceptable orders: A,C,B,D or C,A,B,D,

since A and C can be performed in either order, but only at the '1' position.

The values:

0:ACTION A

2:ACTION B

3:ACTION C

3:ACTION D

require that B,C, and D occur either as B,C,D or as B,D,C. Action A may appear at any point relative to B, C, and D. Thus the complete set of acceptable orders is: A,B,C,D; B,A,C,D; B,C,A,D; B,C,D,A; A,B,D,C; B,A,D,C; B,D,A,C; B,D,C,A.

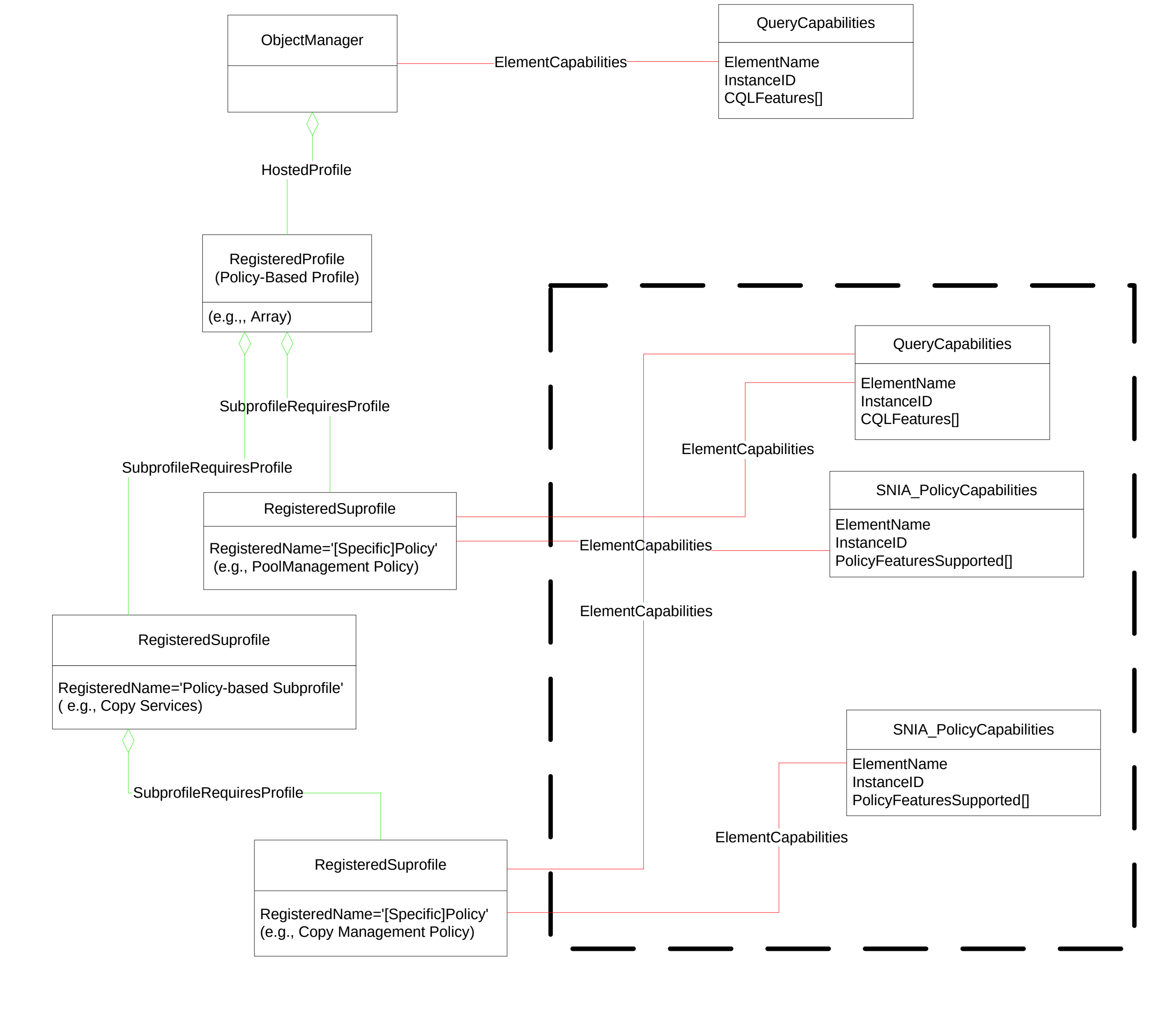
Note that the non-zero sequence numbers need not start with '1', and they need not be consecutive. All that matters is their relative magnitude.

EXPERIMENTAL

Policy (and Query) Capabilities

Implementations of a specific Policy Subprofile can vary in degree of support. The degree of support provided by an implementation can be determined by inspection of the QueryCapabilities and PolicyCapabilities. The constructs used for this are illustrated inFigure 49: "Policy Package support for Policy Capabilities Instance Diagram"

Figure 49: Policy Package support for Policy Capabilities Instance Diagram



In this figure, the policy based profile is an Array Profile. And it has two Specific Policy Subprofiles:

1) a Pool Management Policy Subprofile and,

2) a Copy Management Policy.

Each of these subprofiles shall have their Policy capabilities defined by associating an instance of PolicyCapabilities to each. Similarly, each may refer to a QueryCapabilities instance.

A policy based profile or subprofile would identify its basic capabilities using 2 capabilities classes: A QueryCapabilities class instance and a PolicyCapabilities class instance. Both instances will be associated to a specific Policy RegisteredSubprofile of the Policy-based RegisteredProfile. These classes and associations should be populated in the InterOp Namespace (with the RegisteredSubprofile). If they are populated in the policy based profile namespace, then the ElementCapabilities associations shall (at least) be populated in the InteropNamespace.

Also shown in Figure 49: "Policy Package support for Policy Capabilities Instance Diagram" are the ObjectManager (representing the CIM Server) and its QueryCapabilities instance. The QueryCapabilities instance that is associated to the ObjectManager represents the general capabilities of the CIM Server and may offer more capabilities than are supported for defining QueryConditions for PolicyRules. This instance is not part of the Policy Package (or either of the specific Policy Subprofiles). The ObjectManager version of the QueryCapabilities need not be present. The QueryCapabilities associated with a Specific Policy Subprofile is mandatory if the profile supports “Client defined” QueryConditions. If Client defined QueryConditions are not supported by the profile or subprofile, then the QueryCapabilities instance is not needed for the Specific Policy Subprofile.

The QueryCapabilities that may be supported for the purpose of client defined policies are “Basic Query”, “Simple Join”, “Complex Join”, “Time”, “Basic Like”, ”Full Like”, “Array Elements”, ”Embedded Objects“, “Order By”, “Aggregations”, “Subduer”, “Satisfies Array”, “Distinct”, “Forestland “Path Functions”. For definitions of these values see the *CIM Query Language Specification*.

Any or all of these may be specified in the QueryCapabilities associated with a specific Policy Subprofile.

The second capabilities instance associated to the specific Policy Subprofile is the PolicyCapabilities instance. The PolicyCapabilities class has the following properties that define the capabilities of the subprofile:

PolicyFeaturesSupported[]

“Static Rules” – Static rules are pre-defined by the profile implementation and are available to the Client to enable and disable (or set PolicySetAppliesToElement).

“Dynamic Rules” – Dynamic rules means that the profile implementation has populated PolicyContainers that include QueryConditions and MethodActions that can be constructed into Client specified PolicyRules (using the conditions and actions in the container).

“Client Defined Rules” – Client Defined Rules means that a Client may create its own PolicyRules, specifying its own (Client invented) QueryConditions and MethodActions. The QueryConditions and MethodActions shall, of course, be valid operations on the profile in question. That is, QueryConditions shall address class instances and properties that are part of the profile model and the MethodActions shall be actions supported by the profile.

EXPERIMENTAL

8.2.1.11.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.11.3 Cascading Considerations

Not defined in this standard.

8.2.1.11.4 Supported Subprofiles and Packages

None.

8.2.1.11.5 Methods of the Profile

EXPERIMENTAL

8.2.1.11.5.1 Extrinsic Methods of the Profile

There are no Extrinsic methods defined for this Package. All Policy manipulation actions are done using intrinsic methods. These are described in 8.2.1.11.5.2, "Intrinsic Methods of the Profile" and illustrated in 8.2.1.11.6, "Client Considerations and Recipes". However, it is recognized that some Extrinsic Methods may make Policy manipulation a lot easier and more efficient for clients. Such methods will be considered in a future release.

EXPERIMENTAL

8.2.1.11.5.2 Intrinsic Methods of the Profile

Table 182, “Static Policy Instance Manipulation Methods” 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 182: Static Policy Instance Manipulation 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 183 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 183: Dynamic Policy Instance Manipulation Methods

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
ModifyInstance	N/A	N/A	PolicyRule (Enabled)
ModifyInstance	N/A	N/A	QueryCondition (Trigger)

Table 184 identifies how Policy constructs get created, deleted, or modified for dynamic policies.

Table 184: Methods that cause Instance Creation, Deletion, or Modification of Dynamic Policy Rules

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	CompoundPolicyCondition	N/A	N/A
CreateInstance	PolicySetValidityPeriod	N/A	N/A
CreateInstance	PolicyTimePeriodCondition	N/A	N/A
CreateInstance	CompoundPolicyAction	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
DeleteInstance	N/A	PolicyConditionIn PolicyRule	N/A
DeleteInstance	N/A	PolicyActionInPolicyRule	N/A
DeleteInstance	N/A	CompoundPolicyCondition	N/A
DeleteInstance	N/A	PolicyConditionIn PolicyCondition	N/A
DeleteInstance	N/A	CompoundPolicyAction	N/A
DeleteInstance	N/A	PolicyActionInPolicyAction	N/A
DeleteInstance	N/A	PolicySetValidityPeriod	N/A
DeleteInstance	N/A	PolicyTimePeriodCondition	N/A
ModifyInstance	N/A	N/A	PolicyRule (Enabled)
ModifyInstance	N/A	N/A	PolicyRuleInSystem
ModifyInstance	N/A	N/A	QueryCondition (Trigger)
ModifyInstance	N/A	N/A	QueryCondition
ModifyInstance	N/A	N/A	MethodAction
ModifyInstance	N/A	N/A	PolicyConditionIn PolicyRule
ModifyInstance	N/A	N/A	PolicyActionInPolicyRule
ModifyInstance	N/A	N/A	CompoundPolicy Condi-tion

Table 184: Methods that cause Instance Creation, Deletion, or Modification of Dynamic Policy Rules

Method	Created Instances	Deleted Instances	Modified Instances
ModifyInstance	N/A	N/A	PolicyConditionIn PolicyCondition
ModifyInstance	N/A	N/A	CompoundPolicyAction
ModifyInstance	N/A	N/A	PolicyTimePeriod Condition
ModifyInstance	N/A	N/A	PolicyActionIn PolicyAction

CreateInstance

CreateInstance (

[IN] <instance> NewInstance

)

The CreateInstance intrinsic method is used for the creation of PolicyRules, QueryConditions, ReusablePolicyContainers and MethodActions, It is also used to create PolicyConditionInPolicyRule associations, PolicyConditionInPolicyCondition associations, ReusablePolicyComponent associations, PolicyActionInPolicyRule associations, PolicyActionInPolicyAction associations and PolicySetAppliesToElement associations.

Care should be taken when creating a policy. The following sequence should be followed for enabling **Static Policies**:

- Creation of the PolicyRule (disabled)
- Creation of the QueryCondition(s)
- Immediately followed by Creation of the PolicyConditionInPolicyRule association(s)
- Creation of the MethodAction(s)
- Immediately followed by Creation of the PolicyActionInPolicyRule association(s)
- Creation of one or more PolicySetAppliesToElement associations (if needed)
- ModifyInstance of the PolicyRule (to enable)

If this sequence in not followed, there is no guarantee that the desired Policy will be created. Also note that all steps would need to successfully execute to ensure creation of any of the instances involved in the PolicyRule.

If instances created are not immediately associated with an appropriate PolicyRule, they may be lost. A provider is not required to keep “dangling” instances around indefinitely. Indeed, they are expected to do periodic clean up of “dangling” instances.

The above sequence may not need to be done if there is no PolicySetAppliesToElement. In this case, all copies of the static policy are the same. All that is required is to enable (ModifyInstance) the PolicyRule.

The following sequence should be followed for creating **Dynamic PolicyRules**:

- Creation of the PolicyRule (disabled) (based on a ReusablePolicyContainer)

- Associate selected QueryConditions to the PolicyRule
- Associate selected MethodActions to the PolicyRule
- Create the appropriate PolicySetAppliesToElement associations
- ModifyInstance of the PolicyRule (to enable)

If this sequence in not followed, there is no guarantee that the desired Policy will be created. Also note that all steps would need to successfully execute to ensure creation of any of the instances involved in the PolicyRule

The following sequence should be followed for creating **Client Defined Policies**:

- Creation of the PolicyRule (disabled)
- Creation of the QueryCondition(s)
- Immediately followed by Creation of the PolicyConditionInPolicyRule association(s)
- Creation of the MethodAction(s)
- Immediately followed by Creation of the PolicyActionInPolicyRule association(s)
- Creation of one or more PolicySetAppliesToElement associations (if needed)
- ModifyInstance of the PolicyRule (to enable)

If this sequence in not followed, there is no guarantee that the desired Policy will be created. Also note that all steps would need to successfully execute to ensure creation of any of the instances involved in the PolicyRule

DeleteInstance

Not defined in this standard.

ModifyInstance

Not defined in this standard.

8.2.1.11.6 Client Considerations and Recipes

None.

8.2.1.11.6.1 SMI-S Supported PolicyCapabilities and QueryCapabilities Patterns

The PolicyCapabilities patterns that are formally recognized by this version of SMI-S are shown in Table 185, “SMI-S Supported PolicyCapabilities Patterns”.

Table 185: SMI-S Supported PolicyCapabilities Patterns

PolicyLevels Supported
Static Rules
Static Rules, Dynamic Rules
Static Rules, Client Defined Rules
Static Rules, Dynamic Rules, Client Defined Rules
Dynamic Rules
Dynamic Rules, Client Defined Rules
Client Defined Rules

8.2.1.11.7 Registered Name and Version

Policy version 1.1.0

8.2.1.11.8 CIM Server Requirements

Table 186: CIM Server Requirements for Policy

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.11.9 CIM Elements

Table 187: CIM Elements for Policy

Element Name	Description
Optional Classes	
CIM_CompoundPolicyAction (8.2.1.11.9.1)	A pre-defined Policy action that groups multiple method actions as a unit.
CIM_CompoundPolicyAction (8.2.1.11.9.2)	A Client defined Policy action that groups multiple method actions as a unit.
CIM_CompoundPolicyCondition (8.2.1.11.9.3)	A pre-defined Policy condition that groups multiple query conditions as a unit
CIM_CompoundPolicyCondition (8.2.1.11.9.4)	A Client defined Policy condition that groups multiple query conditions as a unit.
CIM_ElementCapabilities (8.2.1.11.9.5)	This associates the QueryCapabilities to the specific PolicyRegisteredSubprofile.
CIM_MethodAction (8.2.1.11.9.6)	Defines a Method (pre-defined) to be executed as part of a PolicyRule
CIM_MethodAction (8.2.1.11.9.7)	Defines a Method (Client defined) to be executed as part of a PolicyRule
CIM_PolicyActionInPolicyAction (8.2.1.11.9.8)	OPTIONAL: Associates a MethodAction to a pre-defined CompoundPolicyAction.
CIM_PolicyActionInPolicyAction (8.2.1.11.9.9)	OPTIONAL: Associates a MethodAction to a Client defined CompoundPolicyAction.
CIM_PolicyActionInPolicyRule (8.2.1.11.9.10)	OPTIONAL: Associates a MethodAction to the pre-defined PolicyRule of which it is a part.
CIM_PolicyActionInPolicyRule (8.2.1.11.9.11)	OPTIONAL: Associates a MethodAction to the Client defined PolicyRule of which it is a part.
CIM_PolicyConditionInPolicyCondition (8.2.1.11.9.12)	Associates a QueryCondition to a pre-defined CompoundPolicyCondition.

Table 187: CIM Elements for Policy

Element Name	Description
CIM_PolicyConditionInPolicyCondition (8.2.1.11.9.13)	Associates a QueryCondition to a Client defined CompoundPolicyCondition.
CIM_PolicyConditionInPolicyRule (8.2.1.11.9.14)	Associates a pre-defined QueryCondition to the PolicyRules of which it is part.
CIM_PolicyConditionInPolicyRule (8.2.1.11.9.15)	Associates a Client defined QueryCondition to the PolicyRules of which it is part.
CIM_PolicyContainerInPolicyContainer (8.2.1.11.9.16)	Association that collects PolicyContainers in other PolicyContainers.
CIM_PolicyRule (8.2.1.11.9.17)	Defines a Static (pre-defined) PolicyRule.
CIM_PolicyRule (8.2.1.11.9.18)	Defines a PolicyRule created by a client (Dynamic or Client Defined policy).
CIM_PolicyRuleInSystem (8.2.1.11.9.19)	Associates Static PolicyRules to the System that hosts them.
CIM_PolicyRuleInSystem (8.2.1.11.9.20)	Associates Dynamic or Client Defined PolicyRules to the System that hosts them.
CIM_PolicySetAppliesToElement (8.2.1.11.9.21)	An association that may be referenced in QueryConditions or MethodActions to constrain the application of a pre-defined PolicyRule. It associates the PolicyRule to ManagedElements.
CIM_PolicySetAppliesToElement (8.2.1.11.9.22)	An association that may be referenced in QueryConditions or MethodActions to constrain the application of a Dynamic or Client defined PolicyRule. It associates the PolicyRule to ManagedElements.
CIM_PolicySetValidityPeriod (8.2.1.11.9.23)	Associates a PolicyTimePeriodCondition to a pre-defined PolicyRule.
CIM_PolicySetValidityPeriod (8.2.1.11.9.24)	Associates a PolicyTimePeriodCondition to a Dynamic or client defined PolicyRule.
CIM_PolicyTimePeriodCondition (8.2.1.11.9.25)	A pre-defined PolicyCondition that specifies the valid time period for Policy activation.
CIM_PolicyTimePeriodCondition (8.2.1.11.9.26)	A Dynamic or Client defined PolicyCondition that specifies the valid time period for Policy activation.
CIM_QueryCapabilities (8.2.1.11.9.27)	Defines the Query execution capabilities of the profile or CIMOM.
CIM_QueryCondition (8.2.1.11.9.28)	A pre-defined 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.
CIM_QueryCondition (8.2.1.11.9.29)	A Dynamic or Client defined 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.
CIM_ReusablePolicy (8.2.1.11.9.30)	ReusablePolicy associates Policy Conditions and Policy Actions to a ReusablePolicyContainer. It is used for Dynamic Policy support.

Table 187: CIM Elements for Policy

Element Name	Description
CIM_ReusablePolicyContainer (8.2.1.11.9.31)	A ReusablePolicyContainer collects all the Policy Conditions and Actions that may be used in constructing a Dynamic PolicyRule.

8.2.1.11.9.1 CIM_CompoundPolicyAction

CompoundPolicyAction is used to represent an expression consisting of an ordered sequence of action terms. Each action term is represented as a subclass of the PolicyAction class. Compound actions are constructed by associating dependent action terms together using the PolicyActionInPolicyAction aggregation.

CompoundPolicyAction is subclassed from PolicyAction.

An instance of CompoundPolicyAction will exist if any compound actions exist.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 188: SMI Referenced Properties/Methods for CIM_CompoundPolicyAction (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyAction is defined.
PolicyRuleCreationClassName		string	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.
CreationClassName		string	The name of the class or the subclass used in the creation of an instance.
PolicyRuleName		string	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.
PolicyActionName		string	A provider generated user-friendly name of this policy (method) action
Optional Properties/Methods			
ElementName		string	Another provider generated user-friendly name
CommonName		string	A provider generated user-friendly name of the CompoundPolicyAction

Table 188: SMI Referenced Properties/Methods for CIM_CompoundPolicyAction (Pre-defined)

Property	Flags	Type	Description & Notes
DoActionLogging		boolean	
SequencedActions		uint16	This property gives a profile designer 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 -- I don't care about the order. The default value is 3 ("DontCare"). Values { "Mandatory", "Recom-mended", "Dont Care" }
ExecutionStrategy		uint16	A profile designed ExecutionStrategy defines the strategy to be used in exe-cuting the sequenced actions aggre-gated by this CompoundPolicyAction. There are three execution strategies: Do Until Success - execute actions according to predefined order, until successful execution of a single action. 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. Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance. The default value is 2 ("Do All"). Values { "Do Until Success", "Do All", "Do Until Failure" }

8.2.1.11.9.2 CIM_CompoundPolicyAction

CompoundPolicyAction is used to represent an expression consisting of an ordered sequence of action terms. Each action term is represented as a subclass of the PolicyAction class. Compound actions are constructed by associating dependent action terms together using the PolicyActionInPolicyAction aggregation.

CompoundPolicyAction is subclassed from PolicyAction.

An instance of CompoundPolicyAction will exist if any compound actions exist.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance

Class Mandatory: false

Table 189: SMI Referenced Properties/Methods for CIM_CompoundPolicyAction (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyAction is defined.
PolicyRuleCreationClassName		string	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.
CreationClassName		string	The name of the class or the subclass used in the creation of an instance.
PolicyRuleName		string	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.
PolicyActionName		string	A client defined user-friendly name of this policy (method) action
Optional Properties/Methods			
ElementName		string	Another Client defined user-friendly name
CommonName		string	A client-defined user-friendly name of the CompoundPolicyAction
DoActionLogging		boolean	
SequencedActions		uint16	This property gives a policy administrator (client) 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 -- I don't care about the order. The default value is 3 ("DontCare"). Values { "Mandatory", "Recommended", "Dont Care" }

Table 189: SMI Referenced Properties/Methods for CIM_CompoundPolicyAction (Client defined)

Property	Flags	Type	Description & Notes
ExecutionStrategy		uint16	ExecutionStrategy defines the strategy to be used in executing the sequenced actions aggregated by this CompoundPolicyAction. There are three execution strategies: Do Until Success - execute actions according to predefined order, until successful execution of a single action. 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. Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance. The default value is 2 ("Do All"). Values { "Do Until Success", "Do All", "Do Until Failure" }

8.2.1.11.9.3 CIM_CompoundPolicyCondition

CompoundPolicyCondition is used to represent compound conditions formed by aggregating simpler policy conditions. Compound conditions are constructed by associating subordinate condition terms together using the PolicyConditionInPolicyCondition aggregation.

CompoundPolicyCondition is subclassed from PolicyCondition.

An instance of CompoundPolicyCondition will exist if any pre-defined compound conditions exist.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 190: SMI Referenced Properties/Methods for CIM_CompoundPolicyCondition (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyCondition is defined.

Table 190: SMI Referenced Properties/Methods for CIM_CompoundPolicyCondition (Pre-defined)

Property	Flags	Type	Description & Notes
PolicyRuleCreationClassName		string	For a rule-specific PolicyCondition, the CreationClassName 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.
PolicyRuleName		string	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	The name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A provider supplied user-friendly name of this PolicyCondition.
Optional Properties/Methods			
ElementName		string	Another provider supplied user-friendly name
CommonName		string	A provider supplied user-friendly name of the CompoundPolicyCondition.
ConditionListType		uint16	Indicates whether the list of CompoundPolicyConditions associated with this PolicyRule Is in disjunctive normal form (DNF) or conjunctive normal form (CNF). The default value is 1 ("DNF"). Values { "DNF", "CNF" }

8.2.1.11.9.4 CIM_CompoundPolicyCondition

CompoundPolicyCondition is used to represent compound conditions formed by aggregating simpler policy conditions. Compound conditions are constructed by associating subordinate condition terms together using the PolicyConditionInPolicyCondition aggregation.

CompoundPolicyCondition is subclassed from PolicyCondition.

An instance of CompoundPolicyCondition will exist if any client defined compound conditions exist.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance

Class Mandatory: false

Table 191: SMI Referenced Properties/Methods for CIM_CompoundPolicyCondition (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName		string	For a rule-specific PolicyCondition, the CreationClassName 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.
PolicyRuleName		string	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	The name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A client defined user-friendly name of this PolicyCondition.
Optional Properties/Methods			
ElementName		string	Another client defined user-friendly name
CommonName		string	A client defined user-friendly name of the CompoundPolicyCondition.
ConditionListType		uint16	Indicates whether the list of CompoundPolicyConditions associated with this PolicyRule is in disjunctive normal form (DNF) or conjunctive normal form (CNF). The default value is 1 ("DNF"). Values { "DNF", "CNF" }

8.2.1.11.9.5 CIM_ElementCapabilities

CIM_ElementCapabilities represents the association between ManagedElements (i.e.,CIM_RegisteredSubprofile) and their Capabilities (e.g., CIM_QueryCapabilities).

CIM_ElementCapabilities is not subclassed from anything.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 192: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element (Registered-Subprofile)
Capabilities		CIM_Capabilities	The CIM_QueryCapabilities instance associated with the element.

8.2.1.11.9.6 CIM_MethodAction

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. 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.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 193: SMI Referenced Properties/Methods for CIM_MethodAction (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this MethodAction is defined.
PolicyRuleCreationClassName		string	For a rule-specific MethodAction, the CreationClassName of the PolicyRule object with which this Action is associ-ated. For a reusable MethodAction, a special value, 'NO RULE', should be used.
PolicyRuleName		string	For a rule-specific MethodAction, the name of the PolicyRule object with which this Action is associated. For a reusable MethodAction, a special value, 'NO RULE', should be used.
CreationClassName		string	The name of the class or the subclass used in the creation of an instance.

Table 193: SMI Referenced Properties/Methods for CIM_MethodAction (Pre-defined)

Property	Flags	Type	Description & Notes
PolicyActionName		string	A provider supplied user-friendly name of this policy (method) action
InstMethodCallName		string	In the context of the associated PolicyRule, InstMethodCallName defines a unique name for the query results that invoke the method specified in the Query string. It may be used in subsequent MethodActions of the same PolicyRule. This string is treated as a class name, in a query statement.
Query		string	The query that defines the method and the input parameters to that method.
QueryLanguage		uint16	This defines the query language being used, and for the current version of SMI-S, it shall be set to "2" (CQL).
Optional Properties/Methods			
ElementName		string	Another provider supplied user-friendly name
CommonName		string	A provider supplied user-friendly name of the MethodAction.
DoActionLogging		boolean	

8.2.1.11.9.7 CIM_MethodAction

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. 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.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 194: SMI Referenced Properties/Methods for CIM_MethodAction (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this MethodAction is defined.

Table 194: SMI Referenced Properties/Methods for CIM_MethodAction (Client defined)

Property	Flags	Type	Description & Notes
PolicyRuleCreationClassName		string	For a rule-specific MethodAction, the CreationClassName of the PolicyRule object with which this Action is associated. For a reusable MethodAction, a special value, 'NO RULE', should be used.
PolicyRuleName		string	For a rule-specific MethodAction, the name of the PolicyRule object with which this Action is associated. For a reusable MethodAction, a special value, 'NO RULE', should be used.
CreationClassName		string	The name of the class or the subclass used in the creation of an instance.
PolicyActionName		string	A client defined user-friendly name of this policy (method) action
InstMethodCallName		string	In the context of the associated PolicyRule, InstMethodCallName defines a unique name for the query results that invoke the method specified in the Query string. It may be used in subsequent MethodActions of the same PolicyRule. This string is treated as a class name, in a query statement.
Query		string	The query that defines the method and the input parameters to that method.
QueryLanguage		uint16	This defines the query language being used, and for the current version of SMI-S, this shall be set to "2" (CQL).
Optional Properties/Methods			
ElementName		string	Another client defined user-friendly name
CommonName		string	A client defined user-friendly name of the MethodAction.
DoActionLogging		boolean	

8.2.1.11.9.8 CIM_PolicyActionInPolicyAction

PolicyActionInPolicyAction is used to represent the compounding of policy actions into a higher-level policy action.

PolicyActionInPolicyAction is subclassed from PolicyActionStructure.

This association will exist if there is a pre-defined CompoundPolicyAction instance.

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

Class Mandatory: false

Table 195: SMI Referenced Properties/Methods for CIM_PolicyActionInPolicyAction (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_CompoundPolicyAction	This property represents the CompoundPolicyAction that contains one or more PolicyActions.
PartComponent		CIM_PolicyAction	This property holds the name of a PolicyAction contained by one or more CompoundPolicyActions.
Optional Properties/Methods			
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.

8.2.1.11.9.9 CIM_PolicyActionInPolicyAction

PolicyActionInPolicyAction is used to represent the compounding of policy actions into a higher-level policy action.

PolicyActionInPolicyAction is subclassed from PolicyActionStructure.

This association will exist if there is a Client defined CompoundPolicyAction instance.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 196: SMI Referenced Properties/Methods for CIM_PolicyActionInPolicyAction (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_CompoundPolicyAction	This property represents the CompoundPolicyAction that contains one or more PolicyActions.
PartComponent		CIM_PolicyAction	This property holds the name of a PolicyAction contained by one or more CompoundPolicyActions.
Optional Properties/Methods			
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.

8.2.1.11.9.10 CIM_PolicyActionInPolicyRule

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.

This association will exist if there are any Static PolicyRules that have MethodActions.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 197: SMI Referenced Properties/Methods for CIM_PolicyActionInPolicyRule (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PolicyRule	This property represents the PolicyRule that contains one or more PolicyAc-tions.
PartComponent		CIM_PolicyAction	This property holds the name of a Poli-cyAction contained by one or more Pol-icyRules.
Optional Properties/Methods			
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.

8.2.1.11.9.11 CIM_PolicyActionInPolicyRule

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 nosignificance; the property SequencedActions in the aggregating instance of PolicyRule provides this indication.

PolicyActionInPolicyRule is subclassed from PolicyActionStructure.

This association will exist if there are any Client defined PolicyRules that have MethodActions.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 198: SMI Referenced Properties/Methods for CIM_PolicyActionInPolicyRule (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PolicyRule	This property represents the PolicyRule that contains one or more PolicyActions.
PartComponent		CIM_PolicyAction	This property holds the name of a PolicyAction contained by one or more PolicyRules.
Optional Properties/Methods			
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.

8.2.1.11.9.12 CIM_PolicyConditionInPolicyCondition

A CompoundPolicyCondition aggregates zero or more instances of the PolicyCondition class, via the PolicyConditionInPolicyCondition association. A CompoundPolicyCondition that aggregates zero Conditions is not valid; it may, however, be in the process of being defined. Note that a CompoundPolicyCondition should have no effect until it is valid.

CIM_PolicyConditionInPolicyCondition is subclassed from CIM_PolicyConditionStructure.

There would be at least on instance of this association if there are any pre-defined CompoundPolicyConditions.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 199: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyCondition (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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”).

Table 199: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyCondition (Pre-defined)

Property	Flags	Type	Description & Notes
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		CIM_CompoundPolicyCondition	This property represents the CompoundPolicyCondition that contains one or more PolicyConditions.
PartComponent		CIM_PolicyCondition	This property holds the name of a PolicyCondition contained by one or more CompoundPolicyConditions.

8.2.1.11.9.13 CIM_PolicyConditionInPolicyCondition

A CompoundPolicyCondition aggregates zero or more instances of the PolicyCondition class, via the PolicyConditionInPolicyCondition association. A CompoundPolicyCondition that aggregates zero Conditions is not valid; it may, however, be in the process of being defined. Note that a CompoundPolicyCondition should have no effect until it is valid.

CIM_PolicyConditionInPolicyCondition is subclassed from CIM_PolicyConditionStructure.

There would be at least on instance of this association if there are any Client defined CompoundPolicyConditions.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 200: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyCondition (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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		CIM_CompoundPolicyCondition	This property represents the CompoundPolicyCondition that contains one or more PolicyConditions.

Table 200: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyCondition (Client defined)

Property	Flags	Type	Description & Notes
PartComponent		CIM_PolicyCondition	This property holds the name of a PolicyCondition contained by one or more CompoundPolicyConditions.

8.2.1.11.9.14 CIM_PolicyConditionInPolicyRule

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.

CIM_PolicyConditionInPolicyRule is subclassed from CIM_PolicyConditionStructure.

There would be one instance of this association for each pre-defined PolicyCondition in a PolicyRule.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 201: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyRule (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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		CIM_PolicyRule	This property represents the PolicyRule that contains one or more PolicyConditions.
PartComponent		CIM_PolicyCondition	This property holds the name of a PolicyCondition contained by one or more PolicyRules.

8.2.1.11.9.15 CIM_PolicyConditionInPolicyRule

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.

CIM_PolicyConditionInPolicyRule is subclassed from CIM_PolicyConditionStructure.

There would be one instance of this association for each client defined PolicyCondition in a PolicyRule.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 202: SMI Referenced Properties/Methods for CIM_PolicyConditionInPolicyRule (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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		CIM_PolicyRule	This property represents the PolicyRule that contains one or more PolicyConditions.
PartComponent		CIM_PolicyCondition	This property holds the name of a PolicyCondition contained by one or more PolicyRules.

8.2.1.11.9.16 CIM_PolicyContainerInPolicyContainer

A relationship that aggregates one or more lower-level ReusablePolicyContainer instances into a higher-level ReusablePolicyContainer.

CIM_PolicyContainerInPolicyContainer is subclassed form CIM_SystemComponent.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 203: SMI Referenced Properties/Methods for CIM_PolicyContainerInPolicyContainer (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ReusablePolicyContainer	A ReusablePolicyContainer that aggregates other ReusablePolicyContainers.
PartComponent		CIM_ReusablePolicyContainer	A ReusablePolicyContainer aggregated by another ReusablePolicyContainer.

8.2.1.11.9.17 CIM_PolicyRule

The central class used for representing the 'If Condition then Action' semantics of a policy rule. A PolicyRule condition, in the most general sense, is represented as either an ORed set of ANDed conditions (Disjunctive Normal Form, or DNF) or an ANDed set of ORed conditions (Conjunctive Normal Form, or CNF). Individual conditions may either be negated (not C) or unnegated (C). The actions specified by a PolicyRule are to be performed if and only if the PolicyRule condition (whether it is represented in DNF or CNF) evaluates to TRUE.

The conditions and actions associated with a PolicyRule are modeled, respectively, with subclasses of PolicyCondition and PolicyAction. These condition and action objects are tied to instances of PolicyRule by the PolicyConditionInPolicyRule and PolicyActionInPolicyRule aggregations.

A PolicyRule may also be associated with one or more policy time periods, indicating the schedule according to which the policy rule is active and inactive. In this case it is the PolicySetValidityPeriod aggregation that provides this linkage.

The PolicyRule class uses the property ConditionListType, to indicate whether the conditions for the rule are in DNF (disjunctive normal form), CNF (conjunctive normal form) or, in the case of a rule with no conditions, as an UnconditionalRule. The PolicyConditionInPolicyRule aggregation contains two additional properties to complete the representation of the Rule's conditional expression. The first of these properties is an integer to partition the referenced PolicyConditions into one or more groups, and the second is a Boolean to indicate whether a referenced Condition is negated. An example shows how ConditionListType and these two additional properties provide a unique representation of a set of PolicyConditions in either DNF or CNF.

Suppose we have a PolicyRule that aggregates five PolicyConditions C1 through C5, with the following values in the properties of the five PolicyConditionInPolicyRule associations:

C1: GroupNumber = 1, ConditionNegated = FALSE

C2: GroupNumber = 1, ConditionNegated = TRUE

C3: GroupNumber = 1, ConditionNegated = FALSE

C4: GroupNumber = 2, ConditionNegated = FALSE

C5: GroupNumber = 2, ConditionNegated = FALSE

If ConditionListType = DNF, then the overall condition for the PolicyRule is:

(C1 AND (not C2) AND C3) OR (C4 AND C5)

On the other hand, if ConditionListType = CNF, then the overall condition for the PolicyRule is:

(C1 OR (not C2) OR C3) AND (C4 OR C5)

In both cases, there is an unambiguous specification of the overall condition that is tested to determine whether to perform the PolicyActions associated with the PolicyRule.

PolicyRule instances may also be used to aggregate other PolicyRules and/or PolicyGroups. When used in this way to implement nested rules, the conditions of the aggregating rule apply to the subordinate rules as well. However, any side effects of condition evaluation or the execution of actions shall not affect the result of the evaluation of other conditions evaluated by the rule engine in the same evaluation pass. That is, an implementation of a rule engine may evaluate all conditions in any order before applying the priority and determining which actions are to be executed.

CIM_PolicyRule is subclassed from CIM_PolicySet.

There shall be at least one instance of PolicyRule for a policy based profile (a profile with an implementation of the Policy Package).

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 204: SMI Referenced Properties/Methods for CIM_PolicyRule (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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" }
Enabled		uint16	Indicates whether this PolicySet is administratively enabled or administratively disabled. SMI-S does not define a usage for 'Enabled for Debug', but it may be supported by an implementation. ValueMap { "1", "2", "3" }, Values { "Enabled", "Disabled", "Enabled For Debug" }
SystemCreationClassName		string	The scoping System's CreationClassName.
SystemName		string	The scoping System's Name.
CreationClassName		string	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyRuleName		string	A user-friendly name of this PolicyRule.
ExecutionStrategy		uint16	ExecutionStrategy defines the strategy to be used in executing the sequenced actions aggregated by this PolicyRule. There are three execution strategies: Do Until Success - execute actions according to predefined order, until successful execution of a single action. 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. Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance. Values { "Do Until Success", "Do All", "Do Until Failure" }

Table 204: SMI Referenced Properties/Methods for CIM_PolicyRule (Pre-defined)

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
ElementName		string	Another provider supplied user-friendly name
CommonName		string	A provider supplied user-friendly name of the policy rule
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 -- I don't care about the order. The default value is 3 ("DontCare"). Values { "Mandatory", "Recommended", "Dont Care" }

8.2.1.11.9.18 CIM_PolicyRule

Same rules as defined for pre-defined PolicyRules apply to Client Defined PolicyRules.

CIM_PolicyRule is subclassed from CIM_PolicySet.

There shall be at least one instance of PolicyRule for a policy based profile (a profile with an implementation of the Policy Package).

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance

Class Mandatory: false

Table 205: SMI Referenced Properties/Methods for CIM_PolicyRule (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
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" }
Enabled		uint16	Indicates whether this PolicySet is administratively enabled or administratively disabled. SMI-S does not define a usage for 'Enabled for Debug', but it may be supported by an implementation. ValueMap { "1", "2", "3" }, Values { "Enabled", "Disabled", "Enabled For Debug" }
SystemCreationClassName		string	The scoping System's CreationClassName.
SystemName		string	The scoping System's Name.
CreationClassName		string	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyRuleName		string	A user-friendly name of this PolicyRule.
ExecutionStrategy		uint16	ExecutionStrategy defines the strategy to be used in executing the sequenced actions aggregated by this PolicyRule. There are three execution strategies: Do Until Success - execute actions according to predefined order, until successful execution of a single action. 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. Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance. Values { "Do Until Success", "Do All", "Do Until Failure" }
Optional Properties/Methods			
ElementName		string	Another client defined user-friendly name
CommonName		string	A client defined user-friendly name of policy rule.

Table 205: SMI Referenced Properties/Methods for CIM_PolicyRule (Client defined)

Property	Flags	Type	Description & Notes
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 -- I don't care about the order. The default value is 3 ("DontCare"). Values { "Mandatory", "Recommended", "Dont Care" }

8.2.1.11.9.19 CIM_PolicyRuleInSystem

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.

CIM_PolicyRuleInSystem is subclassed from CIM_PolicySetInSystem.

There shall be at least one instance of this association for each Static Policy rule.

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

Class Mandatory: false

Table 206: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The System in whose scope a PolicyRule is defined.
Dependent		CIM_PolicyRule	A PolicyRule named within the scope of a System.
Optional Properties/Methods			
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.

8.2.1.11.9.20 CIM_PolicyRuleInSystem

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.

CIM_PolicyRuleInSystem is subclassed from CIM_PolicySetInSystem.

There shall be at least one instance of this association for each Dynamic or Client Defined Policy Rule.

Created By : CreateInstance
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 207: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The System in whose scope a PolicyRule is defined.
Dependent		CIM_PolicyRule	A PolicyRule named within the scope of a System.

Table 207: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem (Client defined)

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
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, there-fore, have no other relative priority defined. The priority is a non-negative integer; a larger value indicates a higher priority.

8.2.1.11.9.21 CIM_PolicySetAppliesToElement

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.

CIM_PolicySetAppliesToElement is not subclassed from anything.

An instance of this class may or may not exist.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 208: SMI Referenced Properties/Methods for CIM_PolicySetAppliesToElement (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_PolicySet	The PolicyRules and/or groups of rules that are currently applied to an Ele-ment.
ManagedElement		CIM_ManagedElement	The ManagedElement to which the Pol-icySet applies.

8.2.1.11.9.22 CIM_PolicySetAppliesToElement

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.

CIM_PolicySetAppliesToElement is not subclassed from anything.

An instance of this class may or may not exist.

Created By : CreateInstance
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 209: SMI Referenced Properties/Methods for CIM_PolicySetAppliesToElement (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_PolicySet	The PolicyRules and/or groups of rules that are currently applied to an Element.
ManagedElement		CIM_ManagedElement	The ManagedElement to which the PolicySet applies.

8.2.1.11.9.23 CIM_PolicySetValidityPeriod

The PolicySetValidityPeriod aggregation represents scheduled activation and deactivation of a PolicySet. A PolicySet is considered "active" if it is both "Enabled" and in a valid time period.

If a PolicySet is associated with multiple policy time periods via this association, then the Set is in a valid time period if at least one of the time periods evaluates to TRUE. If a PolicySet is contained in another PolicySet via the PolicySetComponent aggregation (e.g., a PolicyRule in a PolicyGroup), then the contained PolicySet (e.g., PolicyRule) is in a valid period if at least one of the aggregate's PolicyTimePeriodCondition instances evaluates to TRUE and at least one of its own PolicyTimePeriodCondition instances also evaluates to TRUE. (In other words, the PolicyTimePeriodConditions are ORed to determine whether the PolicySet is in a valid time period and then ANDed with the ORed PolicyTimePeriodConditions of each of PolicySet instances in the PolicySetComponent hierarchy to determine if the PolicySet is in a valid time period and, if also "Enabled", therefore, active, i.e., the hierarchy ANDs the ORed PolicyTimePeriodConditions of the elements of the hierarchy.

A Time Period may be aggregated by multiple PolicySets. A Set that does not point to a PolicyTimePeriodCondition via this association, from the point of view of scheduling, is always in a valid time period.

CIM_PolicySetValidityPeriod is subclassed from CIM_PolicyComponent.

An instance of this class may or may not exist.

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

Class Mandatory: false

Table 210: SMI Referenced Properties/Methods for CIM_PolicySetValidityPeriod (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PolicySet	This property contains the name of a PolicySet that contains one or more PolicyTimePeriodConditions.
PartComponent		CIM_PolicyTimePeriodCondition	This property contains the name of a PolicyTimePeriodCondition defining the valid time periods for one or more PolicySets.

8.2.1.11.9.24 CIM_PolicySetValidityPeriod

The rules for client defined PolicySetValidityPeriods are the same as those for pre-definedPolicySetValidityPeriods.

CIM_PolicySetValidityPeriod is subclassed from CIM_PolicyComponent.

An instance of this class may or may not exist.

Created By : CreateInstance
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 211: SMI Referenced Properties/Methods for CIM_PolicySetValidityPeriod (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PolicySet	This property contains the name of a PolicySet that contains one or more PolicyTimePeriodConditions.
PartComponent		CIM_PolicyTimePeriodCondition	This property contains the name of a PolicyTimePeriodCondition defining the valid time periods for one or more PolicySets.

8.2.1.11.9.25 CIM_PolicyTimePeriodCondition

This class provides a means of representing the time periods during which a PolicySet is valid, i.e., active. At all times that fall outside these time periods, the PolicySet has no effect. A PolicySet is treated as valid at ALL times, if it does not specify a PolicyTimePeriodCondition.

In some cases a Policy Consumer may need to perform certain setup / cleanup actions when a PolicySet becomes active / inactive. For example, sessions that were established while a PolicySet was active might need to be taken down when the PolicySet becomes inactive. In other cases, however, such sessions might be left up. In this case, the effect of deactivating the PolicySet would just be to prevent the establishment of new sessions.

Setup / cleanup behaviors on validity period transitions are not currently addressed by the Policy Model, and must be specified in 'guideline' documents or via subclasses of CIM_PolicySet, CIM_PolicyTimePeriod Condition or other concrete subclasses of CIM_Policy. If such behaviors need to be under the control of the policy administrator, then a mechanism to allow this control shall also be specified in the subclasses.

PolicyTimePeriodCondition is defined as a subclass of PolicyCondition. This is to allow the inclusion of time-based criteria in the AND/OR condition definitions for a PolicyRule.

Instances of this class may have up to five properties identifying time periods at different levels. The values of all the properties present in an instance are ANDed together to determine the validity period(s) for the instance. For example, an instance with an overall validity range of January 1, 2000 through December 31, 2000; a month mask that selects March and April; a day-of-the-week mask that selects Fridays; and a time of day range of 0800 through 1600 would be represented using the following time periods:

Friday, March 5, 2000, from 0800 through 1600;

Friday, March 12, 2000, from 0800 through 1600;

Friday, March 19, 2000, from 0800 through 1600;

Friday, March 26, 2000, from 0800 through 1600;

Friday, April 2, 2000, from 0800 through 1600;

Friday, April 9, 2000, from 0800 through 1600;

Friday, April 16, 2000, from 0800 through 1600;

Friday, April 23, 2000, from 0800 through 1600;

Friday, April 30, 2000, from 0800 through 1600.

Properties not present in an instance of PolicyTimePeriodCondition are implicitly treated as having their value 'always enabled'. Thus, in the example above, the day-of-the-month mask is not present, and so the validity period for the instance implicitly includes a day-of-the-month mask that selects all days of the month. If this 'missing property' rule is applied to its fullest, we see that there is a second way to indicate that a PolicySet is always enabled: associate with it an instance of PolicyTimePeriodCondition whose only properties with specific values are its key properties.

CIM_PolicyTimePeriodCondition is subclassed from CIM_PolicyCondition.

An instance of this class may or may not exist. If they exist, they can be found by following PolicyConditionInRule associations from PolicyRule instances or ReusablePolicy associations from ReusablePolicyContainer instances.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 212: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The name of the class or the subclass used in the creation of the System object in whose scope this PolicyCondition is defined.

Table 212: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Pre-defined)

Property	Flags	Type	Description & Notes
SystemName		string	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName		string	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	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	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A user-friendly name of this PolicyCondition.
Optional Properties/Methods			
ElementName		string	Another provider supplied user-friendly name.
CommonName		string	A provider supplied user-friendly name of policy object.
TimePeriod		string	This property identifies an overall range of calendar dates and times over which a PolicySet is valid. It is formatted as a string representing a start date and time, in which the character 'T' indicates the beginning of the time portion, followed by the solidus character '/', followed by a similar string representing an end date and time. The first date indicates the beginning of the range, while the second date indicates the end. Thus, the second date and time shall be later than the first. Date/times are expressed as substrings of the form yyyymmddThhmmss.

Table 212: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Pre-defined)

Property	Flags	Type	Description & Notes
MonthOfYearMask		uint8[]	The purpose of this property is to refine the valid time period that is defined by the TimePeriod property, by explicitly specifying in which months the PolicySet is valid. These properties work together, with the TimePeriod used to specify the overall time period in which the PolicySet is valid, and the MonthOfYearMask used to pick out the months during which the PolicySet is valid.
DayOfMonthMask		uint8[]	The purpose of this property is to refine the valid time period that is defined by the TimePeriod property, by explicitly specifying in which days of the month the PolicySet is valid. These properties work together, with the TimePeriod used to specify the overall time period in which the PolicySet is valid, and the DayOfMonthMask used to pick out the days of the month during which the PolicySet is valid.
DayOfWeekMask		uint8[]	The purpose of this property is to refine the valid time period that is defined by the TimePeriod property, by explicitly specifying in which days of the week the PolicySet is valid. These properties work together, with the TimePeriod used to specify the overall time period in which the PolicySet is valid, and the DayOfWeekMask used to pick out the days of the week during which the PolicySet is valid.
TimeOfDayMask		string	The purpose of this property is to refine the valid time period that is defined by the TimePeriod property, by explicitly specifying a range of times in a day during which the PolicySet is valid. These properties work together, with the TimePeriod used to specify the overall time period in which the PolicySet is valid, and the TimeOfDayMask used to pick out the range of time periods in a given day of during which the PolicySet is valid.

Table 212: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Pre-defined)

Property	Flags	Type	Description & Notes
LocalOrUtcTime		uint16	This property indicates whether the times represented in the TimePeriod property and in the various Mask properties represent local times or UTC times. There is no provision for mixing of local times and UTC times: the value of this property applies to all of the other time-related properties. TimePeriods are synchronized worldwide by using the enumeration value 'UTC-Time'.

8.2.1.11.9.26 CIM_PolicyTimePeriodCondition

The rules for client defined PolicyTimePeriodCondition are the same as those described for pre-defined PolicyTimePeriodCondition.

CIM_PolicyTimePeriodCondition is subclassed from CIM_PolicyCondition.

An instance of this class may or may not exist. If they exist, they can be found by following PolicyConditionInRule associations from PolicyRule instances or ReusablePolicy associations from ReusablePolicyContainer instances.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 213: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName		string	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.

Table 213: SMI Referenced Properties/Methods for CIM_PolicyTimePeriodCondition (Client defined)

Property	Flags	Type	Description & Notes
PolicyRuleName		string	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	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A user-friendly name of this PolicyCondition.
Optional Properties/Methods			
ElementName		string	Another client defined user-friendly name.
CommonName		string	A client defined user-friendly name of policy object
TimePeriod		string	
MonthOfYearMask		uint8[]	
DayOfMonthMask		uint8[]	
DayOfWeekMask		uint8[]	
TimeOfDayMask		string	
LocalOrUtcTime		uint16	

8.2.1.11.9.27 CIM_QueryCapabilities

This class defines the capabilities of the Specific Policy Subprofile associated via ElementCapabilities.

CIM_QueryCapabilities is subclassed from CIM_Capabilities.

An instance of this class may or may not exist. An instance of CIM_QueryCapabilities shall exist for each Specific Policy Subprofile that supports client defined queries.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 214: SMI Referenced Properties/Methods for CIM_QueryCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	This is a user-friendly name of the capabilities instance.

Table 214: SMI Referenced Properties/Methods for CIM_QueryCapabilities

Property	Flags	Type	Description & Notes
CQLFeatures		uint16[]	Enumeration of CQL features supported by an Object Manager or Provider associated via ElementCapabilities. (See DSP0202 CIM Query Language Specification for a normative definition of each feature.) Values {"Basic Query", "Simple Join", "Complex Join", "Time", "Basic Like", "Full Like", "Array Elements", "Embedded Objects", "Order By", "Aggregations", "Subquery", "Satisfies Array", "Distinct", "First", "Path Functions"}

8.2.1.11.9.28 CIM_QueryCondition

QueryCondition defines the criteria for generating a set of QueryConditionResult instances that result from the contained query. If there are no instances returned from the query, then the result is false; otherwise, true.

CIM_QueryCondition is subclassed from CIM_PolicyCondition.

QueryCondition instances may or may not exist. If they exist, they can be found by following PolicyConditionInRule associations from PolicyRule instances or ReusablePolicy associations from ReusablePolicyContainer instances.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 215: SMI Referenced Properties/Methods for CIM_QueryCondition (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName		string	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.

Table 215: SMI Referenced Properties/Methods for CIM_QueryCondition (Pre-defined)

Property	Flags	Type	Description & Notes
PolicyRuleName		string	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	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A user-friendly name of this PolicyCondition.
QueryResultName		string	In the context of the associated PolicyRule, QueryResultName defines a unique alias for the query results that may be used in subsequent QueryConditions or MethodActions of the same PolicyRule. This string is treated as a class name, in a query statement.
Query		string	A query expression that defines the condition(s) under which QueryConditionResult instances will be generated. The FROM clause may reference any class, including QueryConditionResult. NOTE THAT the property name, 'QueryConditionPath', shall not be used as the name of a select-list entry in the select-criteria clause of the query.
QueryLanguage		uint16	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	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.
Optional Properties/Methods			
ElementName		string	Another provider supplied user-friendly name

Table 215: SMI Referenced Properties/Methods for CIM_QueryCondition (Pre-defined)

Property	Flags	Type	Description & Notes
CommonName		string	A provider supplied user-friendly name of the QueryCondition

8.2.1.11.9.29 CIM_QueryCondition

QueryCondition defines the criteria for generating a set of QueryConditionResult instances that result from the contained query. If there are no instances returned from the query, then the result is false; otherwise, true.

CIM_QueryCondition is subclassed from CIM_PolicyCondition.

QueryCondition instances may or may not exist. If they exist, they can be found by following PolicyConditionInRule associations from PolicyRule instances or ReusablePolicy associations from ReusablePolicyContainer instances.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 216: SMI Referenced Properties/Methods for CIM_QueryCondition (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	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	The name of the System object in whose scope this PolicyCondition is defined.
PolicyRuleCreationClassName		string	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	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	CreationClassName indicates the name of the class or the subclass used in the creation of an instance.
PolicyConditionName		string	A user-friendly name of this PolicyCondition.

Table 216: SMI Referenced Properties/Methods for CIM_QueryCondition (Client defined)

Property	Flags	Type	Description & Notes
QueryResultName		string	In the context of the associated PolicyRule, QueryResultName defines a unique alias for the query results that may be used in subsequent QueryConditions or MethodActions of the same PolicyRule. This string is treated as a class name, in a query statement.
Query		string	A query expression that defines the condition(s) under which QueryConditionResult instances will be generated. The FROM clause may reference any class, including QueryConditionResult. NOTE THAT the property name, 'QueryConditionPath', shall not be used as the name of a select-list entry in the select-criteria clause of the query.
QueryLanguage		uint16	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	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.
Optional Properties/Methods			
ElementName		string	Another user-friendly name.
CommonName		string	User-friendly name of the QueryCondition.

8.2.1.11.9.30 CIM_ReusablePolicy

The ReusablePolicy association provides for the reuse of any subclass of Policy in a ReusablePolicyContainer. It is used in the Policy Package to associate the ReusablePolicyContainer (Dynamic PolicyRule templates) to the System in which the Dynamic PolicyRule can be defined.

CIM_ReusablePolicy is subclassed from CIM_PolicyInSystem.

This would only be supported if the Policy Package supports Dynamic PolicyRules, as defined in the PolicyFeaturesSupported property of the PolicyCapabilities instance for the Package. There would be one instance of ReusablePolicy for every Dynamic PolicyRule template supported by the profile.

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

Class Mandatory: false

Table 217: SMI Referenced Properties/Methods for CIM_ReusablePolicy

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ReusablePolicyContainer	This property identifies a ReusablePolicyContainer that is a holder for candidate Policies elements (conditions and actions) for defining a Dynamic PolicyRule.
Dependent		CIM_Policy	A reusable policy element (Condition or Action) that may be used in defining a Dynamic PolicyRule.

8.2.1.11.9.31 CIM_ReusablePolicyContainer

ReusablePolicyContainer is a class representing an administratively defined container for reusable policy-related information. This class does not introduce any additional properties beyond those in its superclass AdminDomain. It does, however, participate in a unique association for containing policy elements that may be used in constructing Dynamic PolicyRules.

An instance of this class uses the NameFormat value "ReusablePolicyContainer".

CIM_ReusablePolicyContainer is subclassed from CIM_AdminDomain.

This would only be supported if the Policy Package supports Dynamic PolicyRules, as defined in the PolicyFeaturesSupported property of the PolicyCapabilities instance for the Package. There would be one instance of ReusablePolicyContainer for every Dynamic PolicyRule template supported by the profile.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 218: SMI Referenced Properties/Methods for CIM_ReusablePolicyContainer

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	This should be the Name of the PolicyRule Template as specified in the profile.
NameFormat		string	This shall be set to ReusablePolicyContainer

8.2.1.11.10 Related Standards

Table 219: Related Standards for Policy

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Query Specification	1.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.1.12 Software Installation Service Subprofile

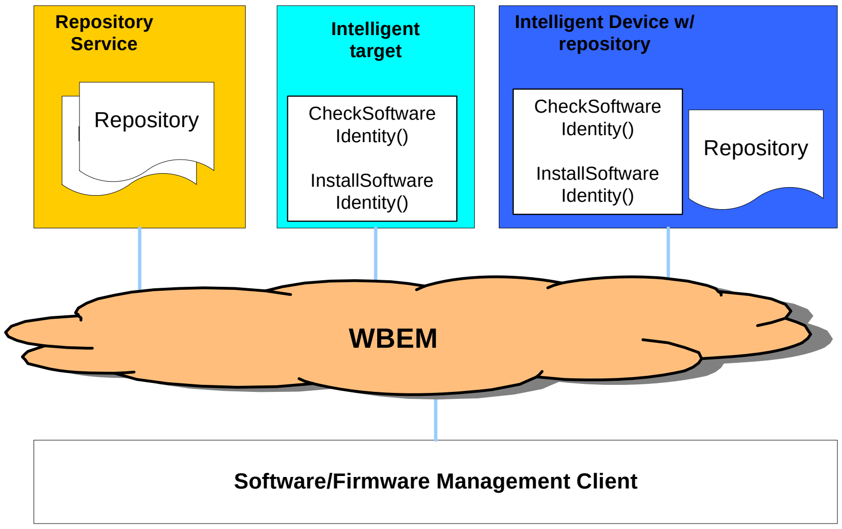
8.2.1.12.1 Description

This profile extends on the Software Package/Subprofile which enables a managed element to advertise version information for installed software elements.

The Software Installation Service Subprofile defines three methods to download and install software (including firmware) using a SMI-S based mechanism. This subprofile defines the use of one of them, which can be used interoperable. The following use cases are considered:

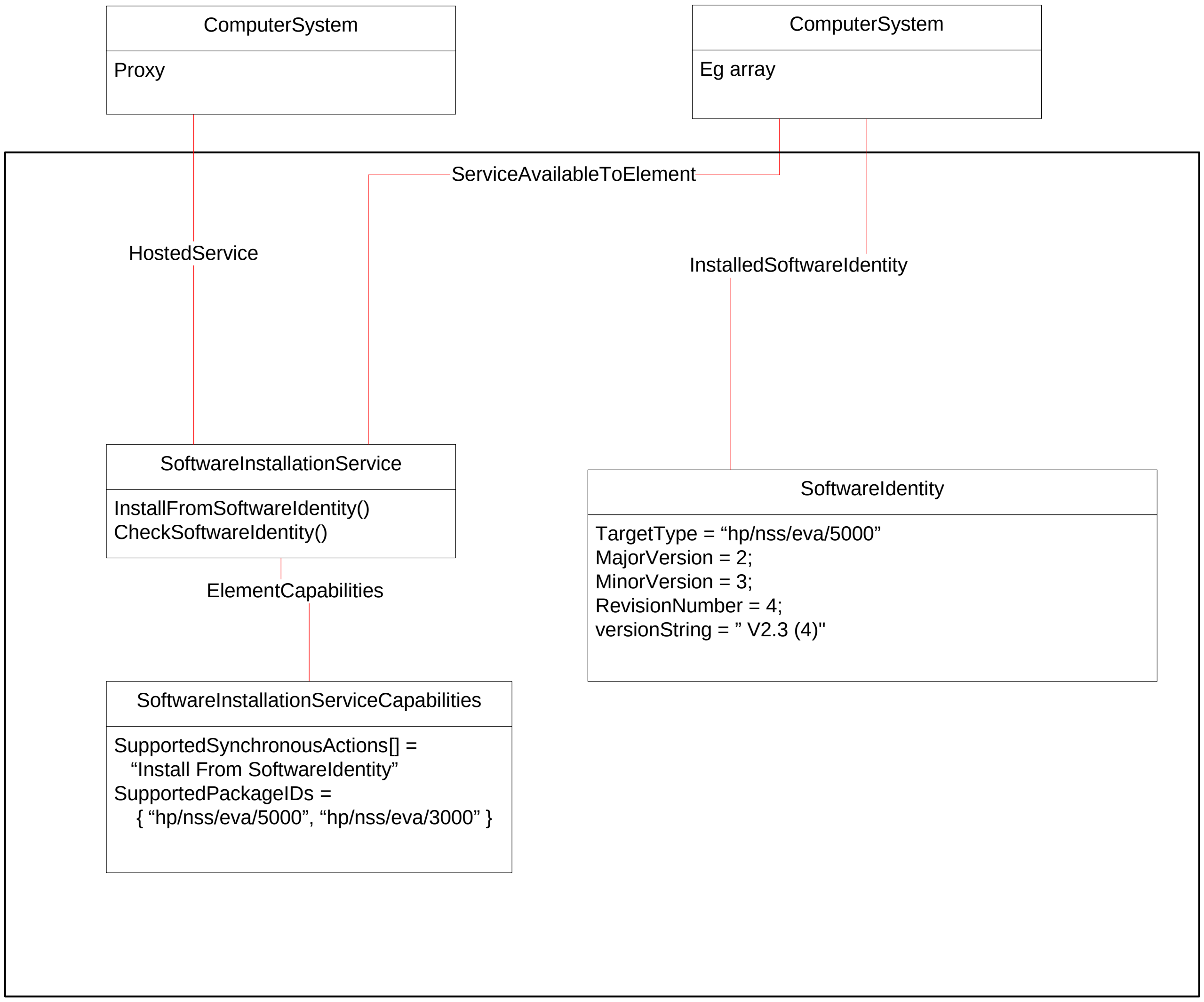
- A device (or provider) that can use a passed CIMObjectPath to a SoftwareIdentity in a separate namespace (or other ‘external’ repository) as a reference to an update.
- A device that also has it's own software repository and (may) find it's own updates from the web or elsewhere and advertise them.

Figure 50: Software Installation Service Overview


Repository Service
Repository
Intelligent target
CheckSoftware Identity()
InstallSoftware Identity()
Intelligent Device w/ repository
CheckSoftware Identity()
InstallSoftware Identity()
Repository
WBEM
Software/Firmware Management Client

This subprofile is closely related to the Software Repository subprofile and both can be used together or separately within a single device. The Software Repository subprofile provides a mechanism for exposing ‘candidate’ SoftwareIndentitys which can be selected by a client application and ‘applied’ to the element managed by the SoftwareInstallationService. The Software Repository and the Software Installation Service subprofiles expose enough information so that a management client can interoperable choose applicable SoftwareIdentity and understand freshening. No information is exposed about dependencies, however, the CheckSoftwareIndentity method can be used to determine if there are missing dependencies.

Figure 51: Example Instance Diagram


ComputerSystem
Proxy
ComputerSystem
Eg array
ServiceAvailableToElement
HostedService
InstalledSoftwareIdentity
SoftwareInstallationService
InstallFromSoftwareIdentity()
CheckSoftwareIdentity()
SoftwareIdentity
TargetType = “hp/nss/eva/5000”
MajorVersion = 2;
MinorVersion = 3;
RevisionNumber = 4;
versionString = ” V2.3 (4)"
ElementCapabilities
SoftwareInstallationServiceCapabilities
SupportedSynchronousActions[] = “Install From SoftwareIdentity”
SupportedPackageIDs = { “hp/nss/eva/5000”, “hp/nss/eva/3000” }

A provider for the SoftwareInstallationService will be take the passed SoftwareIdentity reference, obtain the actual instance and then follow the SAPAvailableForElement association to find the URL for the required bits.

Figure 51: "Example Instance Diagram" shows how this might be instantiated with a proxy provider.

Durable Names and Correlatable IDs of the Profile

Software Identity.TargetType is the only correlatable ID introduced by this subprofile. The TargetType parameter is a correlatable identifier that indicates the ‘type’ of SoftwareIdentity. It allows a ‘repository’ to be queried for applicable software/firmware.

The same format shall be used for the Software Repository and for the Software Installation Service so that correlation can be performed.

Since the SoftwareInstallationService may be able to handle multiple TargetTypes, SoftwareInstallationServiceCapabilities includes an array of supported TargetTypes that indicates the types supported by the service.

8.2.1.12.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.12.3 Cascading Considerations

Not defined in this standard.

8.2.1.12.4 Supported Subprofiles and Packages

None.

8.2.1.12.5 Methods of this Profile

The following methods are used by this subprofile:

```
uint32 InstallFromSoftwareIdentity(
        CIM_ConcreteJob REF Job
        CIM_SoftwareIdentity REF Source
        CIM_ManagedElement REF Target
        CIM_SoftwareIdentityCollection REF Collection
)
```

Start a job to install or update a SoftwareIdentity (Source) on a ManagedElement (Target).

In addition the method can be used to add the SoftwareIdentity simultaneously to a specified SofwareIndentityCollection if the SoftwareRepository subprofile is supported. A client may specify either or both of the Collection and Target parameters. The Collection parameter is only supported if the SoftwareRepository subprofile is supported and SoftwareInstallationService.CanAddToCollection is TRUE. It shall be set to NULL otherwise.

If 0 is returned, the function completed successfully and no ConcreteJob instance was required. If 4 096/0x1000 is returned, a ConcreteJob will be started to perform the install. The Job's reference will be returned in the output parameter Job.

```
uint32 CheckSoftwareIdentity(
        CIM_SoftwareIdentity REF Source
        CIM_ManagedElement REF Target
        CIM_SoftwareIdentityCollection REF Collection
        uint16 InstallCharacteristics [ ]
)
```

This method allows a client application to determine whether a specific SoftwareIdentity can be installed (or updated) on a ManagedElement. It also allows other characteristics to be determined such as whether install will require a reboot. In addition a client can check whether the SoftwareIdentity can be added simultaneously to a specified SofwareIndentityCollection. A client may specify either or both of the Collection and Target parameters. The Collection parameter is only supported if the SoftwareRepository subprofile is supported and SoftwareInstallationService.CanAddToCollection is TRUE. It shall be set to NULL otherwise.

The InstallCharacteristics parameter describes the characteristics of this installation/update:

- **Target automatic reset**: The target element will automatically reset once the installation is complete.
- **System automatic reset**: The containing system of the target ManagedElement (normally a logical device or the system itself) will automatically reset/reboot once the installation is complete.
- **Separate target reset required**: EnabledLogicalElement.RequestStateChange needs to be used to reset the target element after the SoftwareIdentity is installed.
- **Separate system reset required:** EnabledLogicalElement.RequestStateChange needs to be used to reset/reboot the containing system of the target ManagedElement after the SoftwareIdentity is installed.

- **Manual Reboot Required:** The system needs to be manually rebooted by the user.
- **No reboot required:** No reboot is required after installation.
- **User Intervention Recommended**: It is recommended that a user confirm installation of this SoftwareIdentity. Inappropriate application may have serious consequences.
- **may be added to specified collection**: The SoftwareIdentity may be added to specified SoftwareIdentityCollection.

8.2.1.12.6 Client Considerations and Recipes

Not defined in this standard.

8.2.1.12.7 Registered Name and Version

Software Installation Service version 1.1.0

8.2.1.12.8 CIM Server Requirements

Table 220: CIM Server Requirements for Software Installation Service

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.12.9 CIM Elements

Table 221: CIM Elements for Software Installation Service

Element Name	Description
Mandatory Classes	
CIM_ElementCapabilities (8.2.1.12.9.1)	
CIM_HostedService (8.2.1.12.9.2)	
CIM_InstalledSoftwareIdentity (8.2.1.12.9.3)	Indicates that a particular software identity
CIM_ServiceAvailableToElement (8.2.1.12.9.4)	Associates ManagedElements that the service can update
CIM_SoftwareIdentity (8.2.1.12.9.5)	Versioning/identity information for a specific software identity
CIM_SoftwareInstallationService (8.2.1.12.9.6)	The service for installing software/firmware.
CIM_SoftwareInstallationServiceCapabilities (8.2.1.12.9.7)	The capabilities of the Software Installation Service
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SoftwareIdentity	Addition of Software Identity
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SoftwareIdentity	Delete SoftwareIdentity

8.2.1.12.9.1 CIM_ElementCapabilities

Class Mandatory: true

Table 222: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Capabilities		CIM_Capabilities	

8.2.1.12.9.2 CIM_HostedService

Class Mandatory: true

Table 223: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_Service	
Antecedent		CIM_System	

8.2.1.12.9.3 CIM_InstalledSoftwareIdentity

Indicates that a particular software identity

Class Mandatory: true

Table 224: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.1.12.9.4 CIM_ServiceAvailableToElement

associates ManagedElements that the service can update
Class Mandatory: true

Table 225: SMI Referenced Properties/Methods for CIM_ServiceAvailableToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UserOfService		CIM_ManagedElement	
ServiceProvided		CIM_Service	

8.2.1.12.9.5 CIM_SoftwareIdentity

Versioning/identity information for a specific software identity
Class Mandatory: true

Table 226: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
TargetType	C	string	
Optional Properties/Methods			
SerialNumber		string	
ReleaseDate		datetime	

8.2.1.12.9.6 CIM_SoftwareInstallationService

The service for installing software/firmware.
Class Mandatory: true

Table 227: SMI Referenced Properties/Methods for CIM_SoftwareInstallationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
ElementName		string	
CheckSoftwareIdentity()			
InstallFromSoftwareIdentity()			

8.2.1.12.9.7 CIM_SoftwareInstallationServiceCapabilities

The capabilities of the Software Installation Service
Class Mandatory: true

Table 228: SMI Referenced Properties/Methods for CIM_SoftwareInstallationServiceCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SupportedAsynchronousActions	N	uint16[]	
SupportedSynchronousActions	N	uint16[]	
SupportedTargetTypes		string[]	
CanAddToCollection		boolean	

8.2.1.12.10 Related Standards

Table 229: Related Standards for Software Installation Service

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.1.13 Software Package

8.2.1.13.1 Description

Software Package models software or firmware installed on a computer system.

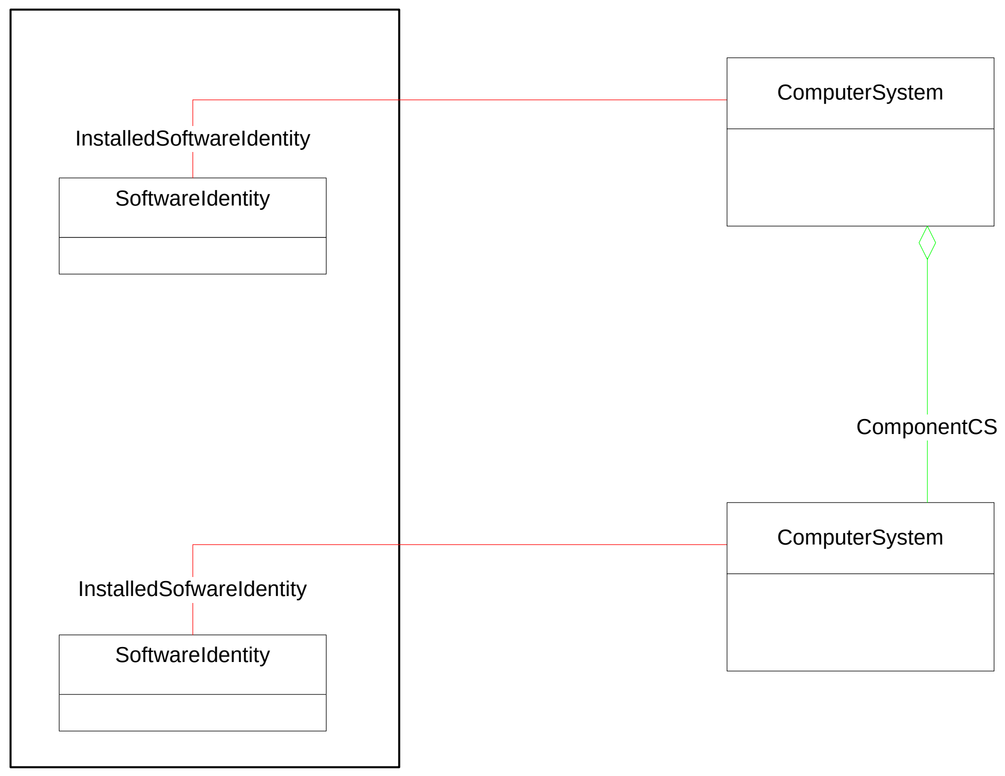
Information on the installed software is given using the SoftwareIdentity class. This is linked to the system using a InstalledSoftwareIdentity association.

Software information may be associated with the “top” level ComputerSystem (if all components are using the same software) or a component ComputerSystem if the software loaded can vary by processor.

Firmware is modeled as SoftwareIdentity. InstalledSoftwareIdentity is used for firmware associated with a System.

Figure 52: "Software Instance Diagram" contains the instance diagram for the Software Package.

Figure 52: Software Instance Diagram



8.2.1.13.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.13.3 Cascading Considerations

Not defined in this standard.

8.2.1.13.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.1.13.5 Methods of this Profile

Not defined in this standard.

8.2.1.13.6 Client Considerations and Recipes

None.

8.2.1.13.7 Registered Name and Version

SoftwarePackage version 1.1.0

8.2.1.13.8 CIM Server Requirements

Table 230: CIM Server Requirements for SoftwarePackage

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.13.9 CIM Elements

Table 231: CIM Elements for SoftwarePackage

Element Name	Description
Mandatory Classes	
CIM_InstalledSoftwareIdentity (8.2.1.13.9.1)	
CIM_SoftwareIdentity (8.2.1.13.9.2)	

8.2.1.13.9.1 CIM_InstalledSoftwareIdentity

Class Mandatory: true

Table 232: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.1.13.9.2 CIM_SoftwareIdentity

Class Mandatory: true

Table 233: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	

Table 233: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
VersionString		string	
Manufacturer		string	
Optional Properties/Methods			
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.1.13.10 Related Standards

Table 234: Related Standards for SoftwarePackage

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.1.14 Software Subprofile

8.2.1.14.1 Description

Information on the installed controller software is given using the SoftwareIdentity class. This is linked to the controller using an InstalledSoftwareIdentity association

See 8.2.1.13, "Software Package" for an instance diagram.

8.2.1.14.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.14.3 Cascading Considerations

Not defined in this standard.

8.2.1.14.4 Supported Subprofiles, and Packages

None.

8.2.1.14.5 Methods of the Profile

None.

8.2.1.14.6 Client Considerations and Recipes

None.

8.2.1.14.7 Registered Name and Version

SoftwarePackage version 1.1.0

8.2.1.14.8 CIM Server Requirements

Table 235: CIM Server Requirements for SoftwarePackage

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.14.9 CIM Elements

Table 236: CIM Elements for SoftwarePackage

Element Name	Description
Mandatory Classes	
CIM_InstalledSoftwareIdentity (8.2.1.14.9.1)	
CIM_SoftwareIdentity (8.2.1.14.9.2)	

8.2.1.14.9.1 CIM_InstalledSoftwareIdentity

Class Mandatory: true

Table 237: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.1.14.9.2 CIM_SoftwareIdentity

Class Mandatory: true

Table 238: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	
Manufacturer		string	
Optional Properties/Methods			
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.1.14.10 Related Standards

Table 239: Related Standards for SoftwarePackage

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

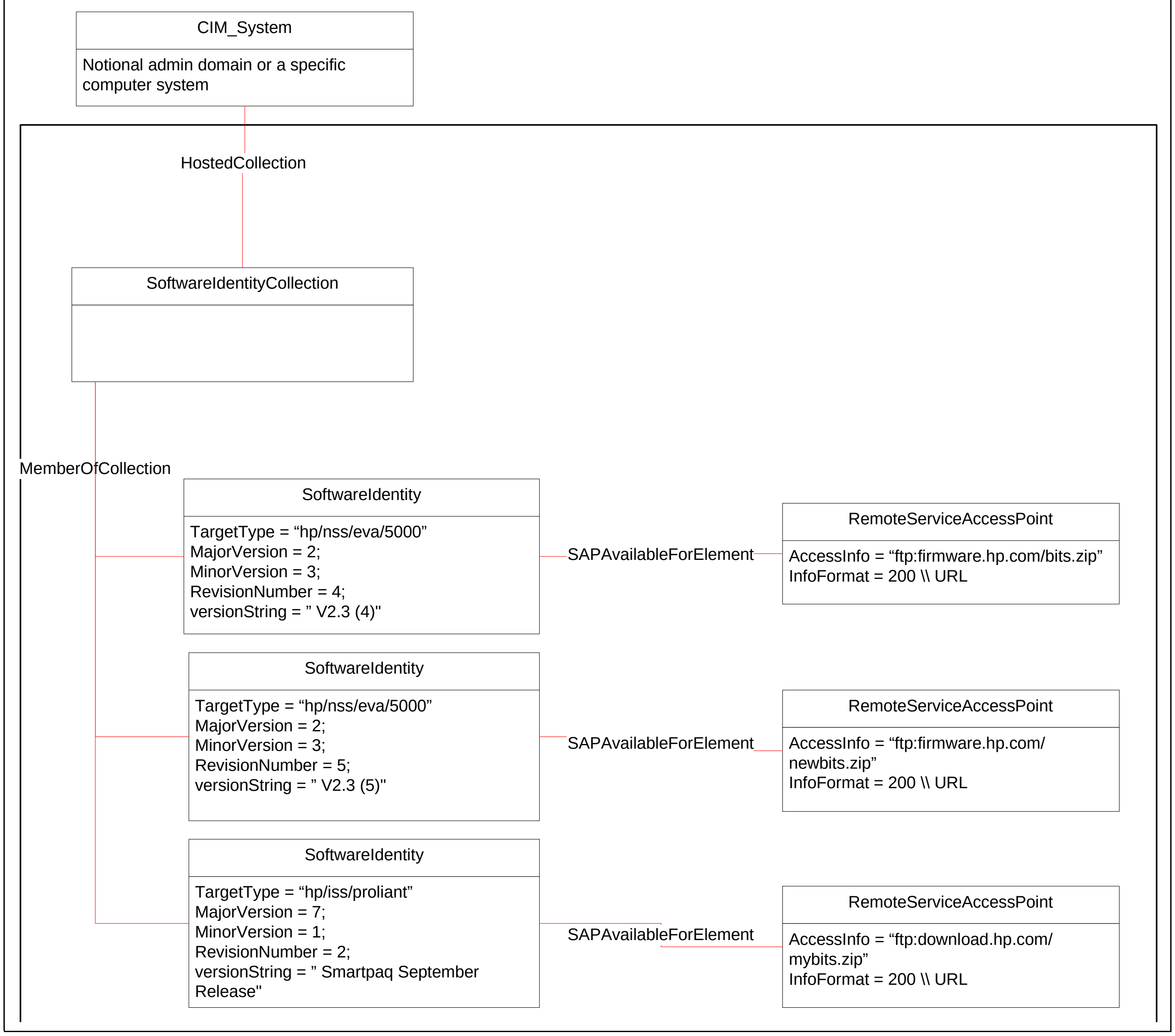
EXPERIMENTAL

8.2.1.15 Software Repository Subprofile

8.2.1.15.1 Description

This profile provides the ability to expose a collection of SoftwareIdentity instances representing software installation packages that can be used in conjunction with the 8.2.1.12, "Software Installation Service Subprofile". These two profiles form a 'pair' that can be used together within a single system or independently on different unaware systems. The different use cases covered are shown in Figure 53: "Software Repository Instance Diagram".

Figure 53: Software Repository Instance Diagram



A typical implementation of a representation would consist of multiple SoftwareIdentitys representing potential upgrades associated by MemberOfCollection to an instance of a SoftwareIdentityCollection which represents the collection itself. The 'location' of the bits needed to install a specific

SoftwareIdentity are represented as RemoteServiceAccessPoint instances one per URL) associated to the SoftwareIdentity by SAPAvailableForElement.

<u>Durable Names and Correlatable IDs of the Profile</u>

Software Identity.TargetType is the only correlatable ID introduced by this subprofile. The TargetType parameter is a correlatable identifier that indicates the ‘type’ of SoftwareIdentity. It allows a ‘repository’ to be queried for applicable software/firmware.

The same format shall be used for the Software Repository and for the Software Installation Service so that correlation can be performed.

Since the SoftwareInstallationService may be able to handle multiple TargetTypes, SoftwareInstallationServiceCapabilities includes an array of supported TargetTypes that indicates the types supported by the service.

8.2.1.15.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.1.15.3 Cascading Considerations

Not defined in this standard.

8.2.1.15.4 Methods of the Profile

None.

8.2.1.15.5 Supported Subprofiles, and Packages

None.

8.2.1.15.6 Client Considerations and Recipes

None.

8.2.1.15.7 Registered Name and Version

Software Repository version 1.1.0

8.2.1.15.8 CIM Server Requirements

Table 240: CIM Server Requirements for Software Repository

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.1.15.9 CIM Elements

Table 241: CIM Elements for Software Repository

Element Name	Description
Mandatory Classes	
CIM_HostedCollection (8.2.1.15.9.1)	The SoftwareIdentityCollection is scoped to a system.
CIM_MemberOfCollection (8.2.1.15.9.2)	Associates SoftwareIdentities to the collection
CIM_RemoteServiceAccessPoint (8.2.1.15.9.3)	Used to express the location of the 'bits' for a software update as an URL
CIM_SAPAvailableForElement (8.2.1.15.9.4)	Links one or more URLs to a SoftwareIdentity.
CIM_SoftwareIdentity (8.2.1.15.9.5)	The information for an available software/firmware update
CIM_SoftwareIdentityCollection (8.2.1.15.9.6)	A collection of SoftwareIdentities that forms the repository
CIM_System (8.2.1.15.9.7)	Represents the system hosting the Software Repository.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SoftwareIdentity	Addition of Software Identity
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SoftwareIdentity	Delete SoftwareIdentity

8.2.1.15.9.1 CIM_HostedCollection

The SoftwareIdentityCollection is scoped to a system.
Class Mandatory: true

Table 242: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_SystemSpecificCollection	

8.2.1.15.9.2 CIM_MemberOfCollection

Associates SoftwareIdentities to the collection
Class Mandatory: true

Table 243: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	
Member		CIM_ManagedElement	

8.2.1.15.9.3 CIM_RemoteServiceAccessPoint

Used to express the location of the 'bits' for a software update as an URL

Class Mandatory: true

Table 244: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
ElementName		string	
AccessInfo		string	
InfoFormat		uint16	

8.2.1.15.9.4 CIM_SAPAvailableForElement

Links one or more URLs to a SoftwareIdentity.
Class Mandatory: true

Table 245: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AvailableSAP		CIM_ServiceAccessPoint	
ManagedElement		CIM_ManagedElement	

8.2.1.15.9.5 CIM_SoftwareIdentity

The information for an available software/firmware update
Class Mandatory: true

Table 246: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
TargetType	C	string	
Optional Properties/Methods			
SerialNumber		string	
ReleaseDate		datetime	

8.2.1.15.9.6 CIM_SoftwareIdentityCollection

A collection of SoftwareIdentities that forms the repository
Class Mandatory: true

Table 247: SMI Referenced Properties/Methods for CIM_SoftwareIdentityCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Optional Properties/Methods			
ElementName		string	

8.2.1.15.9.7 CIM_System

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 248: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name		string	System hosting the Software Repository

8.2.1.15.10 Related Standards

Table 249: Related Standards for Software Repository

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.2 Common Target Port Subprofiles Overview

Target Port Subprofiles model the generic SCSI capabilities and transport-specific aspects of target storage systems such as disk arrays and tape libraries. Separate subprofiles are defined for Fibre Channel, iSCSI, and other transports; but some aspects of these subprofiles follow a common pattern.

SCSIProtocolEndpoint represents SCSI as a protocol, independent of specific transports or device types – i.e., the behavior described in the SCSI Primary Commands (SPC) and SCSI Architecture Model (SAM) specifications from T10. Separating the port from the protocol allows the same type of port to be used with non-SCSI protocols such as IP. SCSIProtocolEndpoint.Role indicates whether this protocol endpoint instance represents a SCSI initiator or target.

Various subclasses of a LogicalPort (e.g., FCPort, EthernetPort) represent the logical aspects of ports, independent from SCSI protocol. SCSIProtocolEndpoint and LogicalPort are associated with DeviceSAPImplementation. For most transports, SCSI protocol is implemented in the port hardware so there is 1-1 cardinality between the LogicalPort and SCSIProtocolEndpoint instances. iSCSI is an exception, many-to-many relationships are possible between EthernetPort and iSCSIProtocolController instances.

A property on LogicalPort called UsageRestriction is indicates whether the port is restricted to use as a “front end” (target) or a “back end” (initiator) interface or both. Note that port may not have a restriction and the actual point-in-time role is modeled in SCISProtocolEndpoint.Role.

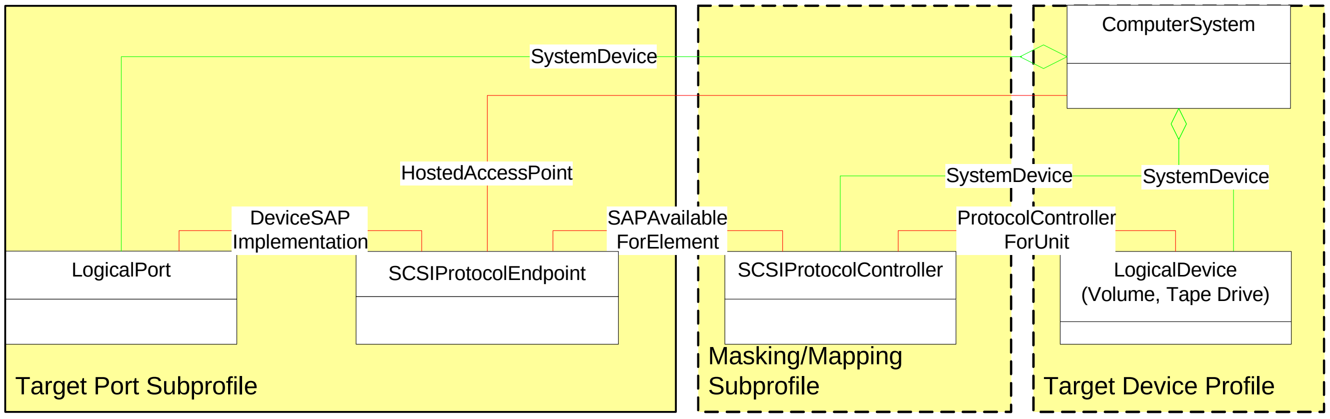
SCSIProtocolController represents the SCSI ‘view’ of ports and logical devices seen by SCSI initiators. In a system supporting LUN Masking, zero or more views exist; defined by the customer to expose subsets of logical units to certain initiators. SAPAvailableForElement connects SCSI ProtocolEndpoint from a target port subprofile to SCSIProtocolController instances from the Masking/Mapping subprofile.

The LogicalDevice object represents SCSI logical units that are visible to external systems. It is subclassed to StorageVolume, TapeDrive, etc. to identify the device type.

LogicalPort instances are associated to a ComputerSystem instance from the target profile. If the port is available to multiple controllers, this would be the top-level ComputerSystem. If the port is vulnerable to failover when a controller fails, it should be associated to non-top-level ComputerSystem that represents that controller. HostedAccessPoint associates SCSIProtocolEndpoint to a ComputerSystem from the target profile (for example, Array Profile), the same ComputerSystem that is associated to the protocol endpoint’s associated port. Other transport-specific classes are defined in the target port subprofiles that follow.

Figure 54: "Generic Storage Target" depicts a generic storage device with elements from a target port subprofile, the Masking/Mapping subprofile, and a target device subprofile.

Figure 54: Generic Storage Target



Modeling SCSI Logical Units

The SCSI standard inquiry response includes a Device Type property with integers representing types of devices. Most of these devices types have a CIM analog. Devices that are used primarily for management are modeled as ArbitraryLogicalUnit. ArbitraryLogicalUnit.DeviceType maps to SCSI device types. Table 250 describes how common storage devices are modeled in CIM.

Table 250: How Common storage devices are modeled in CIM

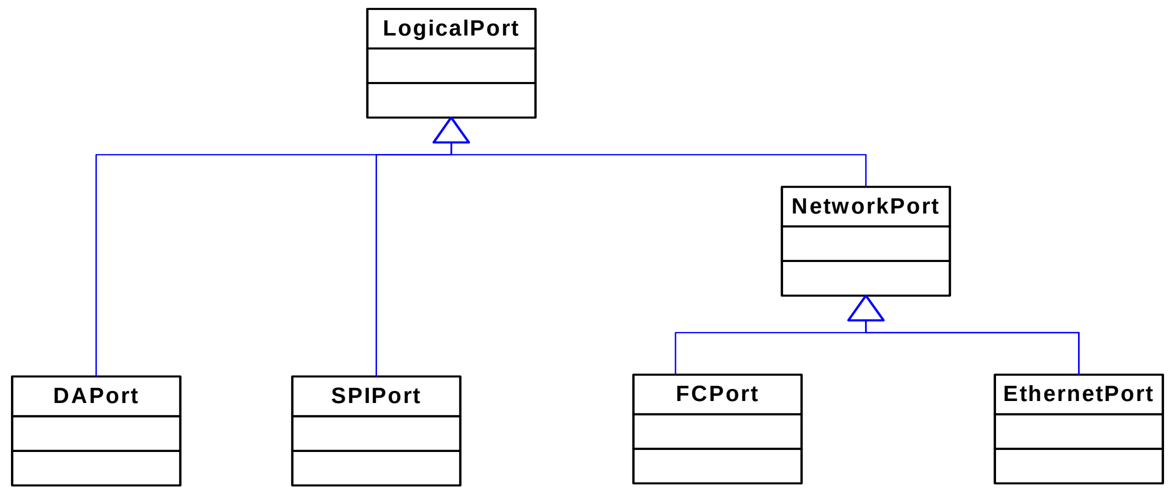
SCSI Device Type	Inquiry Device Type	LogicalDevice
DirectAccessDevice(00)	0	DiskDrive or StorageVolume
SequentialAccessDevice(01)	1	TapeDrive
WriteOnceDevice(04)	4	WormDrive
CD-ROM(05)	5	CDROMDrive
MediaChanger(08)	8	MediaTranferDevice
ArrayController(0C)	0xc	SCSIArbitraryLogicalUnit DeviceType=”SCSI SCC Device”
SES(0D)	0xd	SCSIArbitraryLogicalUnit DeviceType=”SCSI SES”
Other		SCSIArbitraryLogicalUnit DeviceType=”Other”
Unknown		SCSIArbitraryLogicalUnit DeviceType=”Uknown”

All devices (SCSI logical units) visible to external systems shall be modeled.

Types of Ports

LogicalPort and SCSIProtocolEndpoint work together to model the external connection. LogicalPort is subclassed to signify the type of transport. If the port is subclassed directly from LogicalPort it indicates it is connected to a bus. If the port is further subclassed from NetworkPort it indicates the port is capable of being used in a network. Details for each port type are in the following sections.

Figure 55: Port Class Hierarchy



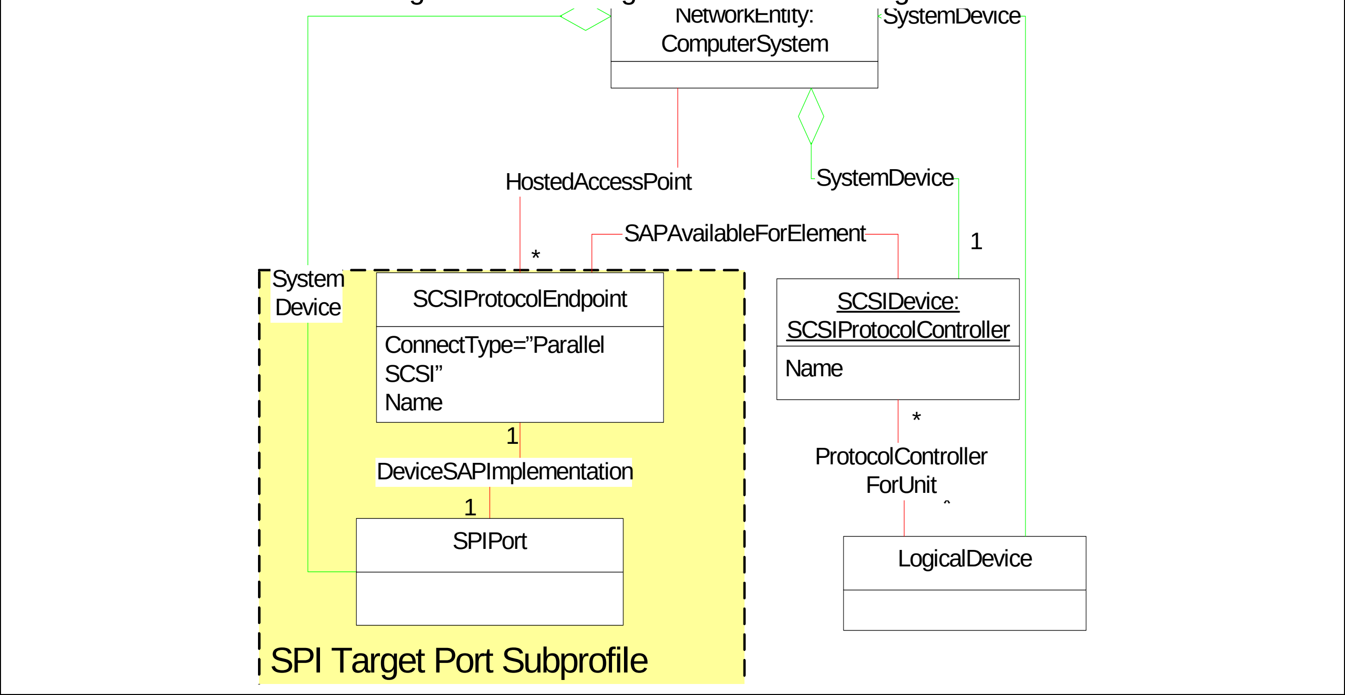
8.2.2.1 Parallel SCSI (SPI) Target Ports Subprofile

8.2.2.1.1 Description

This port represents a classical SCSI Parallel Interface (SPI).

Because of addressing limits, the port may use multiple SCSI IDs to extend the addressing. The LUN Mapping/Masking common subprofile is not used with this port type.

Figure 56: SPI Target Port Instance Diagram



The SCSIProtocolEndpoint.ConnectionType shall be set to “Parallel SCSI”. The SCSIProtocolEndpoint class is connected to a SPIPort. Attributes of SPIPort define the bus width and speed. The port class inherits the UsageRestriction attribute from LogicalPort. This attribute shall be set to “Front-end only”

8.2.2.1.2 Durable Names and Correlatable IDs of the Subprofile

None

8.2.2.1.3 Health and Fault Management

Table 251: SPIPort OperationalStatus

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

8.2.2.1.4 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.2.1.5 Extrinsic Methods of this Subprofile

None

8.2.2.1.6 Client Considerations and Recipes

None

8.2.2.1.7 Registered Name and Version

SPI Target Ports version 1.1.0

8.2.2.1.8 CIM Server Requirements

Table 252: CIM Server Requirements for SPI Target Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.2.1.9 CIM Elements

Table 253: CIM Elements for SPI Target Ports

Element Name	Description
Mandatory Classes	
CIM_DeviceSAPImplementation (8.2.2.1.9.1)	
CIM_HostedAccessPoint (8.2.2.1.9.2)	
CIM_SCSIProtocolEndpoint (8.2.2.1.9.3)	
CIM_SPIPort (8.2.2.1.9.4)	
CIM_SystemDevice (8.2.2.1.9.5)	

8.2.2.1.9.1 CIM_DeviceSAPImplementation

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 254: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.2.1.9.2 CIM_HostedAccessPoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 255: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.2.1.9.3 CIM_SCSIProtocolEndpoint

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 256: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
OtherTypeDescription		string	
ConnectionType		uint16	Shall be 3 (Parallel SCSI)
Role		uint16	Shall be 3 (Target) or 4 (Both Initiator and Target)

8.2.2.1.9.4 CIM_SPIPort

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 257: SMI Referenced Properties/Methods for CIM_SPIPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	

Table 257: SMI Referenced Properties/Methods for CIM_SPIPort

Property	Flags	Type	Description & Notes
UsageRestriction		uint16	Shall be 2 to indicate this is a front end port only or 4 to indicate usage is not restricted.

8.2.2.1.9.5 CIM_SystemDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 258: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.2.1.10 Related Standards

Table 259: Related Standards for SPI Target Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.2.2 FC Target Port Subprofile

8.2.2.2.1 Description

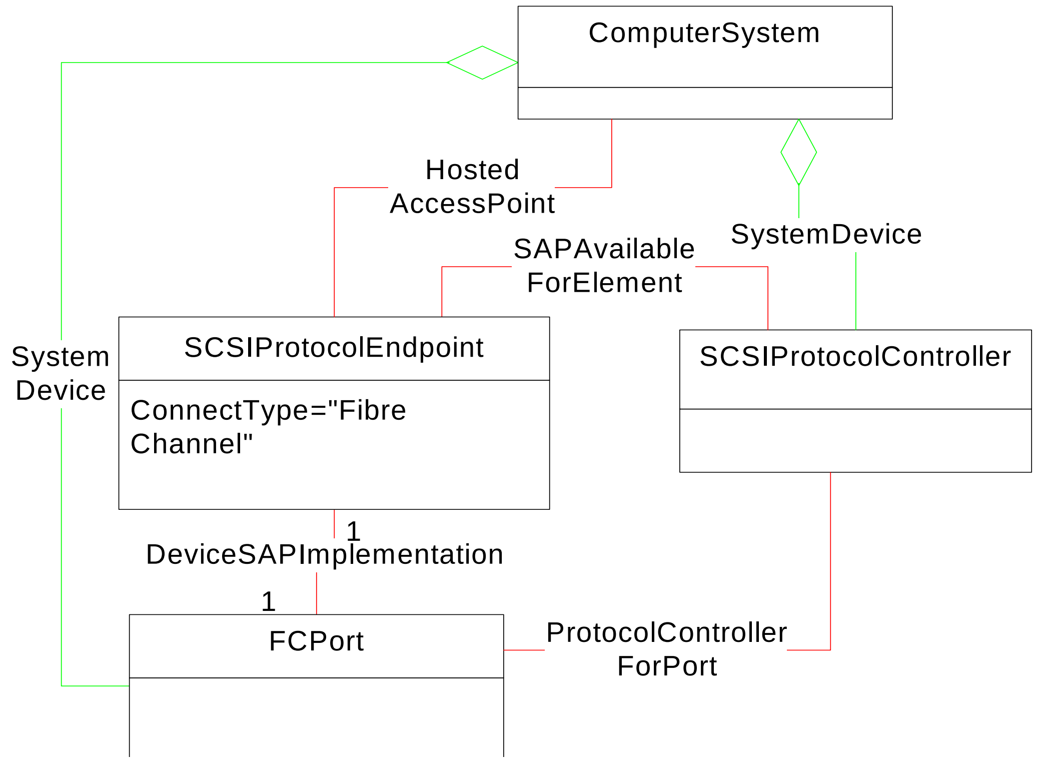
The FC Target Port Subprofile models the Fibre Channel specific aspects of a target storage system.

For Fibre Channel ports, the concrete subclass of LogicalPort is FCPort. FCPort is always associated 1-1 with a SCSIProtocolEndpoint instance.

IS24775-2006, *Storage Management* Backwards Compatibility

SCSIProtocolEndpoint was introduced in this version ofSMI-S to enable support for non-FC transports and for non-SCSI protocols. In IS24775-2006, *Storage Management* (SMI-S 1.0), FCPort was associated directly to SCSIProtocolController. SCSIProtocolEndpoint, DeviceSAPImplementation, and SAPAvailableForElement are required and are used consistently across all target port subprofiles. To maintain backwards compatibility, ProtocolControllerForPort is still required in this version of SMI-S. But this association will be removed in a future versions and clients should start using the newer model.

Figure 57: FC Target Port Instance Diagram



8.2.2.2.2 Durable Names and Correlatable IDs of the Subprofile

FCPort.PermanantAddress shall contain the port’s Port WWN.

8.2.2.2.3 Health and Fault Management

Table 260: FCPort OperationalStatus

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

8.2.2.2.4 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.2.2.5 Extrinsic Methods of this Subprofile

None

8.2.2.2.6 Client Considerations and Recipes

None

8.2.2.2.7 Registered Name and Version

FC Target Ports version 1.1.0

8.2.2.2.8 CIM Server Requirements

Table 261: CIM Server Requirements for FC Target Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.2.2.9 CIM Elements

Table 262: CIM Elements for FC Target Ports

Element Name	Description
Mandatory Classes	
CIM_DeviceSAPImplementation (8.2.2.2.9.1)	
CIM_FCPort (8.2.2.2.9.2)	
CIM_HostedAccessPoint (8.2.2.2.9.3)	
CIM_SAPAvailableForElement (8.2.2.2.9.5)	
CIM_SCSIProtocolEndpoint (8.2.2.2.9.6)	
CIM_SystemDevice (8.2.2.2.9.7)	
Optional Classes	
CIM_ProtocolControllerForPort (8.2.2.2.9.4)	Only required if the instrumentation claims compatibility with 1.0
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort	Create FCPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Change to FCPort OperationalStatus
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.CIM_FCPort::OperationalStatus <> PreviousInstance.CIM_FCPort::OperationalStatus	CQL - Change to FCPort OperationalStatus
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.Speed <> PreviousInstance.Speed	Change to FCPort properties
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.CIM_FCPort::NetworkAddresses <> PreviousInstance.CIM_FCPort::NetworkAddresses	CQL - Change to FCPort properties
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort	Delete FCPort

8.2.2.2.9.1 CIM_DeviceSAPImplementation

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 263: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.2.2.9.2 CIM_FCPort

Created By : External
Modified By : External
Deleted By : External
Standard Names: The PermanentAddress Property follows the requirements in 6.2.4.5.2
Class Mandatory: true

Table 264: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
PermanentAddress	CD	string	Port WWN
OperationalStatus		uint16[]	
UsageRestriction		uint16	Shall be 2 to indicate this is a front end port only or 4 to indicate usage is not restricted.
Optional Properties/Methods			
SupportedCOS		uint16[]	
ActiveCOS		uint16[]	
SupportedFC4Types		uint16[]	
ActiveFC4Types		uint16[]	
PortType		uint16	

8.2.2.2.9.3 CIM_HostedAccessPoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 265: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.2.2.9.4 CIM_ProtocolControllerForPort

Only required if the instrumentation claims compatibility with 1.0
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: false

Table 266: SMI Referenced Properties/Methods for CIM_ProtocolControllerForPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolController	
Dependent		CIM_LogicalPort	

8.2.2.2.9.5 CIM_SAPAvailableForElement

Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 267: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AvailableSAP		CIM_ServiceAccessPoint	The SCSIProtocolEndpoint.
ManagedElement		CIM_ManagedElement	The SCSIProtocolController.

8.2.2.2.9.6 CIM_SCSIProtocolEndpoint

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 268: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
OtherTypeDescription		string	
ConnectionType		uint16	Shall be 2 (Fibre Channel)
Role		uint16	Shall be 3 (Target) or 4 (Both Initiator and Target)

8.2.2.2.9.7 CIM_SystemDevice

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

Class Mandatory: true

Table 269: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.2.2.10 Related Standards

Table 270: Related Standards for FC Target Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.2.3 iSCSI Target Ports Subprofile

8.2.2.3.1 Description

The iSCSI target ports subprofile describes the iSCSI specific aspects of target device. iSCSI terminology is different than that used in other parts of SMI-S. Table 58 uses the UML instance naming notation (InstanceName:ClassName) with the iSCSI-style names before the CIM names. Table 58 explains the use of all these objects.

Note that ComputerSystem, SCSIProtocolController and StorageVolume are not actually part of this subprofile; they would be the parts of the Array Profile that associate with the iSCSI-specific classes. iSCSI does have a specific naming requirement for SCSIProtocolController that is described below.

Figure 58: iSCSI Target Ports Subprofile Instance Diagram

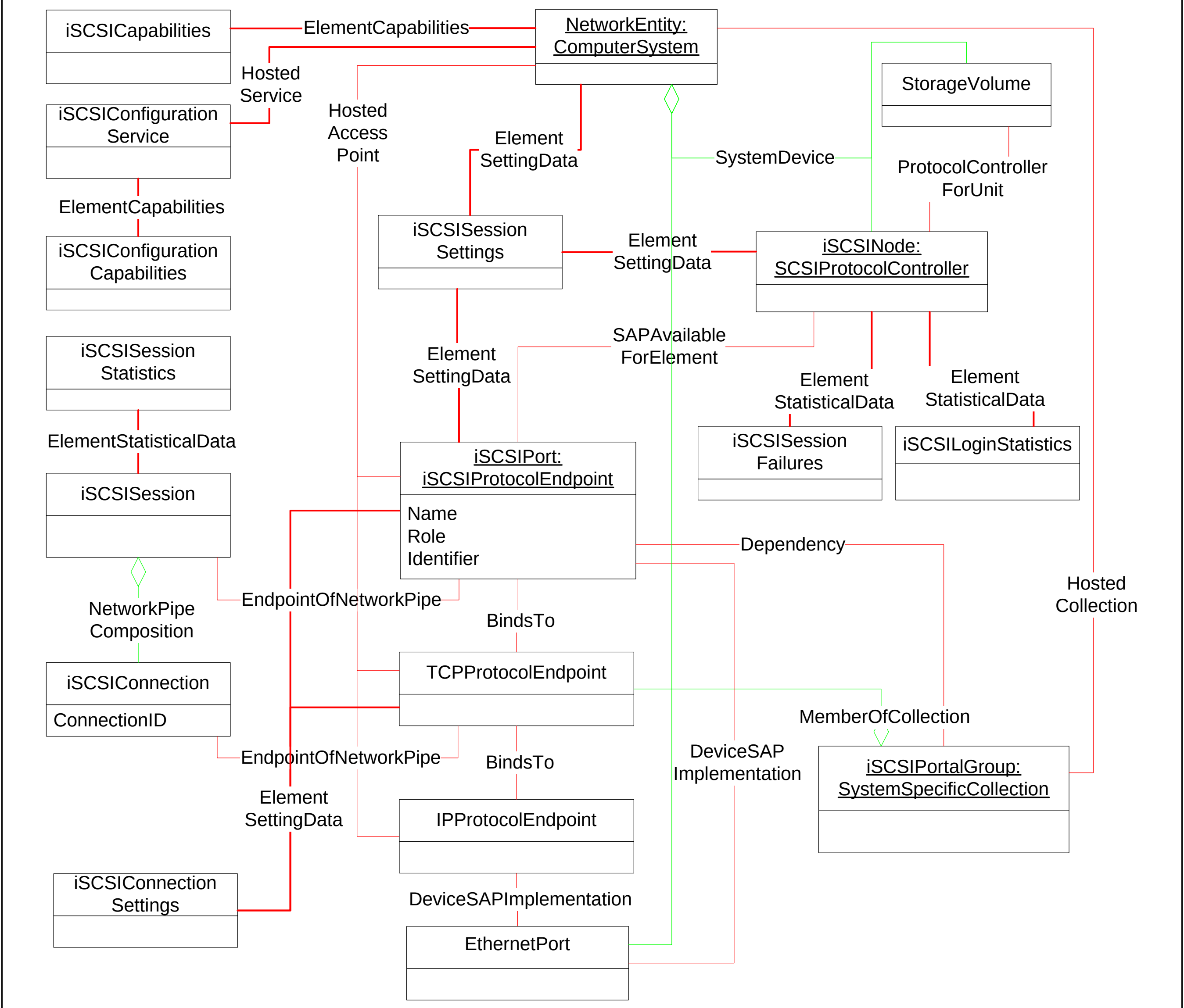


Table 271: iSCSI Terminology and SMI-S Class Names

iSCSI Term	CIM Class Name	Notes
Network Entity	ComputerSystem	The Network Entity represents a device or gateway that is accessible from the IP network. A Network Entity shall have one or more Network Portals, each of which can be used to gain access to the IP network by some iSCSI Nodes contained in that Network Entity.
Session	iSCSISession	The group of TCP connections that link an initiator with a target form a session (loosely equivalent to a SCSI I-T nexus). TCP connections can be added and removed from a session. Across all connections within a session, an initiator sees one and the same target.
Connection	NetworkPipe	A connection is a TCP connection. Communication between the initiator and target occurs over one or more TCP connections. The TCP connections carry control messages, SCSI commands, parameters, and data within iSCSI Protocol Data Units (iSCSI PDUs).
SCSI Port	iSCSIProtocolEndpoint	A SCSI Port using an iSCSI service delivery subsystem. A collection of Network Portals that together act as a SCSI initiator or target.
Portal Group	SystemSpecificCollection	iSCSI supports multiple connections within the same session; some implementations will have the ability to combine connections in a session across multiple Network Portals. A Portal Group defines a set of Network Portals within an iSCSI Network Entity that collectively supports the capability of coordinating a session with connections spanning these portals. Not all Network Portals within a Portal Group need participate in every session connected through that Portal Group. One or more Portal Groups may provide access to an iSCSI Node. Each Network Portal, as utilized by a given iSCSI Node, belongs to exactly one portal group within that node.
Network Portal	TCPProtocolEndpoint, IPProtocolEndpoint, EthernetPort	The Network Portal is a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. A Network Portal in an initiator is identified by its IP address. A Network Portal in a target is identified by its IP address and its listening TCP port.
Node	SCSIProtocolController	The iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals. An iSCSI Node is identified by its iSCSI Name. The separation of the iSCSI Name from the addresses used by and for the iSCSI Node allows multiple iSCSI nodes to use the same address, and the same iSCSI node to use multiple addresses.

Mapping and Masking Considerations

The class SCSIProtocolController is used in the Mapping and Masking subprofile to model a “view”, which is a set of logical devices exposed to an initiator. It is in a sense a virtual SCSI device, but carries no SCSI device name when used with the other Target Ports subprofiles such as the FC Target Port subprofile. In fact the class is even not part of these sub-profiles.

The iSCSI Target Ports subprofile however uses SCSIProtocolController to model the iSCSI Node which is the SCSI Device as defined in the SAM specification. It has a SCSI device name which is the iSCSI Node Name. Thus the presence of instances of SCSIProtocolController with this subprofile has multiple meanings. Whereas there may be no instances of SCSIProtocolController with other Target Port subprofiles until created as views by the Mapping and Masking method ExposePaths, instances of SCSIProtocolControllers as iSCSINodes can be brought into existence by the iSCSI method CreateiSCSINode. The instances can then be used as inputs to ExposePaths to grant access by Initiators to logical devices through the Node. This initial SCSIProtocolController that was created as a Node will be the first view. Additional “view” ProtocolControllers created by ExposePaths would carry the same iSCSI Node name to convey that they represent the same underlying Node.

Settings

An iSCSI Session is established between an Initiator Port and a Target Port through the establishment of an initial iSCSI Connection, which happens during the “Leading” Login. At this time the operational properties for the Session are negotiated and also the operational properties for the initial Connection. Additional Connections for the Session are established through subsequent logins. For many operational properties both the Initiator and Target have settings that specify the starting position for the negotiation process. The settings for negotiating Session-wide operational properties (found in iSCSISession) are in iSCSISessionSettings. Likewise the settings for negotiating Connection level operational properties (found in iSCSI Connection) are in iSCSIConnectionSettings. For example, iSCSISessionSettings contains the property MaxConnectionsPerSession, which is the value that the local system (which in this sub-profile is the Target) would like to use for Session. When the leading login is complete the actual value agreed upon with the Initiator is in the property MaxConnectionsPerSession in iSCSI Session.

Different implementations may scope the settings classes differently.

iSCSISessionSettings can be associated to any one of the following classes:

iSCSIProtocolEndpoint: The Settings apply to Sessions created on the iSCSI Port represented by the iSCSIProtocolEndpoint.

SCSIProtocolController: The Settings apply to Sessions created on all iSCSIProtocolEndpoint belonging to the iSCSI Node represented by the SCSIProtocolController.

ComputerSystem: The Settings apply to Sessions created on all iSCSIProtocolEndpoints belonging to all SCSIProtocolControllers belonging to the ComputerSystem.

iSCSIConnectionSettings can be associated to any one of the following classes:

TCPProtocolEndpoint: The Settings apply to each Connection created using the Network Portal represented by the TCPProtocolEndpoint, regardless of which iSCSIProtocolEndpoint owns the Session that the Connection belongs to.

iSCSIProtocolEndpoint: The Settings apply to Connections using NetworkPortals to which the iSCSIProtocolEndpoint is bound and belonging to Sessions on that same iSCSIProtocolEndpoint.

Durable Names and Correlatable IDs of the Subprofile

The Name property for the iSCSI node (SCSIProtocolController) shall be a compliant iSCSI name as described in 6.2.4.9, "iSCSI Names" and NameFormat shall be set to “iSCSI Name”.

The Name property for iSCSIProtocolEndpoint shall be a compliant iSCSI name as described in 6.2.4.9, "iSCSI Names" and ConnectionType shall be set to “iSCSI”.

8.2.2.3.2 Health and Fault Management

Table 272: EthernetPort OperationalStatus

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

8.2.2.3.3 Supported Subprofiles and Packages

None

8.2.2.3.4 Methods of this Subprofile

The iSCSIConfigurationService provides the following methods that allow a client to manipulate iSCSIProtocolEndpoints in an iSCSI Target Node. The class iSCSIProtocolController models the iSCSI Target Port. The instance of the service is scoped by an instance of ComputerSystem that represents that Network Entity. The capabilities of this service are defined in the companion class iSCSIConfigurationCapabilities.

8.2.2.3.4.1 CreateiSCSINode

This method creates an iSCSI Node in the form of an instance of SCSIProtocolController. As part of the creation process a SystemDevice association is created between the new SCSIProtocolController and the scoping Network Entity (ComputerSystem) hosting this service.

CreateiSCSINode

IN, string **Alias**,

The iSCSI Alias for the new Node.

OUT, SCSIProtocolController REF **iSCSINode,**

A reference to the new SCSIProtocolController that is created.

8.2.2.3.4.1.1 Return Values

Success

Not Supported

Unspecified Error

Timeout

Failed

Node Creation Not Supported

Alias in use by Other Node

8.2.2.3.4.1.2 Created Instances

SCSIProtocolController

SystemDevice

8.2.2.3.4.1.3 Deleted Instances

None

8.2.2.3.4.1.4 Modified Instances

None

8.2.2.3.4.2 DeleteiSCSINode

The method deletes an instance of SCSIProtocolController representing an iSCSI Node and all associations in which this SCSIProtocolController is referenced. If Sessions are active on iSCSIProtocolEndpoints belonging to this Node an error will be returned. If no Sessions are active the scoped iSCSIProtocolEndpoints will be deleted.

DeleteiSCSINode

IN, SCSIProtocolController REF **iSCSINode**

The SCSIProtocolController to be deleted.

8.2.2.3.4.2.1 Return Values

Success

Not Supported

Unspecified Error

Timeout

Failed

Invalid Parameter

SCSIProtocolController Non-existent

Sessions Active on Node Ports

8.2.2.3.4.2.2 Created Instances

None

8.2.2.3.4.2.3 Deleted Instances

SCSIProtocolController

SystemDevice

iSCSIProtocolEndpoint

HostedAccessPoint

SAPAvailableForElement

BindsTo

8.2.2.3.4.2.4 Modified Instances

None

8.2.2.3.4.3 CreateiSCSIProtocolEndpoint

This method creates an iSCSI Port in the form of an instance of iSCSIProtocolEndpoint. As part of the creation process the iSCSIProtocolEndpoint is 'bound to' the underlying TCPProtocolEndpoints which are specified as inputs by creating instances of the BindsTo association between the new instance and those instances. In addition, an instance of SAPAvailableForElement is created between the specified SCSIProtocolController and the new instance of iSCSIProtocolEndpoint.

CreateiSCSIProtocolEndpoint

IN, SCSIProtocolController REF **iSCSINode**,

The SCSIProtocolController instance representing the iSCSI Node that will contain the iSCSI Port.

IN, uint16 **Role**,

For iSCSI, each iSCSIProtocolEndpoint acts as either a target or an initiator endpoint.This property indicates which role this iSCSIProtocolEndpoint implements.

IN, string **Identifier**,

The Identifier shall contain the Target Portal Group Tag (TGPT). Each iSCSIProtocolEndpoint (iSCSI port) associated to a common SCSIProtocolController (iSCSI node) has a unique Identifier. This field is a string that contains 12 hexadecimal digits. If the property IdentifierSelectionSupported in class iSCSIConfigurationCapabilities is false, this parameter shall be set to NULL.

IN, ProtocolEndpoint REF **NetworkPortals[],**

An Array of References to TCPProtocolEndpoints representing Target Network Portals. The TCPProtocolEndpoints specified each shall be associated to an instance of IPProtocolEndpoint via a BindsTo association in order to provide the Target Network Portal functionality. The selected Portal endpoints shall be from the same SystemSpecificCollection, which represents a Portal Group.

OUT, iSCSIProtocolEndpoint REF **iSCSIPort,**

A reference to the new iSCSIProtocolEndpoint that is created.

8.2.2.3.4.3.1 Return Values

Success

Not Supported

Unspecified Error

Timeout

Failed

SCSIProtocolController Non-existent

Role Not Supported By Specified SCSIProtocolController

Identifier In Use, Not Unique

Identifier Selection Not Supported

ProtocolEndpoint Non-Existent

TCPProtocolEndpoint Not Bound To Underlying IPProtocolEndpoint

TCPProtocolEndpoint In Use By Other iSCSIProtocolEndpoint In Same Target SCSIProtocolController.

ProtocolEndpoints Not From Same Endpoint Collection

8.2.2.3.4.3.2 Created Instances

iSCSIProtocolEndpoint

HostedAccessPoint

SAPAvailableForElement

BindsTo

8.2.2.3.4.3.3 Deleted Instances

None

8.2.2.3.4.3.4 Modified Instances

None

8.2.2.3.4.4 DeleteiSCSIProtocolEndpoint

The method deletes an instance of iSCSIProtocolEndpoint and all associations in which this iSCSIProtocolEndpoint is referenced.

DeleteiSCSIProtocolEndpoint

IN, iSCSIProtocolEndpoint REF **iSCSIPort**

The iSCSIProtocolEndpoint to be deleted.

8.2.2.3.4.4.1 Return Values

Success

Not Supported

Unspecified Error

Timeout

Failed

Invalid Parameter

Endpoint Non-existent

8.2.2.3.4.4.2 Created Instances

None

8.2.2.3.4.4.3 Deleted Instances

iSCSIProtocolEndpoint

HostedAccessPoint

SAPAvailableForElement

BindsTo

8.2.2.3.4.4.4 Modified Instances

None

8.2.2.3.4.5 BindiSCSIProtocolEndpoint

This method provides for modification of an existing iSCSI Port by associating a TCPProtocolEndpoint representing a Target Network Portal to the iSCSIProtocolEndpoint. The association is persisted as an instance of BindsTo. The selected Portal endpoint shall be from the same SystemSpecificCollection, which represents a Portal Group, as those endpoints currently bound to the iSCSIProtocolEndpoint.

This action is intended to be reversed by the use of the intrinsic method **'DeleteInstance'**.

BindiSCSIProtocolEndPoint

IN, iSCSIProtocolEndpoint REF **iSCSIPort**,

A reference to the iSCSIProtocolEndpoint

IN, ProtocolEndpoint REF **NetworkPortal**

An instance of TCPProtocolEndpoint representing the Network Portal to be added

8.2.2.3.4.5.1 Return Values

Success

Not Supported

Unspecified Error

Timeout

Failed

Invalid Parameter

ProtocolEndpoint Non-Existent

TCPProtocolEndpoint Not Bound To Underlying IPProtocolEndpoint

ProtocolEndpoint In Use By Other iSCSIProtocolEndpoint In Same Target SCSIProtocolController

ProtocolEndpoint Not From Same Endpoint Collection

8.2.2.3.4.5.2 Created Instances

BindsTo

8.2.2.3.4.5.3 Deleted Instances

None

8.2.2.3.4.5.4 Modified Instances

None

8.2.2.3.5 Client Considerations and Recipes

8.2.2.3.5.1 Discover the iSCSI Target Port capabilities.

```
// DESCRIPTION
// Discover the iSCSI Target Port capabilities.
//
```

```
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The ComputerSystem representing the target system of interest has been
// previously identified and defined in the $NetworkEntity-> variable.

// MAIN
// Step 1. Locate the instance of CIM_iSCSICapabilities associated to the
// target ComputerSystem.
$iSCSICapabilities[] = Associators($NetworkEntity->,
     “CIM_ElementCapabilities”,
     “CIM_iSCSICapabilities”,
     “ManagedElement”,
     “Capabilities”,
     {“MinimumSpecificationVersionSupported”,
     “MaximumSpecificationVersionSupported”,
     “AuthenticationMethodsSupported”})

if ($iSCSICapabilities[] == null || $iSCSICapabilities[].length != 1) {
    <ERROR! The iSCSI capabilities could not be found>
}
$Capabilities = $iSCSICapabilities[0]
```

8.2.2.3.5.2 Identify the iSCSI Nodes in a target system.

Step 1. Look for instances of SCSIProtocolController with NameFormat=”iSCSI Name”.

```
// DESCRIPTION
//
// Identify the iSCSI Nodes in a target system.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The ComputerSystem representing the Network Entity of interest has been
// previously identified and defined in the $NetworkEntity-> variable.

// MAIN
// Step 1. Locate the instances of CIM_SCSIProtocolController with a NameFormat
// property value of “iSCSI Name”.
$ProtocolControllers[] = Associators($NetworkEntity->,
     “CIM_SystemDevice”,
     “CIM_SCSIProtocolController”,
     “GroupComponent”,
     “PartComponent”,
     false,
     false,
     {“Name”, “NameFormat”})

// Step 2. Locate the SCSIProtocolControllers that represent the iSCSI Nodes.
$iSCSINodes[]
#index = 0
```

```
for (#i in $ProtocolControllers[]) {
    if ($ProtocolControllers[#i].NameFormat == “iSCSI Name”) {
     // Filter out SCSIProtocolControllers previously encountered.
     if (!contains($ProtocolControllers[#i].Name, #NodeNames[])) {
         #NodeNames[#index] = $ProtocolControllers[#i].Name
         $iSCSINodes[#index++] = $ProtocolControllers[#i]
     }
    }
}
<EXIT: $Nodes[] contains the results>
```

8.2.2.3.5.3 Identify the iSCSI Ports on an given iSCSI node.

```
// DESCRIPTION
// Identify the iSCSI Ports on an given iSCSI node.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The SCSIProtocolController representing an iSCSI Node of interest has
// been previously identified and defined in the $iSCSINode-> variable.

// This function returns the instance(s) of iSCSI ports on the specified
// iSCSI node, or null if none are found.
sub $iSCSIPorts[] getiSCSIPortsOnNode($Node->) {

    // Step 1. Locate the iSCSI Ports, which are represented by instances of
    // iSCSIProtocolEndpoint, on the iSCSI Node of interest.
    $iSCSIPorts[] = Associators($iSCSINode->,
         “CIM_SAPAvailableForElement”,
         “CIM_iSCSIProtocolEndpoint”,
         “ManagedElement”,
         “AvailableSAP”,
         false,
         false,
         {“Name”, “Identifier”, “Role”})

    if ($iSCSIPorts[].length == 0) {
     return (null)
    }
    return ($iSCSIPorts[])
}

// MAIN
$iSCSIPorts[] = &getiSCSIPortsOnNode($iSCSINode->)
```

8.2.2.3.5.4 Identify the iSCSI sessions existing on an iSCSI node.

```
// DESCRIPTION
// Identify the iSCSI sessions existing on an iSCSI node.
```

```
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The SCSIProtocolController representing the iSCSI Node of interest has
// been previously identified and defined in the $iSCSINode-> variable

// Step 1. Retrieve the CIM_iSCSIProtocolEndpoints for an
// CIM_SCSIProtocolController representing a node.
$iSCSIPorts[] = @getiSCSIPortsOnNode($iSCSINode->)
if ($iSCSIPorts[] == null) {
    <ERROR! No iSCSI ports located on the specified iSCSI node>
}

// Step 2. Retrieve the iSCSI session associated with each iSCSI port.
$iSCSISessions[]
#index = 0
#PropList[] = {“Directionality”, “SessionType”, “TSIH”, “EndPointName”,
     “CurrentConnections”, “InitialR2T”, “ImmediateData”,
     “MaxOutstandingR2T”, “MaxUnsolicitedFirstDataBurstLength”,
     “MaxDataBurstLength”, “AuthenticationMethodUsed”,
     “DataSequenceInOrder”, “DataPDUInOrder”, “ErrorRecoveryLevel”}
for (#i in $iSCSIPorts[]) {
    $Sessions[] = Associators($iSCSIPorts[#i].getObjectPath(),
     “CIM_EndpointOfNetworkPipe”,
     “CIM_iSCSISession”,
     “Antecedent”,
     “Dependent”,
     #PropList[])
    if ($Sessions[] != null && $Sessions[].length == 1) {
     $iSCSISessions[#index++] = $Sessions[0]
    }
}
<EXIT: $iSCSISessions[] contains the iSCSI Sessions>
```

8.2.2.3.5.5 Create an iSCSI Target Node on an iSCSI Network Entity

```
// DESCRIPTION
// Create an iSCSI Target Node on an iSCSI Network Entity
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The ComputerSystem representing the Network Entity of interest has been
// previously identified and defined in the $NetworkEntity-> variable.

// MAIN
// Step 1. Locate the CIM_iSCSIConfigurationService hosted by the System.
// Note that active iSCSI configuration may not be supported by the device.
try {
    $iSCSIConfigurationService->[] = AssociatorNames($NetworkEntity->,
        “CIM_HostedService”,
         “CIM_iSCSIConfigurationService”,
         “Antecedent”,
         “Dependent”)
    // iSCSIConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($iSCSIConfigurationService->[] == null ||
         $iSCSIConfigurationService->[].length == 0) {
     <EXIT: iSCSI Configuration is not supported>
    }
} catch (CIMException $Exception) {
    // iSCSIConfigurationService and/or HostedService may not be included in
    // the model implemented at all if iSCSI Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
     <EXIT: iSCSI Configuration is not supported.>
    }
}

// Step 2. Examine the capabilities to determine if Node creation is supported.
$ConfigurationCapabilities[] = Associators($iSCSIConfigurationService->[0],
     “CIM_ElementCapabilities”,
     “CIM_iSCSIConfigurationCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“iSCSINodeCreationSupported”})
if ($ConfigurationCapabilities[] == null ||
     $ConfigurationCapabilities[].length == 0) {
    <ERROR! Required iSCSI Configuration Service capabilities not available>
}

// Step 3. Create the iSCSI Target Node if supported by the device.
if ($ConfigurationCapabilities[0].iSCSINodeCreationSupported  == true) {
    %InArguments[“Alias”] = “Some Target Alias”
    #ReturnValue = invokeMethod($iSCSIConfigurationService->[0],
         “CreateiSCSINode”,
         %InArguments[],
         %OutArguments[])

    if (#ReturnValue == 0) {
     $NewNode-> = $OutArguments[“iSCSINode”]
     <EXIT: The node was created>
    } else {
     <EXIT: The method returned an error; the Node was not created>
    }
} else {
    <EXIT: Node Creation is not supported>
```

```
}
```

8.2.2.3.5.6 Create an iSCSI Target Port on an iSCSI target node.

```
// DESCRIPTION
// This recipe describes how to create an iSCSI Target Port on an iSCSI target
// node.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the ComputerSystem representing the Network Entity of
// interest has been previously identified and defined in the $NetworkEntity->
// variable.
// 2. The object name for the SCSIProtocolController representing the iSCSI Node
// within which to create the iSCSI Port has been identified and defined in the //
$Node-> variable.
// 3. The object names for one or more TCPProtocolEndpoints representing Target
// Network Portals have been previously identified and defined in the
// Portals->[] array variable.

// MAIN
// Step 1. Find a CIM_iSCSIConfigurationService associated to ComputerSystem
// by HostedService. Note that active iSCSI configuration may not be supported
// by the device.
try {
    $iSCSIConfigurationService->[] = AssociatorNames($NetworkEntity->,
         “CIM_HostedService”,
         “CIM_iSCSIConfigurationService”,
         “Antecedent”,
         “Dependent”)
    // iSCSIConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($iSCSIConfigurationService->[] == null ||
         $iSCSIConfigurationService->[].length == 0) {
     <EXIT: iSCSI Configuration is not supported>
    }
} catch (CIMException $Exception) {
    // iSCSIConfigurationService and/or HostedService may not be included in
    // the model implemented at all if iSCSI Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
     <EXIT: iSCSI Configuration is not supported.>
    }
}

// Step 2. Examine the associated CIM_iSCSIConfigurationCapabilities to
// determine if Target Port manipulation is supported.
$ConfigurationCapabilities[] = Associators($iSCSIConfigurationService->[0],
     “CIM_ElementCapabilities”,
     “CIM_iSCSIConfigurationCapabilities”,
      “ManagedElement”,
      “Capabilities”,
      false,
      false,
      {“iSCSIProtocolEndpointCreationSupported”})

// Step 3. Given an instance of CIM_SCSIProtocolController representing a
// Node($Node->), and one or more TCPProtocolEndpoints representing Target
// Network Portals(Portals->[]), invoke the method CreateiSCSIProtocolEndpoint
// to create the iSCSIProtocolEndpoint.
if ($ConfigurationCapabilities[0].iSCSIProtocolEndpointCreationSupported == true)
{

    %InArguments[“iSCSINode”] = $Node->
    %InArguments[“Role”] = 3// “Target”
    %InArguments[“NetworkPortals”] = Portals->[]

    #ReturnValue = InvokeMethod($iSCSIConfigurationService->[0],
         “CreateiSCSIProtocolEndpoint”,
         %InArguments[],
         %OutArguments[])

    if (#ReturnValue == 0) {
     $NewiSCSIProtocolEndpoint-> = $OutArguments[“iSCSIPort”]
     <EXIT: The ProtocolEndpoint was created>
    } else {
     <EXIT: The method returned an error; the ProtocolEndpoint was not created>
    }
} else {
    <EXIT: iSCSIProtocolEndpoint creation is not supported>
}
```

8.2.2.3.5.7 Add a Network Portal to a Target Port.

```
// DESCRIPTION
// This recipe describes how to add a Network Portal to a Target Port.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the ComputerSystem representing the Network Entity of
// interest has been previously identified and defined in the $NetworkEntity->
// variable.
// 2. The object name for the instance of iSCSIProtocolEndpoint representing a
// Port has been previously identified and defined in the $iSCSIPort-> variable.
// 3. The object name for the instance of TCPProtocolEndpoint representing a
// Target Network Portal has been previously identified and defined in the
// $Portal-> variable.

// MAIN
// Step 1. Find a CIM_iSCSIConfigurationService associated to ComputerSystem by //
HostedService. Note that active iSCSI configuration may not be supported by
// the device.
try {
    $iSCSIConfigurationService->[] = AssociatorNames($NetworkEntity->,
         “CIM_HostedService”,
         “CIM_iSCSIConfigurationService”,
         “Antecedent”,
         “Dependent”)
    // iSCSIConfigurationService and HostedService may not be implemented
    // in the SMI Agent.
    if ($iSCSIConfigurationService->[] == null ||
         $iSCSIConfigurationService->[].length == 0) {
     <EXIT: iSCSI Configuration is not supported>
    }
} catch (CIMException $Exception) {
    // iSCSIConfigurationService and/or HostedService may not be included in
    // the model implemented at all if iSCSI Configuration is not supported.
    if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
     <EXIT: iSCSI Configuration is not supported.>
    }
}

// Step 2. Examine the associated CIM_iSCSIConfigurationCapabilities to
// determine if Target Port manipulation is supported.
$ConfigurationCapabilities[] = Associators($iSCSIConfigurationService->[0],
     “CIM_ElementCapabilities”,
     “CIM_iSCSIConfigurationCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“iSCSIProtocolEndpointCreationSupported”})

// Step 3. Given an instance of CIM_iSCSIProtocolEndpoint representing a
// Port (iSCSIPort->), and an instance of TCPProtocolEndpoint representing a
// Target Network Portal($Portal->), invoke BindiSCSIProtocolEndpoint().
if ($ConfigurationCapabilities[0].iSCSIProtocolEndpointCreationSupported == true)
{

    %InArguments[“iSCSIPort”] = $iSCSIPort->
    %InArguments[“NetworkPortal”] = $Portal->

    #ReturnValue = invokeMethod($iSCSIConfigurationService->[0],
         “BindiSCSIProtocolEndpoint”,
         %InArguments[],
         %OutArguments[])
    if (#ReturnValue == 0) {
     <EXIT: The ProtocolEndpoint was modified>
    } else {
     <EXIT: The method returned an error; the ProtocolEndpoint was not modified>
    }
} else {
    <EXIT: iSCSIProtocolEndpoint modification is not supported>
}
```

8.2.2.3.5.8 Determine the health of Nodes in a target system.

```
// DESCRIPTION
//
// Determine the health of Nodes in a target system.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the SCSIProtocolController representing the iSCSI
// Node of interest has been previously identified and defined in the
// $iSCSINode-> variable.

// Step 1. Given an instance of SCSIProtocolController, get the instances of
// iSCSISessionFailures and iSCSILoginStatistics associated by
// ElementStatisticalData.
//
$SessionFailures[] = Associators($iSCSINode->,
     “CIM_ElementStatisticalData”,
     “CIM_iSCSISessionFailures”,
     “ManagedElement”,
     “Stats”);

$LoginStatistics[] = Associators($iSCSINode->,
     “CIM_ElementStatisticalData”,
     “CIM_iSCSILoginStatistics”,
     “ManagedElement”,
     “Stats”);

<EXIT: The statistics are in $SessionFailures[0] and $LoginStatistics[0]>
```

8.2.2.3.5.9 Determine the health of a Session on a target system.

```
// DESCRIPTION
//
// Determine the health of a Session on a target system.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the iSCSISession of interest has been previously
// identified and defined in the $iSCSISession-> variable.
// Step 1. Given an instance of iSCSISession, get the instance of
// iSCSISessionStatistics associated by ElementStatisticalData.
//
$SessionStatistics[] = Associators($iSCSISession->,
    “CIM_ElementStatisticalData”,
    “CIM_iSCSISessionStatistics”,
    “ManagedElement”,
    “Stats”);

<EXIT: The statistics are in $SessionStatistics[0]>
```

8.2.2.3.5.10 Configure the default settings for Sessions created in a target computer system.

```
// DESCRIPTION
//
// Configure the default settings for iSCSI Sessions created on a target device.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the iSCSI endpoint of interest has been previously
// identified and defined in the $iSCSIProtocolEndpoint-> variable.
// 2. A supported value for the iSCSISessionSettings.MaxConnectionsPerSession
// has previously been identified and defined in the #MaxConnectionsPerSession
// variable.

// Step 1. Find the instance of iSCSISessionSettings associated to the
// iSCSIProtocolEndpoint.
$SessionSettings[] = Associators($iSCSIProtocolEndpoint->,
     “CIM_ElementSettingData”,
     “CIM_iSCSISessionSettings”,
     “ManagedElement”,
     “SettingData”);

// Step 2. Attempt to modify the permissible connections per session setting.
//
try {
    $SessionSettings[0].MaxConnectionsPerSession = #MaxConnectionsPerSession;

    ModifyInstance($SessionSettings[0],
         false,
         {“MaxConnectionsPerSession”});
    <EXIT: Success>
} catch (CIMException $Exception) {
    // Note that the implementation may be read-only and may not support
    // modification of settings. In this case, CIM_ERR_NOT_SUPPORTED must be
    // returned when an attempt is made to modify the instance.
    if ($Exception.CIMStatusCode == CIM_ERR_NOT_SUPPORTED) {
     <EXIT: Session settings modification is not supported>
    }
```

```
        <ERROR! Failure; an unexpected error was encountered.>
    }
```

8.2.2.3.5.11 Configure the default settings for iSCSI Connections created on Network Portals used by an iSCSIProtocolEndpoint.

```
// DESCRIPTION
//
// Configure the default settings for iSCSI Connections created on a Network
// Portal used by an iSCSIProtocolEndpoint.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The object name for the iSCSI endpoint of interest has been previously
// identified and defined in the $iSCSIProtocolEndpoint-> variable.
// 2. A supported value for the
iSCSIConnectionSettings.MaxReceiveDataSegmentLength
// has previously been identified and defined in the #MaxRecvDataSegLength
// variable.

// Step 1. Find the instance of iSCSIConnectionSettings associated to the
// iSCSIProtocolEndpoint.
//
$ConnectionSettings[] = Associators($iSCSIProtocolEndpoint->,
     “CIM_ElementSettingData”,
     “CIM_iSCSIConnectionSettings”,
     “ManagedElement”,
     “SettingData”);

// Step 2. Attempt to modify the permissible received data segment length
// on the connection.
//
try {
    $ConnectionSettings[0].MaxReceiveDataSegmentLength = #MaxRecvDataSegLength;

    ModifyInstance($ConnectionSettings[0],
         false,
         {“MaxReceiveDataSegmentLength”});
    <EXIT: Success>
} catch (CIMException $Exception) {
    // Note that the implementation may be read-only and may not support
    // modification of settings. In this case, CIM_ERR_NOT_SUPPORTED must be
    // returned when an attempt is made to modify the instance.
    if ($Exception.CIMStatusCode == CIM_ERR_NOT_SUPPORTED) {
     <EXIT: Connection settings modification is not supported>
    }
    <ERROR! Failure; an unexpected error was encountered.>
}
```

8.2.2.3.5.12 Get the statistics for a Session on a target system

The statistics are properties in the same class as the health information; see 8.2.2.3.5.9, "Determine the health of a Session on a target system."

8.2.2.3.5.13 Configure Enable/disable header and data digest

See 8.2.2.3.5.11, "Configure the default settings for iSCSI Connections created on Network Portals used by an iSCSIProtocolEndpoint."

8.2.2.3.6 Registered Name and Version

iSCSI Target Ports version 1.1.0

8.2.2.3.7 CIM Server Requirements

Table 273: CIM Server Requirements for iSCSI Target Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.2.3.8 CIM Elements

Table 274: CIM Elements for iSCSI Target Ports

Element Name	Description
Mandatory Classes	
CIM_BindsTo (8.2.2.3.8.1)	
CIM_ConcreteDependency (8.2.2.3.8.2)	
CIM_ElementCapabilities (8.2.2.3.8.4)	
CIM_ElementSettingData (8.2.2.3.8.5)	
CIM_ElementStatisticalData (8.2.2.3.8.6)	
CIM_EndpointOfNetworkPipe (8.2.2.3.8.7)	
CIM_HostedAccessPoint (8.2.2.3.8.9)	
CIM_HostedCollection (8.2.2.3.8.10)	
CIM_IPProtocolEndpoint (8.2.2.3.8.12)	
CIM_NetworkPipeComposition (8.2.2.3.8.14)	
CIM_SAPAvailableForElement (8.2.2.3.8.15)	
CIM_SCSIProtocolController (8.2.2.3.8.16)	
CIM_SystemDevice (8.2.2.3.8.17)	This association links all **LogicalDevices**to the scoping system.
CIM_TCPProtocolEndpoint (8.2.2.3.8.19)	
CIM_iSCSICapabilities (8.2.2.3.8.20)	
CIM_iSCSIProtocolEndpoint (8.2.2.3.8.26)	
CIM_iSCSISession (8.2.2.3.8.27)	
CIM_iSCSISessionSettings (8.2.2.3.8.29)	
Optional Classes	
CIM_DeviceSAPImplementation (8.2.2.3.8.3)	
CIM_EthernetPort (8.2.2.3.8.8)	
CIM_HostedService (8.2.2.3.8.11)	
CIM_MemberOfCollection (8.2.2.3.8.13)	
CIM_SystemSpecificCollection (8.2.2.3.8.18)	

Table 274: CIM Elements for iSCSI Target Ports

Element Name	Description
CIM_iSCSIConfigurationCapabilities (8.2.2.3.8.21)	
CIM_iSCSIConfigurationService (8.2.2.3.8.22)	
CIM_iSCSIConnection (8.2.2.3.8.23)	
CIM_iSCSIConnectionSettings (8.2.2.3.8.24)	
CIM_iSCSILoginStatistics (8.2.2.3.8.25)	
CIM_iSCSISessionFailures (8.2.2.3.8.28)	
CIM_iSCSISessionStatistics (8.2.2.3.8.30)	
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_iSCSIProtocolEndpoint	Create iSCSIProtocolEndpoint
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_iSCSIProtocolEndpoint	Delete SCSIProtocolEndpoint
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SCSIProtocolController	Create SCSIProtocolController
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SCSIProtocolController	Delete iSCSIProtocolController
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_iSCSISessionSettings	Modify iSCSISessionSettings
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_EthernetPort	Create EthernetPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_EthernetPort AND SourceInstance.CIM_EthernetPort::OperationalStatus <> PreviousInstance.CIM_EthernetPort::OperationalStatus	CQL - Modify EthernetPort
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_EthernetPort	Delete EthernetPort
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_iSCSISession	Create iSCSISession
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_iSCSISession AND SourceInstance.CIM_iSCSISession::CurrentConnections <> PreviousInstance.CIM_iSCSISession::CurrentConnections	CQL - Modify iSCSISession
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_iSCSISession	Delete iSCSISession
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_iSCSIConnection	Create iSCSIConnection
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_iSCSIConnection	Delete iSCSIConnection
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_iSCSIConnectionSettings	Modify iSCSIConnectionSettings

8.2.2.3.8.1 CIM_BindsTo

Created By : External or StaticExtrinsic(s):

Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 275: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolEndpoint	
Dependent		CIM_ServiceAccessPoint	

8.2.2.3.8.2 CIM_ConcreteDependency

Created By : StaticExtrinsic(s):
Deleted By : Extrinsic(s):
Class Mandatory: true

Table 276: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	
Dependent		CIM_ManagedElement	

8.2.2.3.8.3 CIM_DeviceSAPImplementation

Created By : Static or External
Deleted By : External
Class Mandatory: false

Table 277: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.2.3.8.4 CIM_ElementCapabilities

Created By : Static
Class Mandatory: true

Table 278: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Capabilities		CIM_Capabilities	

8.2.2.3.8.5 CIM_ElementSettingData

Created By : Static

Class Mandatory: true

Table 279: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
SettingData		CIM_SettingData	

8.2.2.3.8.6 CIM_ElementStatisticalData

Created By : Static
Class Mandatory: true

Table 280: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Stats		CIM_StatisticalData	

8.2.2.3.8.7 CIM_EndpointOfNetworkPipe

Created By : External
Deleted By : External
Class Mandatory: true

Table 281: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	
Dependent		CIM_NetworkPipe	

8.2.2.3.8.8 CIM_EthernetPort

Created By : Static or External
Deleted By : External
Standard Names: The PermanentAddress Property follows the requirements in6.2.4.5.2
Class Mandatory: false

Table 282: SMI Referenced Properties/Methods for CIM_EthernetPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
PermanentAddress	CD	string	

8.2.2.3.8.9 CIM_HostedAccessPoint

Created By : External or StaticExtrinsic(s):
Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 283: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.2.3.8.10 CIM_HostedCollection

Created By : Static
Class Mandatory: true

Table 284: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_SystemSpecificCollection	

8.2.2.3.8.11 CIM_HostedService

Created By : Static
Class Mandatory: false

Table 285: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_Service	

8.2.2.3.8.12 CIM_IPProtocolEndpoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 286: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	

Table 286: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
ProtocolIFType		uint16	
Optional Properties/Methods			
IPv4Address	CD	string	
IPv6Address	CD	string	

8.2.2.3.8.13 CIM_MemberOfCollection

Created By : Static or External
Deleted By : External
Class Mandatory: false

Table 287: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	
Member		CIM_ManagedElement	

8.2.2.3.8.14 CIM_NetworkPipeComposition

Created By : External
Deleted By : External
Class Mandatory: true

Table 288: SMI Referenced Properties/Methods for CIM_NetworkPipeComposition

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_NetworkPipe	
PartComponent		CIM_NetworkPipe	

8.2.2.3.8.15 CIM_SAPAvailableForElement

Created By : StaticExtrinsic(s):
Deleted By : Extrinsic(s):
Class Mandatory: true

Table 289: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AvailableSAP		CIM_ServiceAccessPoint	
ManagedElement		CIM_ManagedElement	

8.2.2.3.8.16 CIM_SCSIProtocolController

Created By : StaticExtrinsic(s):
Modified By : Extrinsic(s):
Deleted By : Extrinsic(s):

Class Mandatory: true

Table 290: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ElementName		string	iSCSI Alias
Name	CD	string	
NameFormat		uint16	

8.2.2.3.8.17 CIM_SystemDevice

This association links all **LogicalDevices**to the scoping system.
Created By : StaticExtrinsic(s):
Deleted By : Extrinsic(s):
Class Mandatory: true

Table 291: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.2.3.8.18 CIM_SystemSpecificCollection

Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 292: SMI Referenced Properties/Methods for CIM_SystemSpecificCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	

8.2.2.3.8.19 CIM_TCPProtocolEndpoint

Created By : Static or External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 293: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
PortNumber		uint32	
ProtocolIFType		uint16	

8.2.2.3.8.20 CIM_iSCSICapabilities

Created By : Static
Class Mandatory: true

Table 294: SMI Referenced Properties/Methods for CIM_iSCSICapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
MinimumSpecificationVersionSupported		uint8	
MaximumSpecificationVersionSupported		uint8	
AuthenticationMethodsSupported		uint16[]	

8.2.2.3.8.21 CIM_iSCSIConfigurationCapabilities

Created By : Static
Class Mandatory: false

Table 295: SMI Referenced Properties/Methods for CIM_iSCSIConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
iSCSINodeCreationSupported		boolean	
iSCSIProtocolEndpointCreationSupported		boolean	
IdentifierSelectionSupported		boolean	

8.2.2.3.8.22 CIM_iSCSIConfigurationService

Created By : Static

Class Mandatory: false

Table 296: SMI Referenced Properties/Methods for CIM_iSCSIConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
CreateiSCSINode()			
DeleteiSCSINode()			
CreateiSCSIProtocolEndpoint()			
DeleteiSCSIProtocolEndpoint()			
BindiSCSIProtocolEndPoint()			

8.2.2.3.8.23 CIM_iSCSIConnection

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 297: SMI Referenced Properties/Methods for CIM_iSCSIConnection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ConnectionID		uint32	
MaxReceiveDataSegmentLength		uint32	
MaxTransmitDataSegmentLength		uint32	
HeaderDigestMethod		uint16	
DataDigestMethod		uint16	
ReceivingMarkers		boolean	
SendingMarkers		boolean	
ActiveiSCSIVersion		boolean	
AuthenticationMethodUsed		uint16	
MutualAuthentication		boolean	
Optional Properties/Methods			
OtherHeaderDigestMethod		string	
OtherDataDigestMethod		string	

8.2.2.3.8.24 CIM_iSCSIConnectionSettings

Created By : Static
Modified By : ModifyInstance

Class Mandatory: false

Table 298: SMI Referenced Properties/Methods for CIM_iSCSIConnectionSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
MaxReceiveDataSegmentLength		uint32	
PrimaryHeaderDigestMethod		uint16	
PrimaryDataDigestMethod		uint16	
SecondaryHeaderDigestMethod		uint16	
SecondaryDataDigestMethod		uint16	
RequestingMarkersOnReceive		boolean	
PrimaryAuthenticationMethod		uint16	
SecondaryAuthenticationMethod		uint16	
Optional Properties/Methods			
OtherPrimaryHeaderDigest-Method		string	
OtherPrimaryDataDigestMethod		string	
OtherSecondaryHeaderDigest-Method		string	
OtherSecondaryDataDigest-Method		string	

8.2.2.3.8.25 CIM_iSCSILoginStatistics

Created By : Static
Modified By : External
Class Mandatory: false

Table 299: SMI Referenced Properties/Methods for CIM_iSCSILoginStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
Optional Properties/Methods			
LoginFailures		uint64	
LastLoginFailureTime		datetime	
LastLoginFailureType		uint16	
OtherLastLoginFailureType		string	
LastLoginFailureRemoteNode-Name		string	
LastLoginFailureRemoteAd-dressType		uint16	
LastLoginFailureRemoteAddress		uint32	

Table 299: SMI Referenced Properties/Methods for CIM_iSCSILoginStatistics

Property	Flags	Type	Description & Notes
SuccessfulLogins		uint64	
NegotiationLoginFailures		uint64	
AuthenticationLoginFailures		uint64	
AuthorizationLoginFailures		uint64	
LoginRedirects		uint64	
OtherLoginFailures		uint64	
NormalLogouts		uint64	
OtherLogouts		uint64	

8.2.2.3.8.26 CIM_iSCSIProtocolEndpoint

Created By : StaticExtrinsic(s):
Modified By : Extrinsic(s):
Deleted By : Extrinsic(s):
Standard Names: The Name Property follows the requirements in6.2.4.5.2
Class Mandatory: true

Table 300: SMI Referenced Properties/Methods for CIM_iSCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name	CD	string	
ConnectionType		uint16	Shall be 7 (iSCSI)
Identifier		string	ISID or TPGT
ProtocolIFType		uint16	Other
OtherTypeDescription		string	
Role		uint16	Shall be 3 (Target) or 4 (Both Initiator and Target)

8.2.2.3.8.27 CIM_iSCSISession

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 301: SMI Referenced Properties/Methods for CIM_iSCSISession

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Directionality		uint16	
SessionType		uint16	
TSIH		uint32	

Table 301: SMI Referenced Properties/Methods for CIM_iSCSISession

Property	Flags	Type	Description & Notes
EndPointName		string	
CurrentConnections		uint32	
InitialR2T		boolean	
ImmediateData		boolean	
MaxOutstandingR2T		uint32	
MaxUnsolicitedFirstDataBurst-Length		uint32	
MaxDataBurstLength		uint32	
DataSequenceInOrder		boolean	
DataPDUInOrder		boolean	
ErrorRecoveryLevel		uint32	
MaxConnectionsPerSession		uint32	
DefaultTimeToWait		uint32	
DefaultTimeToRetain		uint32	

8.2.2.3.8.28 CIM_iSCSISessionFailures

Created By : Static
Modified By : External
Class Mandatory: false

Table 302: SMI Referenced Properties/Methods for CIM_iSCSISessionFailures

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
Optional Properties/Methods			
SessionFailures		uint64	
LastSessionFailureType		uint16	
OtherLastSessionFailureType		string	
LastSessionFailureRemoteNode-Name		string	
SessionDigestFailures		uint64	
SessionConnectionTimeoutFail-ures		uint64	
SessionFormatErrors		uint64	

8.2.2.3.8.29 CIM_iSCSISessionSettings

Created By : Static
Modified By : ModifyInstance

Class Mandatory: true

Table 303: SMI Referenced Properties/Methods for CIM_iSCSISessionSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
MaxConnectionsPerSession		uint32	
InitialR2TPreference		boolean	
ImmediateDataPreference		boolean	
MaxOutstandingR2T		uint32	
MaxUnsolicitedFirstDataBurst-Length		uint32	
MaxDataBurstLength		uint32	
DataSequenceInOrderPreference		boolean	
DataPDUInOrderPreference		boolean	
DefaultTimeToWaitPreference		uint32	
DefaultTimeToRetainPreference		uint32	
ErrorRecoveryLevelPreference		uint32	

8.2.2.3.8.30 CIM_iSCSISessionStatistics

Created By : Static
Modified By : External
Class Mandatory: false

Table 304: SMI Referenced Properties/Methods for CIM_iSCSISessionStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
Optional Properties/Methods			
CommandPDUsTransferred		uint64	
ResponsePDUsTransferred		uint64	
BytesTransmitted		uint64	
BytesReceived		uint64	
DigestErrors		uint64	
ConnectionTimeoutErrors		uint64	

8.2.2.3.9 Related Standards

Table 305: Related Standards for iSCSI Target Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
Representation of CIM using XML	2.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.2.4 Direct Attach (DA) Port Subprofile

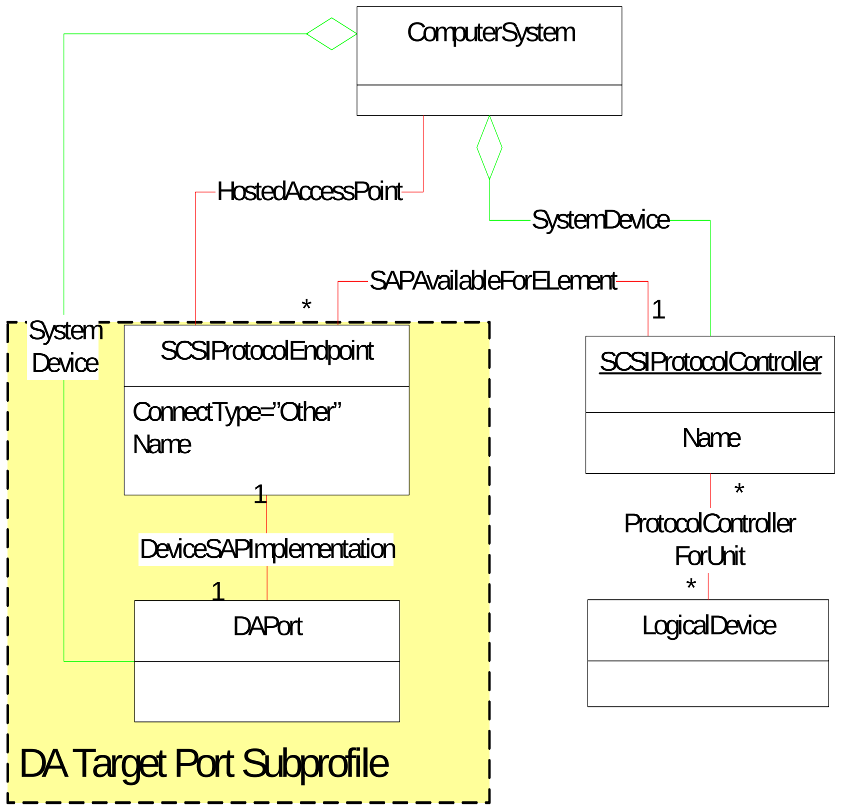
8.2.2.4.1 Description

The DAPort (Direct Attach) port models storage systems that attach directly to buses in a host system (e.g., ISA, EISA, PCI, PCI-E, and chip interfaces on a motherboard). The DAPort can be viewed as both the initiator and Target ports.

This port can not be used with the LUN Mapping/Masking common subprofile. All volumes served by this port are fully accessible by the host system.

Volumes served by this port shall be discovered and presented by the Host Discovered Resources Profile.

Figure 59: DA Port Instance Diagram



The DAPort class is connected to the SCSIProtocolEndpoint and optionally to a PhysicalPackage. The DAPort also contains a port type attribute to identify the interconnect technology.

8.2.2.4.2 Durable Names and Correlatable IDs of the Subprofile

None

8.2.2.4.3 Health and Fault Management

Table 306: DAPort OperationalStatus

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

8.2.2.4.4 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.2.4.5 Extrinsic Methods of this Subprofile

None

8.2.2.4.6 Client Considerations and Recipes

8.2.2.4.7 Registered Name and Version

DA Target Ports version 1.1.0

8.2.2.4.8 CIM Server Requirements

Table 307: CIM Server Requirements for DA Target Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.2.4.9 CIM Elements

Table 308: CIM Elements for DA Target Ports

Element Name	Description
Mandatory Classes	
CIM_DAPort (8.2.2.4.9.1)	
CIM_DeviceSAPImplementation (8.2.2.4.9.2)	
CIM_HostedAccessPoint (8.2.2.4.9.3)	
CIM_SCSIProtocolEndpoint (8.2.2.4.9.4)	
CIM_SystemDevice (8.2.2.4.9.5)	

8.2.2.4.9.1 CIM_DAPort

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 309: SMI Referenced Properties/Methods for CIM_DAPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
UsageRestriction		uint16	Shall be 2 to indicate this is a front end port.

8.2.2.4.9.2 CIM_DeviceSAPImplementation

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

Class Mandatory: true

Table 310: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.2.4.9.3 CIM_HostedAccessPoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 311: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.2.4.9.4 CIM_SCSIProtocolEndpoint

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 312: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
OtherTypeDescription		string	
ConnectionType		uint16	Simulates SPI
Role		uint16	

8.2.2.4.9.5 CIM_SystemDevice

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

Class Mandatory: true

Table 313: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.2.4.10 Related Standards

Table 314: Related Standards for DA Target Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.3 Common Initiator Port Subprofiles Overview

Hosts and some storage systems provide interfaces to discover and manage the connections between the system and connected storage (physical disks, external storage). For example, an array may have an interface to acquire and optimize the utilization of separate buses, loops, or fabrics connecting backend storage. The discovered target elements (port and logical units) may be modeled by the scoping profile.

SMI-S includes separate initiator ports subprofiles for several types of transport (Fibre Channel, iSCSI, Parallel SCSI,...). Rather than include the same classes, the appropriate initiator ports subprofile is also required in the FC HBA and iSCSI Initiator Profiles.

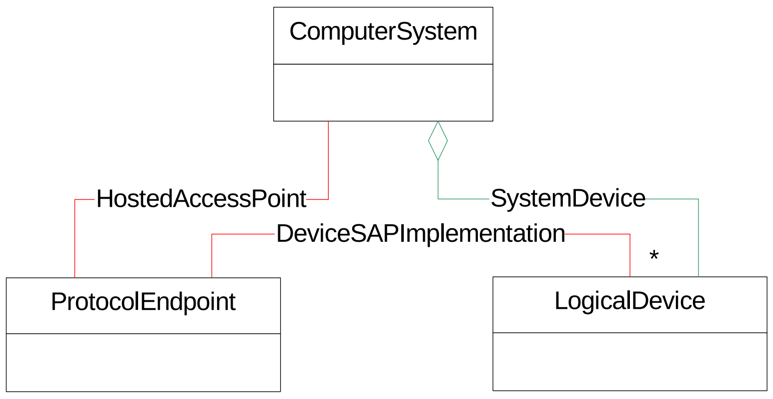
Generic Model

The initiator port is modeled as a ProtocolEndpoint connected to a port (LogicalPort) The port is modeled as two separate instances to capture the manageable properties of the physical connector and transport (LogicalPort) separately from those of the protocol(s) (ProtocolEndpoint) used for communication. Some transports (such as parallel SCSI - SPI) lock the physical port to a specific protocol. But an FC Port can carry SCSI and/or IP as protocols. With iSCSI, an Ethernet port can support SCSI and a variety of network protocols.

The LogicalDevice instances may represent local storage (embedded in the system containing the initiator ports) or remote storage. When it represents remote storage the Name and NameFormat properties are used as correlatable ids to reference the remote device. When the LogicalDevice represents local disk storage, it may be represented as an instance of StorageVolume (subclass of LogicalDevice) or part of an instance of the Disk Drive common subprofile. A property on LogicalPort called UsageRestriction is available to indicate whether the controller is capable of providing a “front end” (target), a “back end” (initiator), or both interfaces.

Figure 60:, "Generic Initiator Port Model" depicts the generic model.

Figure 60: Generic Initiator Port Model

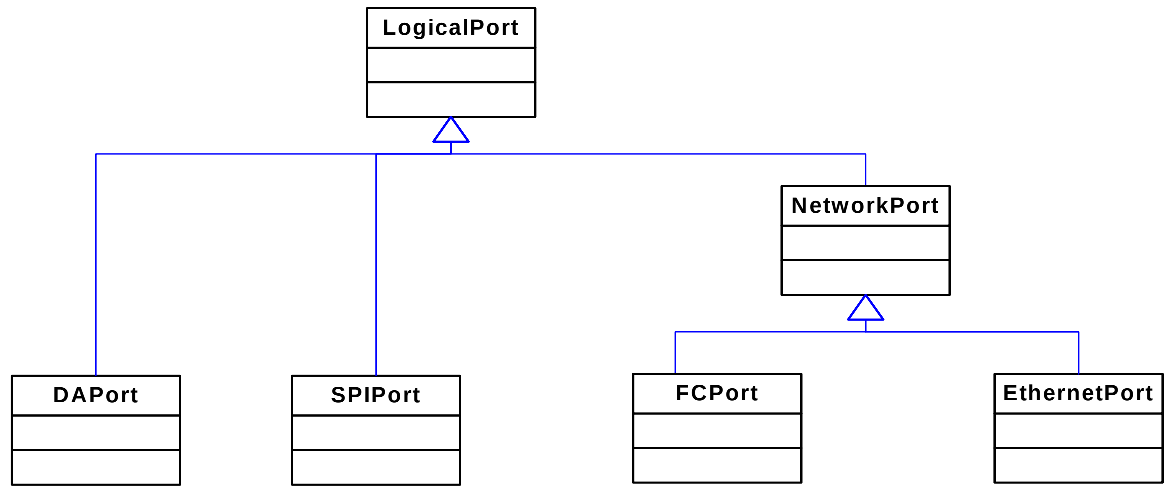


Ports and Transport Types

Several technologies are used to attach storage to initiators. The interconnect technology is represented by subclasses of LogicalPort. FCPort and EthernetPort are further sub classed from NetworkPort to show that the port participates in a network. The other ports are simple buses.

SPIPort	SCSI Parallel Interface
FCPort	Fibre Channel
EthernetPort	iSCSI

Figure 61: Logical Port Hierarchy



Associations to the Containing Profile

Each Initiator Port Subprofile includes instances of subclasses of LogicalPort and ProtocolEndpoint. These are associated to the ComputerSystem of the containing profile. Other associations may also tie to instances from the containing profile or one of its other subprofiles.

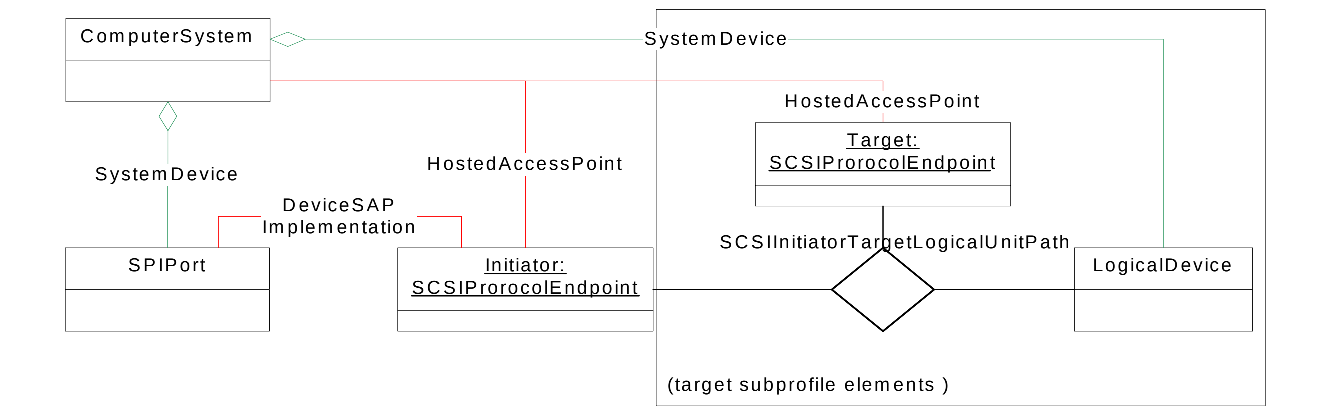
EXPERIMENTAL

8.2.3.1 Parallel SCSI (SPI) Initiator Port Subprofile

8.2.3.1.1 Description

The SPI Initiator Port Subprofile defines the model to parallel SCSI ports. A typical instance diagram is provided in Figure 62: "SPI Initiator Port Instance Diagram".

Figure 62: SPI Initiator Port Instance Diagram



8.2.3.1.2 Health and Fault Management Considerations

Table 315: “SPIPort OperationalStatus” summarized the Health and Fault Management issues that are unique to this profile.

Table 315: SPIPort OperationalStatus

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

8.2.3.1.3 Dependencies on Profiles, Subprofiles and Packages

None

8.2.3.1.4 Methods of the Subprofile

None

8.2.3.1.5 Client Considerations and Recipes

None

8.2.3.1.6 Registered Name and Version

SPI Initiator Ports version 1.1.0

8.2.3.1.7 CIM Server Requirements

Table 316: CIM Server Requirements for SPI Initiator Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.3.1.8 CIM Elements

Table 317: CIM Elements for SPI Initiator Ports

Element Name	Description
Mandatory Classes	
CIM_DeviceSAPImplementation (8.2.3.1.8.1)	
CIM_HostedAccessPoint (8.2.3.1.8.2)	
CIM_SCSIProtocolEndpoint (8.2.3.1.8.5)	
CIM_SPIPort (8.2.3.1.8.6)	
CIM_SystemDevice (8.2.3.1.8.7)	
Optional Classes	
CIM_LogicalDevice (8.2.3.1.8.3)	
CIM_SCSIInitiatorTargetLogicalUnitPath (8.2.3.1.8.4)	

8.2.3.1.8.1 CIM_DeviceSAPImplementation

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 318: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.3.1.8.2 CIM_HostedAccessPoint

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

Class Mandatory: true

Table 319: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.3.1.8.3 CIM_LogicalDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 320: SMI Referenced Properties/Methods for CIM_LogicalDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Name		string	
OperationalStatus		uint16[]	

8.2.3.1.8.4 CIM_SCSIInitiatorTargetLogicalUnitPath

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 321: SMI Referenced Properties/Methods for CIM_SCSIInitiatorTargetLogicalUnitPath

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Initiator		CIM_SCSIProtocolEndpoint	
Target		CIM_SCSIProtocolEndpoint	
LogicalUnit		CIM_LogicalDevice	

8.2.3.1.8.5 CIM_SCSIProtocolEndpoint

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

Class Mandatory: true

Table 322: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
OtherTypeDescription		string	
ConnectionType		uint16	
Role		uint16	Shall be 2 (Initiator) or 4 (Both Initiator and Target)

8.2.3.1.8.6 CIM_SPIPort

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 323: SMI Referenced Properties/Methods for CIM_SPIPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
UsageRestriction		uint16	Shall be 3 for ports restricted to Back-end only or 4 if the port is unrestricted

8.2.3.1.8.7 CIM_SystemDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 324: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.3.1.9 Related Standards

Table 325: Related Standards for SPI Initiator Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

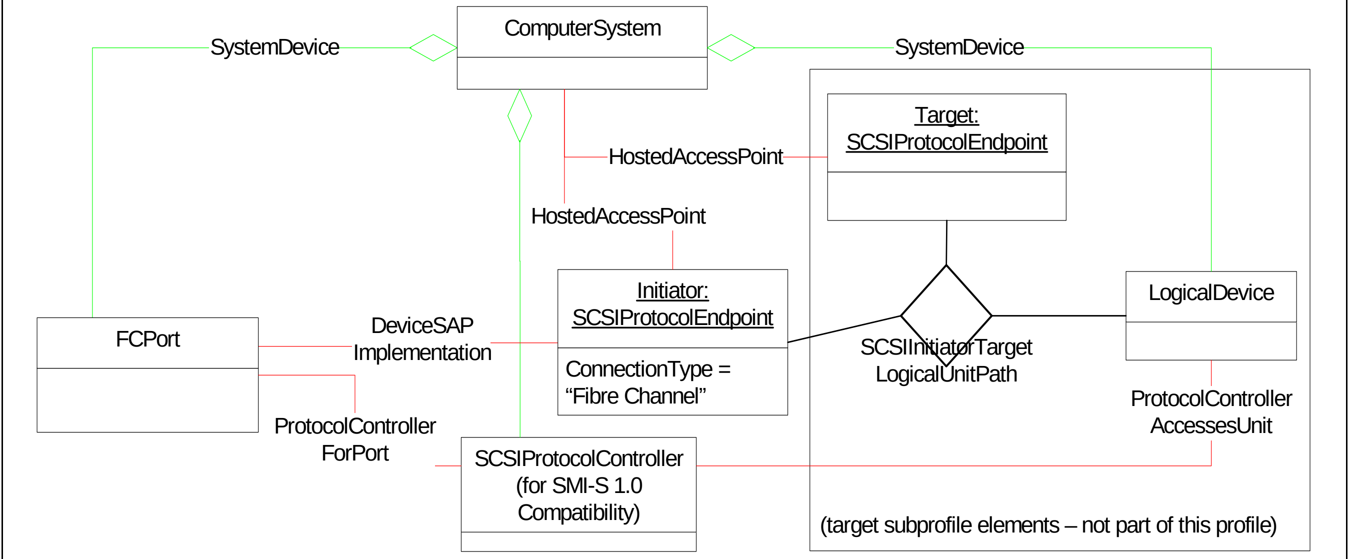
EXPERIMENTAL

8.2.3.2 Fibre Channel Initiator Port Subprofile

8.2.3.2.1 Description

Table 63 is an example of a single port and drive connected to a single system using Fibre Channel. The instance diagram shows a disk (LogicalDevice in the diagram would be subclassed as something like StorageExtent) in an array, connected by a Fibre Channel port. The full model for the disk is shown in the Disk Drive subprofile. SCSIProtocolController is not generally used in initiator contexts. It is included here to be compatible with SMI-S 1.0 clients as specified in IS24775-2006, *Storage Managemen*t

Figure 63: Fibre Channel Initiator Instance Diagram


ComputerSystem
SystemDevice
SystemDevice
Target: SCSIProtocolEndpoint
HostedAccessPoint
HostedAccessPoint
Initiator: SCSIProtocolEndpoint
ConnectionType = "Fibre Channel"
LogicalDevice
FCPort
DeviceSAP Implementation
SCSIInitiatorTarget LogicalUnitPath
ProtocolController ForPort
ProtocolController AccessesUnit
SCSIProtocolController (for SMI-S 1.0 Compatibility)
(target subprofile elements – not part of this profile)

8.2.3.2.2 Health and Fault Management Considerations

Table 326: "FCPort OperationalStatus" summarized the Health and Fault Management considerations specific to this profile.

Table 326: FCPort OperationalStatus

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

8.2.3.2.3 Cascading Considerations

Not defined in this standard.

8.2.3.2.4 Supported Subprofiles and Packages

None.

8.2.3.2.5 Methods of the Profile

None.

8.2.3.2.6 Client Considerations and Recipes

None

8.2.3.2.7 Registered Name and Version

FC Initiator Ports version 1.1.0

8.2.3.2.8 CIM Server Requirements

Table 327: CIM Server Requirements for FC Initiator Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.3.2.9 CIM Elements

Table 328: CIM Elements for FC Initiator Ports

Element Name	Description
Mandatory Classes	
CIM_DeviceSAPImplementation (8.2.3.2.9.1)	
CIM_FCPort (8.2.3.2.9.2)	
CIM_HostedAccessPoint (8.2.3.2.9.3)	
CIM_SCSIProtocolEndpoint (8.2.3.2.9.6)	
CIM_SystemDevice (8.2.3.2.9.7)	
Optional Classes	
CIM_ProtocolControllerForPort (8.2.3.2.9.4)	
CIM_SCSIProtocolController (8.2.3.2.9.5)	Represents a 'LUN Map'
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort	Create FCPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Modify FCPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.CIM_FCPort::OperationalStatus <> PreviousInstance.CIM_FCPort::OperationalStatus	CQL - Modify FCPort
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort	Delete FCPort

8.2.3.2.9.1 CIM_DeviceSAPImplementation

Created By : External
Modified By : Extrinsic(s):
Deleted By : External
Class Mandatory: true

Table 329: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.3.2.9.2 CIM_FCPort

Created By : External
Modified By : External
Deleted By : External
Standard Names: The PermanentAddress Property follows the requirements in 6.2.4.5.2

Class Mandatory: true

Table 330: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ElementName		string	Port Symbolic Name
UsageRestriction		uint16	Shall be 3 for ports restricted to Back-end only or 4 if the port is unrestricted
OperationalStatus		uint16[]	
Speed		uint64	
MaxSpeed		uint64	Port Supported Speed from HBA API.
PortType		uint16	"Unknown" = 0, "Other" = 1, "N" = 10, "NL" = 11, "F/NL" = 12, "Nx" = 13, "E" = 14, "F" = 15, "FL" = 16, "B" = 17, "G" = 18.
LinkTechnology		uint16	
SupportedMaximumTransmission-Unit		uint64	
Optional Properties/Methods			
PortNumber		uint16	
PermanentAddress	CD	string	PermanentAddress is optional when used as a backend port in a device. This may be overridden in profiles that use this subprofile.
NetworkAddresses		string[]	For Fibre Channel end device ports, the Fibre Channel ID
SupportedCOS		uint16[]	
ActiveCOS		uint16[]	
SupportedFC4Types		uint16[]	
ActiveFC4Types		uint16[]	
ActiveMaximumTransmissionUnit		uint64	

8.2.3.2.9.3 CIM_HostedAccessPoint

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

Class Mandatory: true

Table 331: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.3.2.9.4 CIM_ProtocolControllerForPort

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 332: SMI Referenced Properties/Methods for CIM_ProtocolControllerForPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalPort	
Antecedent		CIM_ProtocolController	

8.2.3.2.9.5 CIM_SCSIProtocolController

Represents a 'LUN Map'
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 333: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
Optional Properties/Methods			
ElementName		string	
OperationalStatus		uint16[]	
MaxUnitsControlled		uint32	

8.2.3.2.9.6 CIM_SCSIProtocolEndpoint

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

Class Mandatory: true

Table 334: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
OtherTypeDescription		string	
ConnectionType		uint16	Shall be 2 (Fibre Channel)
Role		uint16	Shall be 2 (Initiator) or 4 (Both Initiator and Target)

8.2.3.2.9.7 CIM_SystemDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 335: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.3.2.10 Related Standards

Table 336: Related Standards for FC Initiator Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

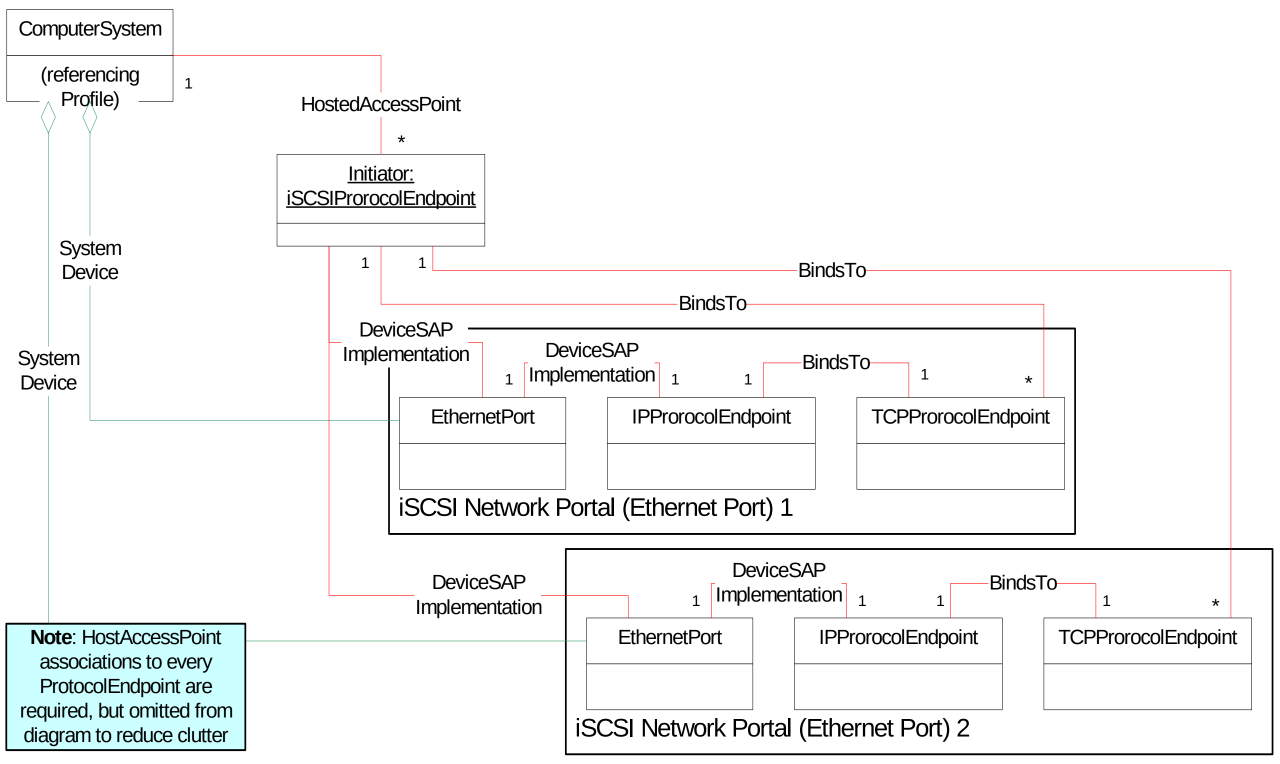
8.2.3.3 iSCSI Initiator Port Subprofile

8.2.3.3.1 Description

Other port subprofiles have a single physical port (LogicalPort subclass) associated with each SCSI initiator (SCSIProtocolEndpoint). iSCSI allows multiple connections (each with a single Ethernet port) in a session that acts as a SCSI initiator. This subprofile includes the subset of classes that model the SCSI initiator and its relationship to logical classes that model physical elements (Ethernet ports).

Table 64 depicts a configuration with an initiator with two Ethernet ports that are part of a single session that acts as a SCSI initiator. The Ethernet ports (referred to in iSCSI literature as Network Portals) are modeled as instances of EthernetPort, IPProtocolEndpoint, and TCPProtocolEndpoint with 1-1 cardinality. These ports are in the initiator side, the target ports are not required in this subprofile. Note that all ProtocolEndpoint instances need a HostAccessPoint association to the ComputerSystem, some are omitted to keep the diagram less cluttered.

Figure 64: iSCSI Initiator Port Instance Diagram



8.2.3.3.2 Durable Names and Other Correlatable ids of the Subprofile

LogicalDevice.Name is the name of a SCSI Logical Unit as defined in the Correlateable Names section

8.2.3.3.3 Health and Fault Management Considerations

Table 337: EthernetPort OperationalStatus

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

8.2.3.3.4 Dependencies on Profiles, Subprofiles and Packages

None

8.2.3.3.5 Extrinsic Methods of the Subprofile

None

8.2.3.3.6 Client Considerations and Recipes

8.2.3.3.7 Registered Name and Version

iSCSI Initiator Ports version 1.1.0

8.2.3.3.8 CIM Server Requirements

Table 338: CIM Server Requirements for iSCSI Initiator Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.3.3.9 CIM Elements

Table 339: CIM Elements for iSCSI Initiator Ports

Element Name	Description
Mandatory Classes	
CIM_BindsTo (8.2.3.3.9.1)	
CIM_DeviceSAPImplementation (8.2.3.3.9.2)	
CIM_EthernetPort (8.2.3.3.9.3)	
CIM_HostedAccessPoint (8.2.3.3.9.4)	
CIM_IPProtocolEndpoint (8.2.3.3.9.5)	
CIM_SystemDevice (8.2.3.3.9.7)	
CIM_TCPProtocolEndpoint (8.2.3.3.9.8)	
CIM_iSCSIProtocolEndpoint (8.2.3.3.9.9)	
Optional Classes	
CIM_LogicalDevice (8.2.3.3.9.6)	
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_EthernetPort	Port Creation
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_EthernetPort AND SourceInstance.CIM_EthernetPort::OperationalStatus <> PreviousInstance.CIM_EthernetPort::OperationalStatus	CQL - Port Status Change
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_EthernetPort	Port Removal

8.2.3.3.9.1 CIM_BindsTo

Created By : External
Modified By : Extrinsic(s):
Deleted By : External
Class Mandatory: true

Table 340: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolEndpoint	
Dependent		CIM_ServiceAccessPoint	

8.2.3.3.9.2 CIM_DeviceSAPImplementation

Created By : External
Modified By : Extrinsic(s):
Deleted By : External

Class Mandatory: true

Table 341: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	
Dependent		CIM_ServiceAccessPoint	

8.2.3.3.9.3 CIM_EthernetPort

Created By : External
Modified By : External
Deleted By : External
Standard Names: The PermanentAddress Property follows the requirements in 6.2.4.5.2
Class Mandatory: true

Table 342: SMI Referenced Properties/Methods for CIM_EthernetPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
PortType		uint16	
OperationalStatus		uint16[]	
PermanentAddress	CD	string	

8.2.3.3.9.4 CIM_HostedAccessPoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 343: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_ServiceAccessPoint	

8.2.3.3.9.5 CIM_IPProtocolEndpoint

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

Class Mandatory: true

Table 344: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
OtherTypeDescription		string	
ProtocolIFType		uint16	
Optional Properties/Methods			
IPv4Address	CD	string	Maps to IMA_NETWORK_PORTAL_PROPERTIES, ipAddress
IPv6Address	CD	string	Maps to IMA_NETWORK_PORTAL_PROPERTIES, ipAddress

8.2.3.3.9.6 CIM_LogicalDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 345: SMI Referenced Properties/Methods for CIM_LogicalDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Name		string	
OperationalStatus		uint16[]	

8.2.3.3.9.7 CIM_SystemDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 346: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.3.3.9.8 CIM_TCPProtocolEndpoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 347: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
PortNumber	CD	uint32	
ProtocolIFType		uint16	

8.2.3.3.9.9 CIM_iSCSIProtocolEndpoint

Created By : Static
Modified By : Static
Deleted By : Static
Standard Names: The Name Property follows the requirements in 6.2.4.5.2
Class Mandatory: true

Table 348: SMI Referenced Properties/Methods for CIM_iSCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name	CD	string	
ProtocolIFType		uint16	Other
OtherTypeDescription		string	
ConnectionType		uint16	iSCSI
Role		uint16	Shall be 2 (Initiator)
Identifier		string	ISID

8.2.3.3.10 Related Standards

Table 349: Related Standards for iSCSI Initiator Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

DEPRECATED

8.2.3.4 Back End Ports Subprofile (DEPRECATED)

8.2.3.4.1 Description

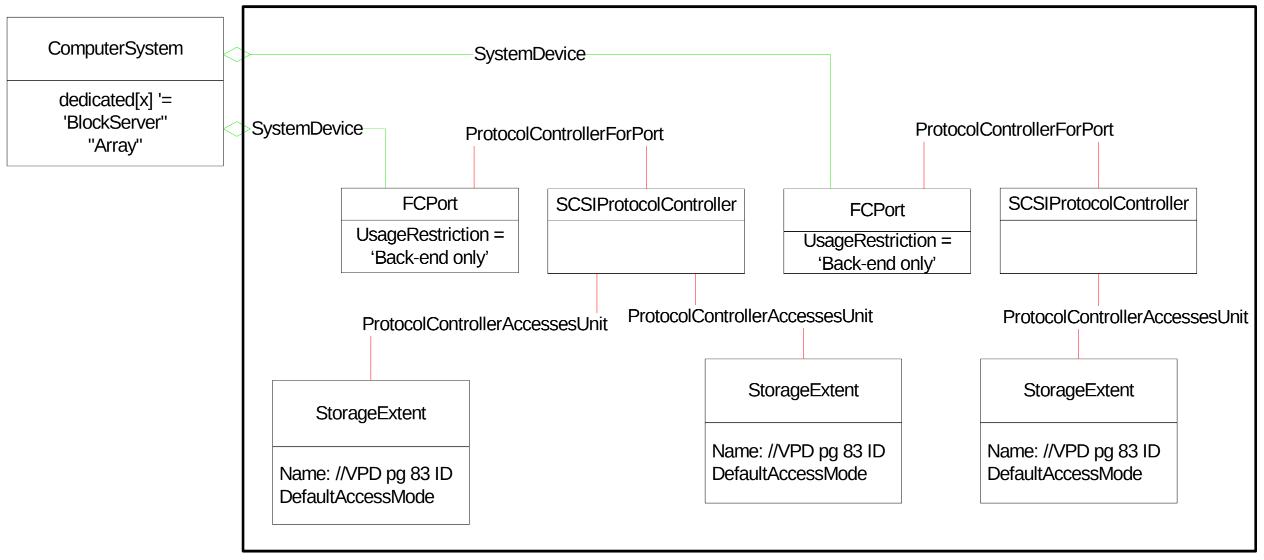
Some RAID systems provide interfaces to discover and manage the internal connections between the RAID processors and physical disks. For example, an array may have an interface to acquire and optimize the utilization of separate buses, loops, or fabrics to back-end storage. In this case, the ports to individual disks can be modeled similarly to a JBOD configuration as well as the ports on the RAID processors.

A property on FCPort called UsageRestriction is available to indicate whether the controller is providing a front end (target) or back end (initiator) interface.

The RAID controller itself has front-end ports (connected to customer hosts or switches) and back-end ports (connected to the internal disks). Figure 65: "Back-end Ports Instance" shows an instance diagram for three disks (StorageExent only shown) in an array, connected by a FC loop. The full model for the disk is shown in 8.2.8.13, "Disk Drive Subprofile (DEPRECATED)".

Instance Diagram

Figure 65: Back-end Ports Instance


ComputerSystem
dedicated[x] '= 'BlockServer" "Array"
SystemDevice
SystemDevice
ProtocolControllerForPort
ProtocolControllerForPort
FCPort
UsageRestriction = 'Back-end only'
SCSIProtocolController
FCPort
UsageRestriction = 'Back-end only'
SCSIProtocolController
ProtocolControllerAccessesUnit
ProtocolControllerAccessesUnit
ProtocolControllerAccessesUnit
StorageExtent
Name: //VPD pg 83 ID
DefaultAccessMode
StorageExtent
Name: //VPD pg 83 ID
DefaultAccessMode
StorageExtent
Name: //VPD pg 83 ID
DefaultAccessMode

8.2.3.4.2 Health and Fault Management

Not defined in this standard.

8.2.3.4.3 Cascading Considerations

Not defined in this standard.

8.2.3.4.4 Dependencies on Profiles, Subprofiles, and Packages

None.

8.2.3.4.5 Methods of the Profile

None.

8.2.3.4.6 Client Considerations and Recipes

None.

8.2.3.4.7 Registered Name and Version

Backend Ports version 1.0.2

8.2.3.4.8 CIM Server Requirements

Table 350: CIM Server Requirements for Backend Ports

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.3.4.9 CIM Elements

Table 351: CIM Elements for Backend Ports

Element Name	Description
Mandatory Classes	
CIM_FCPort (8.2.3.4.9.1)	
CIM_ProtocolControllerAccessesUnit (8.2.3.4.9.2)	
CIM_ProtocolControllerForPort (8.2.3.4.9.3)	
CIM_SCSIProtocolController (8.2.3.4.9.4)	
CIM_StorageExtent (8.2.3.4.9.5)	
CIM_SystemDevice (8.2.3.4.9.6)	

8.2.3.4.9.1 CIM_FCPort

Class Mandatory: true

Table 352: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
Optional Properties/Methods			
Speed		uint64	

8.2.3.4.9.2 CIM_ProtocolControllerAccessesUnit

Class Mandatory: true

Table 353: SMI Referenced Properties/Methods for CIM_ProtocolControllerAccessesUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolController	
Dependent		CIM_LogicalDevice	
DeviceNumber		string	
Optional Properties/Methods			
TargetControllerNumber		string	

8.2.3.4.9.3 CIM_ProtocolControllerForPort

Class Mandatory: true

Table 354: SMI Referenced Properties/Methods for CIM_ProtocolControllerForPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolController	
Dependent		CIM_LogicalPort	
Optional Properties/Methods			
AccessPriority		uint16	

8.2.3.4.9.4 CIM_SCSIProtocolController

Class Mandatory: true

Table 355: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Optional Properties/Methods			
ElementName		string	
OperationalStatus		uint16[]	

8.2.3.4.9.5 CIM_StorageExtent

Class Mandatory: true

Table 356: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	

Table 356: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
SystemName		string	
CreationClassName		string	
DeviceID		string	
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
BlockSize		uint64	
Primordial		boolean	
Optional Properties/Methods			
Name		string	VPD 83 identifier for this volume (ideally a LUN WWN)
NumberOfBlocks		uint64	
IsBasedOnUnderlyingRedundancy		boolean	

8.2.3.4.9.6 CIM_SystemDevice

Class Mandatory: true

Table 357: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.3.4.10 Related Standards

Table 358: Related Standards for Backend Ports

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.8.0	DMTF

DEPRECATED

8.2.4 CIM Server Related Profiles

8.2.4.1 Server Profile

8.2.4.1.1 Description

A CIM Server is anything that supports the CIM-XML protocol and supports the basic read functional profile as defined by the CIM Operations over HTTP specification.

The Server Profile is mandatory for all compliant SMI-S Servers.

The Object Manager part of the model defines the capabilities of a CIM Object Manager based on the communication mechanisms that it supports.

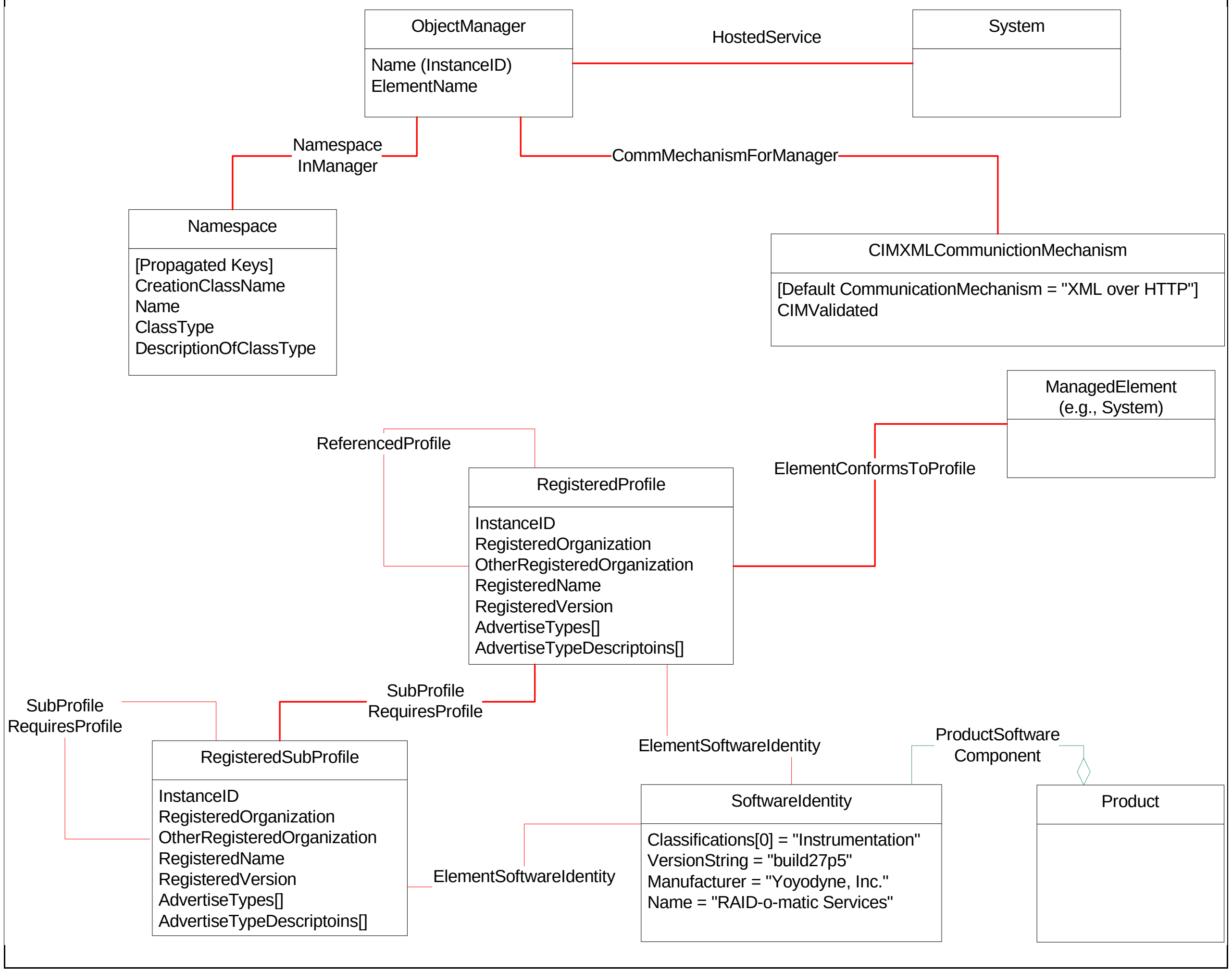
The namespace model of the Server Profile describes the namespaces managed by the Object Manager and the type information contained within the namespace. The main information provided in the namespace part of the model is the namespace itself and its association to the ObjectManager.

The RegisteredProfile part of the model is used to specify the Profiles supported by the Object Manager. It also includes the specification of subprofiles that are supported by the profile.

In this section there are references to the InteropNamespace and the use of the InteropNamespace for finding RegisteredProfiles and other related classes associated with the Server Profile. The InteropNamespace refers to the first namespace found in the InteropSchemaNamespace attribute of the SLP Template.

8.2.4.1.1.1 Instance Diagram

Figure 66: Server Model



A Server is modeled as a System with a HostedService association to an ObjectManager. The ObjectManager is subclassed from Service.

It is mandatory that all namespaces supported by the Server be identified (the Namespace class) and associated to the ObjectManager via the NamespaceInManager association

Note: Most classes of the Server Profile (as shown in Figure 66: "Server Model") are in the Interop Namespace. with the exception of the “ManagedElement” that is referenced from the RegisteredProfile. This makes traversing the Server Profile relatively simple. The only time a traversal may require crossing namespaces is when following the “ElementConformsToProfile” association.

The communication protocols supported by the ObjectManager should also be identified. Specifically, the CIMXMLCommunicationMechanism shall be present for standard communication support for clients. This class is associated to the ObjectManager via the CommMechanismForManager association.

The next set of classes and associations deal with Profiles supported by the ObjectManager. A Profile is modeled using the RegisteredProfile class. One instance is created for each ManagedElement that is covered by a profile and is managed by the Server. The RegisteredProfile instances can be found by enumerating RegisteredProfiles within the interop namespace. A client would find all profiles supported by the Server by enumerating RegisteredProfiles, enumerating RegisteredSubprofiles and subtracting the second list from the first list. This will yield the list of Profiles supported by the ObjectManager.

For each Profile instance, the subprofiles that have been implemented (for the instance) should be identified via the SubprofileRequiresProfile association. Subprofiles are modeled using the RegisteredSubProfile class.

In addition, the ElementConformsToProfile association ties the “top-level” Profile (RegisteredProfile) to the scoping ManagedElements. These ManagedElements are typically ComputerSystems or AdminDomains.

A single ManagedElement may have zero or more ElementConformsToProfile associations to RegisteredProfiles. Regardless of the number of associated RegisteredProfiles the ManagedElement represents one set of resources. So for example, consider a ManagedElement that is a System that supports both the Array and Storage Virtualizer profiles. If one asks for the total amount of mapped capacity, the answer applies to both Array and Virtualizer and is not additive.

Each RegisteredProfile and RegisteredSubprofile instance shall be associated to one (or more) SoftwareIdentity instances containing information about the software packages required to deploy the instrumentation (including providers). These are associated using ElementSoftwareIdentity. SoftwareIdentity instance may optionally be associated to Product instances representing a software product.

8.2.4.1.1.2 Use of model fields to Populate the SLP template

The data used to populate the SLP template for advertising SMI-S profiles is found in the CIM Server profile. The SLP template fields are populated as follows:

template-url-syntax: =string
The following quotation is from the “WBEM SLP Template v1.0.0.
(http://www.dmtf.org/standards wbem/wbem.1.0.en)

“The template-url-syntax shall be the WBEM URI Mapping of the location of one service access point offered by the WBEM Server over TCP transport. This attribute shall provide sufficient addressing information so that the WBEM Server can be addressed directly using the URL.”

The WBEM URI Mapping is defined in the WBEM URI Mapping Specification 1.0.0 (DSP0207). Example: (template-url-syntax=https://localhost:5989 [^])

service-hi-name: ObjectManager.ElementName

service-hi-description: ObjectManager.Description

service-id: ObjectManager.Name

Service-location-tcp: The location of one service access point offered by the CIM Server over TCP transport. This attribute shall provide sufficient addressing information that the CIM Server can be addressed directly using only this attribute.

CommunicationMechanism: ObjectManagerCommunicationMechanism.CommunicationMechanism

OtherCommunicationMechanism:
ObjectManagerCommunicationMechanism.OtherCommunicationMechanism

InteropSchemaNamespace: Namespace.Name for the InteropNamespace

ProtocolVersion: ObjectManagerCommunicationMechanism.Version

FunctionalProfilesSupported: ObjectManagerCommunicationMechanism.FunctionalProfilesSupported

FunctionalProfileDescriptions: ObjectManagerCommunicationMechanism.FunctionalProfileDescriptions

MultipleOperationsSupported: ObjectManagerCommunicationMechanism.MultipleOperationsSupported

AuthenticationMechanismSupported: ObjectManagerCommunicationMechanism.AuthenticationMechanismsSupported

OtherAuthenticationDescription: ObjectManagerCommunicationMechanism.AuthenticationMechanismDescriptions

Namespace: Namespace.Name for each Namespace instance supported

Classinfo: Namespace.Classinfo for each Namespace instance

RegisteredProfilesSupported: The concatenation of:

- RegisteredProfile.RegisteredOrganization;
- RegisteredProfile.RegisteredName;
- RegisteredProfile.RegisteredName (where the second RegisteredName is the name of a subprofile that is identified for SLP advertisement).

8.2.4.1.1.3 HTTP Security Background

Section 4.4 of “Specification for CIM Operations over HTTP, Version 1.1” from DMTF describes the requirements for CIM clients and servers. The authentication methods referred to in the above specification are described in the IETF RFCs 1945 and 2068, “Hypertext Transfer Protocol -- HTTP/ 1.0(1.1)” and IETF RFC 2069 “An Extension to HTTP: Digest Access Authentication””. The Transport Layer Security Protocol Version 1.0 (TLS) is defined by ETF RFC 4346, which contains specifications for both versions 1.0 and 1.1. The Secure Sockets Layer 3.0 (SSL 3.0) protocol specification can be downloaded from HTTP://wp.netscape.com/end/ssl3/.

Section 4.4 of “Specification for CIM Operations over HTTP, Version 1.1" defines additional requirements for HTTP authentication, above those found in HTTP 1.1 [RFC 2068], or the HTTP authentication documents [RFC 2069, RFC 2617]. HTTP authentication generally starts with an HTTP client request, such as “GET Request-URI” (where Request-URI is the resource requested). If the client request does not include an “Authorization” header line and authentication is required, the server responds with a “401 unauthorized” status code, and a “WWW-Authenticate” header line. The HTTP client shall then respond with the appropriate “Authorization” header line in a subsequent request. The format of the “WWW-Authenticate” and “Authorization” header lines varies depending on the type of authentication required: basic authentication or digest authentication. If the authentication is successful, the HTTP server will respond with a status code of “200 OK”.

Basic authentication involves sending the user name and password in the clear, and should only be used on a secure network, or in conjunction with a mechanism that ensures confidentiality, such as TLS. Digest authentication sends a secure digest of the user name and password (and other information including a nonce value), so that the password is not revealed. “401Unauthorized” responses should not include a choice of authentication

SSL 3.0 and TLS provide both confidentiality and integrity in communication, which precludes eavesdropping, tampering, and message forgery. While TLS 1.1 and TLS 1.0 are based on SSL 3.0 and the differences between them are not dramatic, it is important to note that these differences are significant enough that TLS 1.1, TLS 1.0 and SSL 3.0 will not interoperate. However, both versions of TLS do provide mechanisms for backwards compatibility with the earlier versions.

Both TLS and SSL 3.0 package one key establishment, confidentiality, signature and hash algorithm into a "cipher suite." A registered 16-bit (4 hexadecimal digit) number, called the cipher suite index, is assigned for each defined cipher suite. For example, RSA key agreement, RSA signature, Triple Data Encryption Standard (3DES) using Encryption-Decryption-Encryption (EDE) and Cipher Block Chaining (CBC) confidentiality, and the Secure Hash Algorithm (SHA-1) hash is assigned the hexadecimal value {0x000A} for TLS. Note especially that TLS 1.1 requires (IEFT RFC 4346, Section 9 - Mandatory Cipher Suites): "In the absence of an application profile standard specifying otherwise, a TLS compliant application shall implement the cipher suite TLS_RSA_WITH_3DES_EBE_CBC_ SHA" described above.

The client always initiates the TLS and SSL 3.0 session and starts cipher suite negotiation by transmitting a handshake message that lists the cipher suites (by index value) that it will accept. The server responds with a handshake message indicating which cipher suite it selected from the list or an "abort" as described below. Although the client is required to order its list by increasing "strength" of cipher suite, the server may choose ANY of the cipher suites proposed by the client. Therefore, there is NO guarantee that the negotiation will select the strongest suite. If no cipher suites are mutually supported, the connection is aborted. When the negotiated options, including optional public key certificates and random data for developing keying material to be used by the cryptographic algorithms, are complete, messages are exchanged to place the communications channel in a secure mode.

SMI-S clients and servers may be attacked by setting up a false SMI-S server to capture userids and passwords or to insert itself as an undetected proxy between an SMI-S client and server. The most effective countermeasure for this attack is the controlled use of server certificates with SSL 3.0 or TLS, matched by client controls on certificate acceptance on the assumption that the false server will be unable to obtain an acceptable certificate. Specifically, this could be accomplished by configuring clients to always use SSL 3.0 or TLS underneath HTTP authentication, and only accept certificates from a specific local certificate authority. See 8.2.4.1.1.4 for requirements in this area. In the absence of this countermeasure, some protection can by obtained by limiting the scope of SMI-S discovery, including SLP, by IP address range (this involves client configuration plus SLP DA configuration, if any SLP DA is used), and the use of firewalls to block ports used by SMI-S and SLP in order to prevent SMI-S access to/from points outside a protected area of the network.

8.2.4.1.1.4 HTTP Security

This section specified security requirements on the protocol for communication between a Client and an SMI-S Server, but not the mechanism of authentication used by the SMI-S Server.

Client authentication to the SMI-S Server is based on an authentication service. Differing authentication schemes may be supported, including host-based authentication, Kerberos, PKI, or other.

For the purposes of SMI-S, basic strength ciphersuites include 512-bit (or longer) asymmetric algorithms (RSA or Diffie-Hellman), combined with 40-bit (or longer) symmetric algorithms (Triple DES, IDEA, RC4-128) and either SHA-1 or MD5. Enhanced strength ciphersuites combine 1 024-bit (or longer) asymmetric algorithms (RSA or Diffie-Hellman) with 128-bit (or longer) symmetric algorithms (Triple DES, IDEA, RC4-128, AES) and either SHA-1 or MD5.

General Requirements

The following are general requirements for the support of security when using HTTP.

a) SMI-S Servers and Clients shall conform to section 4.4 of “Specification for CIM Operations over HTTP, Version 1.1”.

b) HTTP Basic Authentication shall be implemented. HTTP Digest Authentication should be implemented.

c) To minimize compromising user identities, and credentials such as passwords, implementers should use HTTP Basic Authentication ONLY in conjunction with SSL 3.0 or TLS and an enhanced strength ciphersuite.

d) Where neither SSL 3.0 or TLS are used, or where they are used with a basic strength ciphersuite, implementers should utilize HTTP Digest Authentication.

e) To ensure a minimum level of security and interoperability between implementations, support for the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher suite shall be included in all implementations. Implementers are free to include additional cipher suites.

EXPERIMENTAL

When such cipher suites are supported, SSL_RSA_WITH_3DES_EDE_CBC_SHA for SSL 3.0 and TLS_RSA_WITH_3DES_EDE_CBC_SHA for TLS shall be supported at a minimum. Additionally, Table 359 identifies the SSL and TLS cipher suites (in order of descending preference) that should be supported and used by SMI-S implementations:

Table 359: SMI-S Preferred Cipher Suites

TLS 1.0 & 1.1	SSL 3.0
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	SSL_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	SSL_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA

The order of the cipher suites in Table 359 is the order of preference (i.e., cipher suites higher in the table are preferred over those lower in the table) when multiple cipher suites are offered unless overridden by local security policy. Within each pair of cipher suites, the "_DHE_" suite uses a Diffie-Hellman exchange to provide forward secrecy so that future disclosure of the RSA key(s) used will not compromise previous secured traffic.

Recognizing that implementers are likely to start with the least preferred 3DES-based cipher suites and then consider the AES suites, it is important to note that the National Institute of Standards and Technology (NIST) is currently encouraging transition to AES. Implementers should be aware that AES_128 is not only a stronger encryption algorithm than 3DES, but also that AES_128 tends to be more efficient and of higher performance when implemented.

For these reasons, if an SMI-S implementation supports 3DES, then support of AES_128 is strongly recommended. It is reasonable to expect that a future version of SMI-S will include a mandatory AES_128-based cipher suite.

EXPERIMENTAL

f) If no enhanced strength ciphersuite is supported, then HTTP Digest Authentication shall be implemented.

g) A user identity and credential used with one type of HTTP Authentication (i.e., Basic or Digest) shall not ever be subsequently used with the other type of HTTP Authentication. To avoid compromising the integrity of a stronger scheme, established good security practices avoids the reuse of identity & credential information across schemes of different strengths.

h) SSL 3.0 and TLS 1.0 shall be supported; TLS 1.1 is currently an allowed option that is strongly recommended. SSL support is currently required for backwards compatibility as described in *Appendix E of RFC 4346*.

EXPERIMENTAL

Additionally, SMI-S implementations shall have configurable mechanisms to only use cipher suites that include RSA, DHE-RSA, or DHE-DSS key establishment mechanisms and RSA or DSA signature mechanisms (i.e., only certificate-based cipher suites). These mechanisms shall further prevent the negotiation of the "EXPORT" cipher suites (identified in Section A.5 of RFC 4346 as TLS 1.1 must not negotiate cipher suites; in addition, SMI-S prohibits use of "EXPORT" cipher suites with SSL 3.0 and TLS 1.0).

Although DES is an allowed cipher when used with the appropriate key exchange mechanism, DES is vulnerable to brute-force attacks. When such an attack is a concern, a stronger cipher should be used.

It is important to recognize that maintaining security often requires changing requirements to reflect advances in technology, discovery of vulnerabilities, and defenses against new attacks. Consequently, it is expected that future versions of SMI-S will require TLS 1.1 to be implemented, deprecate support for SSL 3.0, deprecate cipher suites that include DES (any key size) as the cipher, and deprecate cipher suites that include MD5 as the hash.

EXPERIMENTAL

i) Clients that fail to contact an SMI-S server via HTTP over SSL 3.0 or TLS on TCP port 5 989 should retry with HTTP on TCP port 5 988 if their security policy allows it.

j) In order for Clients and Servers to communicate, they need to be using a consistent approach to security. It is possible for properly configured Clients and Servers to fail to communicate if one is relying upon port 5 989 and the other on port 5 988.

k) Servers can accelerate discovery that a secure channel is needed by responding to HTTP contacts on TCP port 5 988 with an HTTP REDIRECT to the appropriate HTTPS: URL (HTTP over SSL or TLS on TCP port 5 989) to avoid the need for clients to timeout the HTTP contact attempt. Clients should honor such redirects in this situation.

Requirements for the support of HTTP Realm

The relationship of the realm-value to an authentication service, and one or more sets of user identity and credential, is determined separately by the configuration of each SMI-S client, and configurations may differ between multiple SMI-S clients in the same system. The means of creating this configuration in the SMI-S client is outside of the scope of this specification. The client configuration is expected to a contain at least a default set of user identity and credential per realm-value. When the configuration associates a single realm-value with multiple sets of user identity and credential, the basis on which a single set is selected is also outside of the scope of this specification (and may include considerations such as the need to assert elevated privilege at the server to perform specific operations.)

Where the Realm field is not used, or the realm-value is unrecognized, the SMI-S Client may use means outside of the scope of this specification to identify the user identity and credential to be used, including the use of information obtained during Service Discovery.

For this revision of the specification, it is recommended that a single realm-value per SMI-S Server be defined by means such as a configuration file. In future revisions, the definition of multiple and dynamic user identities and credentials per SMI-S Server will be addressed, and may use other communication methods in addition to, or in place of, the Realm field.

a) The Realm field defined by *HTTP Version 1.1* (see RFC 2617 section 1.2 and RFC 2616) shall be implemented by the SMI-S Server, and should be used to identify to the Client the authentication service to be used to access the server.

b) The realm-value contains information to help determine which specific user identity and credential (e.g., user ID & password) and are to be used with the authentication service, but shall not contain any portion of an identity or a credential itself.

c) The exact form of the authentication service is not defined by SMI-S, and may either be part of the configuration of an SMI-S Server, or may involve an external entity such as a RADIUS server. A single authentication service may be utilized by multiple SMI-S Servers. Realm-values shall be unique throughout the scope of the authentication service.

d) When provided, the realm-value shall meet all of the requirements contained in RFC 2616 and RFC 2617, with the exception of the specific requirement in section 3.2.1 of RFC 2617 that the realm-value “be displayed to users”. In SMI-S, the realm-value may be handled by the SMI-S Client without reference to a user.

e) Where no format for the realm-value has been defined by other standards or conventions, and where an authentication is handled autonomously by an SMI-S server, then a string in the format defined in “SMI-S defined format for HTTP Realm ()” in 8.2.4.1.1.4 is recommended.

f) Where a single authentication service is utilized by multiple SMI-S Servers, the SMI-S recommended format defined in ““SMI-S defined format for HTTP Realm” on page 370 in 8.2.4.1.1.4 should not be used, and use of SHA-1 in the creation of realm-values is recommended.

SMI-S defined format for HTTP Realm

The format is based on components of the definition of the Uniform Resource Identifier (URI) in RFC 2396 and extended in RFC 2732, and is described using the BNF-like grammar of those documents as:

[1*(unreserved) "."] "smis@" host

where:

- unreserved is as defined in section 2.3 of RFC 2396
- "." is a dot
- "smis@" is a string literal
- host is as defined in section 3 of RFC 2732

The combination of the unreserved and host portions should be defined in a manner that allows an administrator to quickly identify a specific SMI-S Server in his configuration. Note that some portion of unreserved could be generated randomly in the SMI-S Server to reduce the chance of accidental realm collisions.

An example of the use of the recommended format defined above is as follows: Consider a single server system labelled Server6 owned by Widgets Inc. (owner of the example.com domain) that hosts two SMI-S Servers, one from Acme Inc., and the other from XYZ Ltd. The realm-value reported by the Acme SMI-S Server might be "ug723.acme.net.smis@server6.example.com". In the configuration of a specific SMI-S client accessing the Acme SMI-S Server, this realm-value might identify a server-specific authentication service and a user identity of "arrayuser74" and a password of "YT56z". Similarly, the realm-value reported by the XYZ Ltd SMI-S Server might be "bx48d.xyz.co.uk.smis@server6.example.com". In the configuration of a different SMI-S client accessing the XYZ SMI-S Server, this realm-value might identify a SMIS-server-specific authentication service and a user identity of "42fred" and a password of "OTH3afa".

Certificate Usage with SSL 3.0 and TLS

Within SMI-S, SSL 3.0 and TLS are used with public key certificates (or identity certificates) for authentication. These X.509 certificates conform to the format and semantics specified in IETF RFC 3280 and use a digital signature to bind together a public key with an identity. These signatures will often be issued by a certification authority (CA) associated with an internal or external public key infrastructure (PKI); however, an alternate approach uses self-signed certificates (the certificate is digitally signed by the very same key-pair whose public part appears in the certificate data). The trust models associated with these two approaches are very different. In the case of PKI certificates, there is a hierarchy of trust and a trusted third-party that can be consulted in the certificate validation process, which enhances security at the expense of increased complexity. The self-signed certificates can be used to form a web of trust (trust decisions are in the hands of individual users/administrators), but is considered less secure as there is no central authority for trust (e.g., no identity assurance or revocation). This reduction in overall security, which may still offer adequate protections for some environments, is accompanied by an easing of the overall complexity of implementation.

With PKI certificates, it is often necessary to traverse the hierarchy or chain of trust in search of a root of trust or trust anchor (a trusted CA). This trust anchor may be an internal CA, which has a certificate signed by a higher ranking CA, or it may be the end of a certificate chain with the highest ranking CA. This highest ranking CA provides the ultimate in attestation authority in a particular PKI scheme and its certificate, known as a root certificate, can only be self-signed. Establishing a trust anchor at the root certificate level, especially for commercial CAs, can have undesirable side effects resulting from the implicit trust afforded all certificates issued by that commercial CA. Ideally the trust anchor should be established with the lowest ranking CA that is practical.

The remainder of this subsection provides certificate-related requirements that apply to any SMI-S implementation that supports SSL 3.0 or TLS.

Certificate Usage with SSL 3.0 and TLS: Requirements

a) Require support for existing common practice for certificate usage.

- SMI-S uses X.509 version 3 public key certificates that are conformant with the Certificate and Certificate Extension Profile defined in Section 4 of IETF RFC 3280. This profile specifies the mandatory fields that shall be included in the certificate as well as optional fields and extensions that may be included in the certificate.
- Server certificates shall be supported and client certificates MAY be supported. A server certificate is presented by the server to authenticate the server to the client; likewise, a client certificate is presented by the client to authenticate itself to the server. For public web sites offering secure communications via SSL 3.0 or TLS, server certificate usage is quite common, but client certificates are rarely used.
- SMI-S clients and servers shall perform basic path validation, extension path validation, and CRL validation as specified in Section 6 of IETF RFC 3280 for all presented certificates. These validations include, but are not limited to, the following:
  - The certificate is a validly constructed certificate
  - The signature is correct for the certificate
  - The date of its use is within the validity period (i.e., it has not expired)
  - The certificate has not been revoked (applies only to PKI certificates)
  - The certificate chain is validly constructed (considering the peer certificate plus valid issuer certificates up to the maximum allowed chain depth; applies only to PKI certificates).
- When SMI-S clients and servers use certificate revocation lists (CRL), they shall uses X.509 version 2 CRLs that are conformant with the CRL and CRL Extension Profile defined in Section 5 of IETF RFC 3280.

- When PKI certificates and self-signed certificates are used together in a single management domain, it is important to recognize that the level of security is lowered to that afforded by self-signed certificates.

b) Allow customers to enforce their own certificate usage and acceptance policies.

- All certificates identifying SMI-S management entities and their associated private keys shall be replaceable. SMI-S clients and servers shall either 1) have the ability to import an externally generated certificate and corresponding private key or 2) have the ability to generate and install a new self-signed certificate along with its corresponding private key.
- When PKI certificates are used by SMI-S clients and servers, the implementations shall include the ability to import, install/store, and remove the CA root certificates; support for multiple trusted issuing CAs shall be included. CA certificates are used to verify that a certificate has been signed by a key from an acceptable certification authority.
- To facilitate the use of certificates, SMI-S implementations should include configurable mechanisms that allow for one of the following mutually exclusive operating modes to be in force at any point in time for end-entity certificates (i.e., not CA certificates):
    - Unverifiable end-entity (self-signed) certificates are automatically installed as trust anchors when they are presented; such certificates shall be determined to not be CA root certificates prior to being installed as trust anchors and shall not serve as trust anchors to verify any other certificates. If a CA certificate is presented as an end-entity certificate in this mode, it shall be rejected. For SMI-S clients, a variant of this option, which consults the user before taking action, should be implemented and used when possible. NOTE: The use of this operating mode should be limited to a learning or enrollment period during which communication is established with all other SMI-S systems with which security communication is desired. Use of a timeout to force automatic exit from this mode is recommended.
    - Unverifiable end-entity (self-signed) certificates can be manually imported and installed as trust anchors (in a fashion similar to manually importing and installing a CA root certificate), but they are not automatically added when initially encountered. Administrative privilege may be required to import and install an end-entity certificate as a trust anchor. NOTE: This is considered the normal operating mode.
- All certificate acceptance policies for SMI-S clients and servers shall be configurable. The configurable mechanisms determine how the SMI-S implementation handles presented certificates. Under normal operating mode, SMI-S servers should not accept certificates from unknown trust authorities (i.e., the CA root certificate has not been installed).
- When self-signed certificates are used in conjunction with SLPv2, the trustworthiness of these certificates becomes an important factor in preventing SLPv2 from becoming an attack vector.

c) Default to facilitating interoperability where not specifically disallowed by security policy.
Interactive clients should provide a means to query the user about acceptance of a certificate from an unrecognized certificate authority (no corresponding CA root certificate installed in client), and accept responses allowing use of the certificate presented, or all certificates from the issuing CA. Servers should not support acceptance of unrecognized certificates; it is expected that a limited number of CAs will be acceptable for client certificates in any site that uses them.

Pre-configuring root certificates from widely used CAs is optional, but simplifies initial configuration of certificate-based security, as certificates from those CAs will be accepted. These CA root certificates can be exported from widely available web browsers.

d) Require support for certificate acquisition from and revocation by common PKI/CA software.
All interfaces for certificate configuration in (b) and (c) above shall support the following certificate formats:

- DER encoded X.509
International Telecommunications Union Telecommunication Standardization Sector (ITU-T), Recommendation X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks, May 2000.
Specification and technical corrigenda can be obtained from:
http://www.itu.int/publications

- Base64 encoded X.509 (often called PEM)
N. Freed and N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, IETF RFC 2045, November 1996, Section 6.8.
Available at: http://`www.ietf.org/rfc/rfc2045.txt`

- PKCS#12
RSA Laboratories, PKCS #12: Personal Information Exchange Syntax, Version 1.0, June 1999. Specification and Technical Corrigendum. Available at:
`http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/index.html`

All certificate validation software MUST support local certificate revocation lists, and at least one list per CA root certificate supported. Support is REQUIRED for both DER encoded X.509 and Base64 encoded X.509 formats, but this support may be provided by using one format in the software and providing a tool to convert lists from the other format. OCSP and other means of immediate online verification of certificate validity are OPTIONAL, as connectivity to the issuing Certificate Authority cannot be assured.

e) Allow security policy control to be restricted to security administrators.
All certificate interfaces required above MUST support access restrictions that permit access only by suitably privileged administrators. A suitably privileged security administrator MUST be able to disable functionality for acceptance of unrecognized certificates described in (c) above.

The above requirements can be satisfied via appropriate use of the readily-available OpenSSL toolkit software (www.openssl.org). Support for PKCS#7 certificate format was deliberately omitted from the requirements. This format is primarily used for online interaction with certificate authorities; such functionality is not appropriate to require of all SMI-S storage management software, and tools are readily available to convert PKCS#7 certificates to or from other certificate formats.

8.2.4.1.2 Health and Fault Management

Not defined in this standard.

8.2.4.1.3 Cascading Considerations

Not defined in this standard.

Supported Subprofiles and Packages

Table 360: Supported Subprofiles for Server

Registered Subprofile Names	Mandatory	Version
Object Manager Adapter	No	1.1.0
Indication	No	1.1.0

8.2.4.1.4 Methods of the Profile

None.

8.2.4.1.5 Client Considerations and Recipes

Applicability of Security Considerations

The security requirements for HTTP implementation given in 8.2.4.1.1.4, "HTTP Security" apply to both SMI-S servers and clients. An SMI-S client shall comply with all security requirements for HTTP specified in 8.2.4.1.1.4, "HTTP Security" that are applicable to clients.

SMI-S Client support for HTTP security is *REQUIRED*. This includes the following requirements applicable to clients:

- SSL 3.0 and TLS shall be supported.
- HTTP Basic Authentication shall be supported. HTTP Digest Authentication should be supported.
- HTTP Realms shall be supported.
- All certificates, including CA Root Certificates used by clients for certificate validation, shall be replaceable.
- The DER encoded X.509, Base64 encoded X.509 and PKCS#12 certificate formats shall be supported.
- Certificate Revocation Lists shall be supported in the DER encoded X.509 and Base64 encoded X.509 formats.

The above list is not comprehensive; see Section 8.2.4.1.1.4 for the complete requirements. If there is any conflict between this text and Section 8.2.4.1.1.4, the text in Section 8.1.4.1.1.2 is the final specification of the requirements.

Using the CIM Server Model to Determine SNIA Profiles Supported

All SNIA Profiles require the implementation of the Server Profile as part of the CIM Server. This allows a client to determine which SNIA Profiles are supported by the a proxy, embedded or general purpose SMI-S Server. SMI-S clients can use SLP to search for services that support SNIA profiles. Indeed, a client may restrict its search to specific types of SNIA profiles. The client would get a response for each CIM Server service that supports a SNIA profile. From the responses, the client should use the “service-id” to determine the unique CIM Servers it is dealing with.

For each CIM Server, the client can determine the types of entities supported by inspecting the RegisteredProfilesSupported attribute returned for the SLP entries. This identifies the types of entities (e.g., devices) supported by the CIM Server.

The Client may determine more detail on the support for the Profiles by going to the service advertised for the CIM Server and inspecting the RegisteredProfiles maintained in the server profile. This would be done by enumerating RegisteredProfiles and RegisteredSubprofiles within the interop namespace. By inspection of the actual profile instances, the client can determine the SNIA version (RegisteredVersion) of profile, associated namespaces and associated managed elements (e.g., systems).

Using the CIM Server Model to Determine Optional Features supported

From the RegisteredProfiles within the namespace of the ObjectManager, a client can determine the “optional features” that are supported for the profile by following the SubprofileRequiresProfile association. This returns a set of RegisteredSubProfile instances that represent Subprofiles of the specific Profile instance. The name of the subprofile is scoped by the Profile. See individual Profile descriptions in this specification for the specific list of "optional subprofiles" supported. For a given profile instance there may be zero, one or many subprofiles. The optional subprofiles documented in this specification merely list the subprofiles that may be associated with the profile (via the SubprofileRequiresProfile association).

All Subprofiles that are supported by a Profile shall be directly associated to the Profile via the SubprofileRequiresProfile association. All subprofiles (either direct or indirect via subprofiles) shall be directly attached to the Profile. For example, the Array Profile instance can support two subprofiles: LUN Creation and Job Control. Both of these subprofiles would be directly attached to the Array Profile instance, even though the Job Control subprofile is actually a subprofile of LUN Creation.

Note: The RegisteredVersion property of subprofiles shall match the RegisteredVersion property of its parent Profile.

Recipe Assumptions

For discovery recipes, the following are assumed:

a) A top-level object (class instance) exists for each Profile, and

b) the client knows what the top level object is.

The top-level object for each of the SMI-S Profiles are:

- ComputerSystem: For Array, Storage (Media) Libraries, Virtualizers, Switches, and HBAs. This is the top-level ComputerSystem instance for the Profile (not the component ComputerSystem or the member ComputerSystem);
- AdminDomain: For Fabric and HostDiscoveredResources;
- ObjectManager: For Server.

The top-level object (class instance) is associated to the RegisteredProfile instance for the Profile via the ElementConformsToProfile association.

Note: Other ManagedElement instances may be associated to the RegisteredProfile, but the meaning and behavior of such associations are not defined by SMI-S and are not mandatory.

8.2.4.1.1.4, "HTTP Security" and its subsections contain security requirements, some of which are applicable to clients. All SMI-S shall satisfy every security requirement in that section and its subsection that is applicable to clients.

8.2.4.1.5.1 Find Servers Supporting a Given Profile

```
// DESCRIPTION
// A management application wishes to find all CIM Servers on a
// particular subnet that support one or more SMI-S profiles.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume CIM Servers have advertised their services (SrvReg)
// 2.Assume there may (or may not) be Directory Agents in the subnet
// 3.Assume no security on SLP discovery
// 4.#DirectoryList[] is an array of directory URLs
// 5.#ServiceList[] is an array of service agent URLs
// 6.#DirectoryEntries [] is an array of directory entry Structures.
//   The structure matches the “wbem” SLP Template (see Clause 5,
//   section 10).

// Step 1:  Set the Previous Responders List to the Null String.
#PRList = ““

// Step 2: Multicast a Service Request for a Directory Server Service.
//   This is to find Directory Agents in the subnet.
//
SrvRqst (
     #PRList,        // The Previous Responders list
     ”service:directory-agent” // Service type
     “DEFAULT”,          // The scope
     NULL,             // The predicate
     NULL)             // SLP SPI (security token)

// Step 3: Listen for Response from Directory Agent(s)
#DirectoryList[] = DAAdvert (
     BootTimestamp, // Time of last reboot of DA
     URL,      // The URL of the DA
     ScopeList,// The scopes supported by the DA
     AttrList,// The DA Attributes
     SLP SPI List,// SLP SPI (SPIs the DA can verify)
     Authentication Block)
// Iterate on Steps 2 & 3, until a response has been received or the client has
// reached a UA configured CONFIG_RETRY_MAX seconds.  If no DA if found,
// proceed to step 4.  If a DA is found, proceed to step 7.

// Step 4: Set the Previous Responders List to the Null String.
#SAPRList = ““

// Step 5: Multicast a Service Request for Service Agent Services. This
//   is to find Service Agents in the subnet that are not advertised
//   in a Directory.

SrvRqst (
     #SAPRList,        // The Previous Responders list
     “service:service-agent” // Service type
     “DEFAULT”,          // The scope
     “(Service-type=WBEM)”,        // The predicate
     NULL)             // SLP SPI (security token)

// Step 6: Listen for Response from Service Agent(s)
#SAList[] = SAAdvert (
     URL,      // The URL of the SA
     ScopeList,// The scopes supported by the SA
     AttrList,// The SA Attributes
     Authentication Block)
// Iterate on Steps 5 & 6, until a response has been received or the client has
// reached a UA configured CONFIG_RETRY_MAX seconds.  If no SA if found,
// Then record an error.   There are NO WBEM SAs.  Otherwise proceed to
// Step 8.

//Step 7: Unicast a Service Request to each of the DAs specifying
//   a query predicate to select CIM Servers that support SNIA profiles
//   and listen for responses.
for #j in #DirectoryList[]
{
     SrvRqst (
         #PRList,        // The Previous Responders list
         ”service:wbem”,    // Service type
         “DEFAULT”,          // The scope
         RegisteredProfilesSupported=“SNIA:*”, // The predicate
         NULL)             // SLP SPI (security token)

     #ServiceList [#j] = SrvRply  (
         Count,       // count of URLs
         URL for each SA returned)
}
Go to Step 9.

//Step 8: Unicast a Service Request to each of the SAs specifying
//   a query predicate to select CIM Servers that support SNIA profiles
//   and listen for responses.
for #j in #SAList[]
{
     SrvRqst (
         #PRList,        // The Previous Responders list
         ”service:wbem”,    // Service type
         “DEFAULT”,          // The scope
         RegisteredProfilesSupported=“SNIA:*”, // The predicate
         NULL)             // SLP SPI (security token)

     #ServiceList [#j] = SrvRply  (
         Count,       // count of URLs
         URL for each SA returned)
}


// Step 9: Next retrieve the attributes of each advertisement
For #i in #ServiceList[]  // for each url in list
{
     AttrRqst (
         #PRList,       // The Previous Responders list
```

```
        #ServiceList[#i],// a url from #ServiceList[]
        "DEFAULT", // The scope
        NULL,  // Tag list.  NULL means return all attributes
        NULL)  // SLP SPI (security token)
    #DirectoryEntries [#i] = AttrRply (attr-list)
}

// Step 10: Correlate responses to the Service Request on unique
//   "service-id" to determine unique CIM Servers. The client will get
//   multiple responses (one for each access point) for each CIM
//   Server. At this point, the client has a list of CIM Servers that
//   claim to support SNIA profiles.
```

8.2.4.1.5.2 Enumerate Profiles Supported by a Given CIM Server

```
// DESCRIPTION
// A management application wishes to determine the Profiles supported by
// a particular CIM Server.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume the client only wants to know the "top level" profiles
//   supported by the CIM Server
// 2.Assume the client has used SLP to find the CIM Servers and has a
//   #DirectoryEntries [] structure
// 3.This recipe describes the operations for one of the entries in
//   the #DirectoryEntries [] structure.
// 4.    Assume the index into #DirectoryEntries[] for the CIM Server of
//        interest is #i.

// Step 1: Get the server url for the CIM Server.
#ServerName = #DirectoryEntries[#i].service-id

// Step 2: Get the Interop Namespace for the CIM Server.
#Inamespace = #DirectoryEntries[#i].InteropSchemaNamespace[1]

// Step 3:  Establish a connection to the CIM Server with
// #INameSpace. Note that the WBEM operations throughout the remainder
// of this recipe are performed with this client handle.
<Make client connection to this server using the interop namespace>

// Step 4:  Get the names of all the RegisteredProfiles in the
// Interop Namespace
#ProfileName[] = EnumerateInstances("CIM_RegisteredProfile",
TRUE, TRUE, FALSE, FALSE,
["RegisteredName"])

// Step 5:  Get all the RegisteredSubprofiles in the Interop Namespace
#SubprofileName[] = EnumerateInstances("CIM_RegisteredSubprofile",
TRUE, TRUE, FALSE, FALSE,
["RegisteredName"])


// Step 6:  Subtract the list RegisteredSubprofiles from the list of
// RegisteredProfiles
     #k = 0
     for #i in #ProfileName[i] {
        for #i in #SubprofileName[j] {
           if #ProfileName[#i] != #SubProfileName[#j] {
              #TempArray[#k+1]=#ProfileName[#i]
           }
        }
     }
     #ProfileName[] = #TempArray[]
```

8.2.4.1.5.3 Identify the ManagedElement Defined by a Profile

```
// DESCRIPTION
// A management application wishes to determine the ManagedElement that
// is defined by a particular Profile.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume the client has located the profile and has its object path
//   ($RegisteredProfile->)

// Step 1: Determine the ManagedElement (System) by traversing the
//    ElementConformsToProfile association from the RegisteredProfile
//    that is the top level Profile that applies to the System
$ManagedElement->[] = AssociatorNames (
     $RegisteredProfile->,
     "CIM_ElementConformsToProfile",
     "CIM_System",    // Note: substitute "CIM_AdminDomain" for Fabrics
                   // or "CIM_ComputerSystem" for Arrays, tape libraries, switchs
and the like
                               // or "CIM_ObjectManager" for Servers
     NULL,
     NULL)


// Step 2: The object name of more than one System may be contained
// in the array returned. Examine the contents of $ManagedElement[]
// and save the name of the System of interest as $Name.

// NOTE: "Top" level object for each profile will be returned. It MUST have
// an ElementConformsToProfile association.  To accommodate other
// potential ManagedElements, then it will be necessary need to throw out
// the ones that are NOT top level objects.
// NOTE: The object path for the ManagedElement MAY be in a Namespace
//   that is different than the Interop Namespace. As a result, if the
//   client wishes to actually access the ManagedElement, the client
//   may get the namespace for the element by cracking the REF to the
//   element:
#NameSpace=$Name.getNameSpace()
```

8.2.4.1.5.4 Determine the SNIA Version of a Profile

```
// DESCRIPTION
// A management application wishes to determine the SNIA version
// that a particular Profile supports.

//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume the client only wants to know version information
//   for a SNIA profile
// 2.Assume the client has already found the profile and has the
//   $RegisteredProfile-> reference


     // Step 1: Get the Instance of the Profile name.
     $Profile = GetInstance($RegisteredProfile->)

     // Step 2: Determine the SNIA Version for the Profile selected.
     #SNIAVersion = $Profile.RegisteredVersion
```

8.2.4.1.5.5 Determine the Subprofile Capabilities of a Profile

```
// DESCRIPTION
// A management application wishes to determine the optional subprofiles
// supported by a SNIA Profile.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume the client has already discovered the CIM Server that
//   supports the SNIA profile
// 2.Assume the client already has a $ObjectManager-> reference for
//   the CIMOM on the WBEM Server.
// 3.Assume the client already has a $RegisteredProfile-> reference
//   for the profile in question.

// Step 1: Check the version of the supported profile. Based on the
//   RegisteredVersion property, the client should know what functions
//   are REQUIRED as part of the profile definition.
$Profile = GetInstance($RegisteredProfile->)
#ProfileVersion = $Profile.RegisteredVersion
```

```
// Step 2: For each Profile, traverse the SubProfileRequiresProfile
//   association to determine what optional subprofiles are also
//   supported. If the subprofile (e.g., CopyServices subprofile)
//   exists for a profile, this means that the copy services are
//   supported. The Copy Services also has a Version
//   (RegisteredSubProfile.RegisteredVersion). The RegisteredVersion
//   of the subprofile MUST match the RegisteredVersion of the profile.
//   The RegisteredVersion implies a set of functional capabilities
//   that are defined for that version of the subprofile.
$Subprofiles[] = Associators (
     $RegisteredProfile->,
“CIM_SubProfileRequiresProfile”,
     “CIM_RegisteredProfile”,
     NULL, NULL, false, false, NULL)

// Step 3: Verify that each Subprofile has the same version as the
//   Profile
for #i in $Subprofiles[]
{

     #SubprofileVersion = $Subprofile[#i].RegisteredVersion
     if (!compare(#SubprofileVersion, #ProfileVersion))
     {
        Error(“Subprofile version mismatch with Profile version”)
     }

}
```

8.2.4.1.5.6 Find all Profiles and Subprofiles on a Server

```
// DESCRIPTION
// A management application wishes to list all the SNIA profiles and
// their related subprofiles for a specific CIM Server.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume the client has already discovered the CIM Servers that
//   support SNIA profiles

// Step 1:  Get the names of all the RegisteredProfiles and their names
// in the Interop Namespace
$ProfileName[] = EnumerateInstances(“CIM_RegisteredProfile”
                            true, true, false, false, {“RegisteredName”})

// Step 2:  Get all the RegisteredSubprofiles in the Interop Namespace
$SubprofileName[] = EnumerateInstances(“CIM_RegisteredSubprofile”,
                            true, true, false, false, {“RegisteredName”})
```

```
// Step 3:  Subtract the list RegisteredSubprofiles from the list of
// RegisteredProfiles
#k = 0
for #i in #ProfileName[#i] {
    for #j in $SubprofileName[#j] {
        if ($ProfileName[#i] != $SubProfileName[#j]) {
               #TempArray[#k+1]=#ProfileName[#i]
         }
     }
}
#ProfileName[] = #TempArray[]

// Step 4:  Get the ObjectName for the Profiles
for #i in #ProfileName[] {
$Profile->[#i]=$Name.getObjectPath(#ProfileName[#i])
}

// Step 5: Get the subprofiles associated to the profiles.
for #i in $ProfileName[]
{
    $Subprofile[] = Associators(
                           $ProfileName[#j].getObjectPath(),
                           "CIM_SubprofileRequiresProfile",
                           "CIM_RegisteredSubprofile",
                           NULL, NULL, false, false, NULL)
}
```

8.2.4.1.5.7 Segregate a SAN Device Type

```
// DESCRIPTION
// A management application wishes to manage a particular type of SAN
// device, but not other devices. So the management application needs to
// isolate the particular CIM Servers that support the type of device it
// wants to manage.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.Assume CIM Servers have advertised their services (SrvReg)
// 2.Assume there are one or more Directory Agents in the subnet
// 3.Assume no security on SLP discovery
// 4.#DirectoryList[] is an array of directory URLs
// 5.#DirectoryEntries [] is an array of directory entry Structures.
//   The structure matches the "wbem" SLP Template (see "Standard
//   WBEM Service Type Templates").
// 6.Assume that the device is #DesiredProfile and the device is an
//   SMI-S device (a SNIA defined profile)
// Step 1:  Set the Previous Responders List to the Null String.
#PRList = ““

// Step 2: Multicast a Service Request for a Directory Server Service.
//   This is to find Directory Agents in the subnet.
//
SrvRqst (
     #PRList,        // The Previous Responders list
     ”service:directory-agent” // Service type
     “DEFAULT”,          // The scope
     NULL,             // The predicate
     NULL)             // SLP SPI (security token)

// Step 3: Listen for Response from Directory Agent(s)
#DirectoryList[] = DAAdvert (
     BootTimestamp, // Time of last reboot of DA
     URL,      // The URL of the DA
     ScopeList,// The scopes supported by the DA
     AttrList,// The DA Attributes
     SLP SPI List,// SLP SPI (SPIs the DA can verify)
     Authentication Block)
// Iterate on Steps 2 & 3, until a response has been received or the client
// has reached a UA configured CONFIG_RETRY_MAX seconds.

// Step 4: Unicast a Service Request to each of the DAs specifying a
//   query predicate to select CIM Servers that support SNIA
//   #DesiredDevice profiles and listen for responses.
for #j in #DirectoryList[]
{

     SrvRqst (
        #DAPRList,        // The Previous Responders list
        “service:wbem”,    // Service type
        “DEFAULT”,          // The scope
        “RegisteredProfilesSupported=SNIA:”+#DesiredProfile+”*”,
                                           // The predicate
        NULL)             // SLP SPI (security token)
     #ServiceList [#j] = SrvRply  (
        Count,       // count of URLs
        #SAPRList[])
}

// Step 5: Next retrieve the attributes of each advertisement
For #i in #ServiceList[]  // for each url in list
{
     AttrRqst (
        #SAPRList,       // The Previous Responders list
```

```
                #ServiceList[#i ],// a url from #ServiceList[]
                “DEFAULT”, // The scope
                NULL,  // Tag list.  NULL means return all
                       // attributes
                NULL)  // SLP SPI (security token)
             #DirectoryEntries [#i] = AttrRply (#attr-list)
        }

        // Step 7: Correlate the responses to the Service Request on unique
        //   “service-id” to determine unique CIM Servers. The client will get
        //   multiple responses (one for each access point) for each CIM
        //   Server. At this point, the client has a list of CIM Servers that
        //   claim to support SNIA #DesiredProfile profiles.
```

8.2.4.1.6 Registered Name and Version

Server version 1.1.0

8.2.4.1.7 CIM Server Requirements

Table 361: CIM Server Requirements for Server

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.4.1.8 CIM Elements

Table 362: CIM Elements for Server

Element Name	Description
Mandatory Classes	
CIM_CIMXMLCommunicationMechanism (8.2.4.1.8.1)	
CIM_CommMechanismForManager (8.2.4.1.8.2)	This associates the ObjectManager and the communication classes it supports
CIM_ElementConformsToProfile (8.2.4.1.8.3)	Ties managed elements (e.g., Device system) to the registered profile that applies
CIM_ElementSoftwareIdentity (8.2.4.1.8.4)	Associates the profile and SoftwareIdentity instances
CIM_HostedAccessPoint (8.2.4.1.8.5)	This associates the communication mechanisms with the hosting System
CIM_HostedService (8.2.4.1.8.6)	Connects the ObjectManager to the System that is hosting the ObjectManager
CIM_Namespace (8.2.4.1.8.7)	There would be one for every namespace supported.
CIM_NamespaceInManager (8.2.4.1.8.8)	This osculates the namespace to the ObjectManager
CIM_ObjectManager (8.2.4.1.8.9)	This is the Object Manager service of the CIM Server
CIM_ReferencedProfile (8.2.4.1.8.12)	Ties profiles to other profiles
CIM_RegisteredProfile (8.2.4.1.8.13)	A registered profile that is supported by the CIM Server
CIM_RegisteredSubProfile (8.2.4.1.8.14)	For each subprofile of a profile that is supported
CIM_SoftwareIdentity (8.2.4.1.8.15)	A representation of some bundle of providers and supporting software that shares a version number.
CIM_SubProfileRequiresProfile (8.2.4.1.8.16)	Ties profiles to their subprofiles
CIM_System (8.2.4.1.8.17)	The System that is hosting the Object Manager (CIM Server)
Optional Classes	
CIM_Product (8.2.4.1.8.10)	optional
CIM_ProductSoftwareComponent (8.2.4.1.8.11)	optional
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_RegisteredProfile	Creation of a registered profile instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_RegisteredProfile	Deletion of a registered profile instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ObjectManager AND SourceInstance.Started < > PreviousInstance.Started	Deprecated WQL - Start of object manager
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ObjectManager AND SourceInstance.CIM_ObjectManager::Started < > PreviousInstance.CIM_ObjectManager::Started	CQL - Start of object manager

8.2.4.1.8.1 CIM_CIMXMLCommunicationMechanism

Class Mandatory: true

Table 363: SMI Referenced Properties/Methods for CIM_CIMXMLCommunicationMechanism

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ElementName		string	
CommunicationMechanism		uint16	
Version		string	
CIMValidated		boolean	
FunctionalProfilesSupported		uint16[]	
MultipleOperationsSupported		boolean	
AuthenticationMechanismsSupported		uint16[]	
OperationalStatus		uint16[]	
Optional Properties/Methods			
OtherCommunicationMechanismDescription		string	This shall not be NULL if Other is identified in CommunicationMechanism'
StatusDescriptions		string[]	
FunctionalProfileDescriptions		string[]	

8.2.4.1.8.2 CIM_CommMechanismForManager

This associates the ObjectManager and the communication classes it supports
Class Mandatory: true

Table 364: SMI Referenced Properties/Methods for CIM_CommMechanismForManager

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ObjectManager	
Dependent		CIM_ObjectManagerCommunicationMechanism	

8.2.4.1.8.3 CIM_ElementConformsToProfile

Ties managed elements (e.g., Device system) to the registered profile that applies
Class Mandatory: true

Table 365: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	The RegisteredProfile to which the ManagedElement conforms.

Table 365: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
ManagedElement		CIM_ManagedElement	The ManagedElement that conforms to the RegisteredProfile.

8.2.4.1.8.4 CIM_ElementSoftwareIdentity

Associates the profile and SoftwareIdentity instances
Class Mandatory: true

Table 366: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SoftwareIdentity	The Software
Dependent		CIM_ManagedElement	The Profile or Subprofile

8.2.4.1.8.5 CIM_HostedAccessPoint

This associates the communication mechanisms with the hosting System
Class Mandatory: true

Table 367: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System
Dependent		CIM_ServiceAccessPoint	The ServiceAccessPoints hosted by this system

8.2.4.1.8.6 CIM_HostedService

Connects the ObjectManager to the System that is hosting the ObjectManager
Class Mandatory: true

Table 368: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System.
Dependent		CIM_Service	The Service hosted on the System.

8.2.4.1.8.7 CIM_Namespace

There would be one for every namespace supported.
Class Mandatory: true

Table 369: SMI Referenced Properties/Methods for CIM_Namespace

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	

Table 369: SMI Referenced Properties/Methods for CIM_Namespace

Property	Flags	Type	Description & Notes
ObjectManagerCreationClass-Name		string	
ObjectManagerName		string	
CreationClassName		string	
Name		string	
ClassType		uint16	
Optional Properties/Methods			
DescriptionOfClassType		string	Mandatory if ClassType is set to Other"
ClassInfo		uint16	Deprecated in the MOF, but required for 1.0 compatibility. Not required if all hosted profiles are new in 1.1
DescriptionOfClassInfo		string	Deprecated in the MOF, but mandatory for 1.0 compatibility. Mandatory if ClassInfo is set to Other"

8.2.4.1.8.8 CIM_NamespaceInManager

This osculates the namespace to the ObjectManager
Class Mandatory: true

Table 370: SMI Referenced Properties/Methods for CIM_NamespaceInManager

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ObjectManager	The ObjectManager containing a Namespace
Dependent		CIM_Namespace	The Namespace in an ObjectManager

8.2.4.1.8.9 CIM_ObjectManager

This is the Object Manager service of the CIM Server
Class Mandatory: true

Table 371: SMI Referenced Properties/Methods for CIM_ObjectManager

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
ElementName		string	
Description		string	
OperationalStatus		uint16[]	
Started		boolean	
Optional Properties/Methods			
StopService()			

8.2.4.1.8.10 CIM_Product

optional
Class Mandatory: false
No specified properties or methods.

8.2.4.1.8.11 CIM_ProductSoftwareComponent

optional
Class Mandatory: false
No specified properties or methods.

8.2.4.1.8.12 CIM_ReferencedProfile

Ties profiles to other profiles
Class Mandatory: true

Table 372: SMI Referenced Properties/Methods for CIM_ReferencedProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_RegisteredProfile	The RegisteredProfile that is referenced by the Dependent Profile.
Dependent		CIM_RegisteredProfile	A RegisteredProfile that references other profiles.

8.2.4.1.8.13 CIM_RegisteredProfile

A registered profile that is supported by the CIM Server
Class Mandatory: true

Table 373: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	This is a unique value for the profile instance
RegisteredOrganization		uint16	This is the official name of the organization that created the Profile. For SMI-S profiles, this would be SNIA.
RegisteredName		string	This is the name assigned by the organization that created the profile
RegisteredVersion		string	This is the version number of the organization that defined the Profile.
AdvertiseTypes		uint16[]	Defines the advertisement of this profile. If the property is null then no advertisement is defined. A value of 1 is used to indicate other and a 3 is used to indicate 'SLP"
Optional Properties/Methods			
OtherRegisteredOrganization		string	
AdvertiseTypeDescriptions		string[]	This shall not be NULL if Other is identified in AdvertiseType'

8.2.4.1.8.14 CIM_RegisteredSubProfile

For each subprofile of a profile that is supported
Class Mandatory: true

Table 374: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	This is a unique value for the subprofile instance
RegisteredOrganization		uint16	This is the official name of the organization that created the subprofile. For SMI-S profiles, this would be SNIA.
RegisteredName		string	This is the name assigned by the organization that created the profile (or subprofile)
RegisteredVersion		string	This is the version number of the organization that defined the subprofile. It shall be the same as its parent profile
AdvertiseTypes		uint16[]	Should be NotAdvertised for subprofiles
Optional Properties/Methods			
OtherRegisteredOrganization		string	
AdvertiseTypeDescriptions		string[]	This field should be null

8.2.4.1.8.15 CIM_SoftwareIdentity

A representation of some bundle of providers and supporting software that shares a version number.
Class Mandatory: true

Table 375: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	A user-friendly name for the instrumentation software
InstanceID		string	
VersionString		string	
Manufacturer		string	The name of the company associated with the instrumentation software
Classifications		uint16[]	

8.2.4.1.8.16 CIM_SubProfileRequiresProfile

Ties profiles to their subprofiles

Class Mandatory: true

Table 376: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_RegisteredProfile	The RegisteredProfile that is referenced/required by the subprofile.
Dependent		CIM_RegisteredSubProfile	A RegisteredSubProfile that requires a scoping profile, for context.

8.2.4.1.8.17 CIM_System

The System that is hosting the Object Manager (CIM Server)
Class Mandatory: true

Table 377: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	
Description		string	
ElementName		string	
OperationalStatus		uint16[]	
NameFormat		string	

8.2.4.1.9 Related Standards

Table 378: Related Standards for Server

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
Representation of CIM using XML	2.2.0	DMTF
WBEM Discovery using SLP	1.0.0	DMTF
WBEM URI Specification	1.0.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.4.2 Indications Subprofile

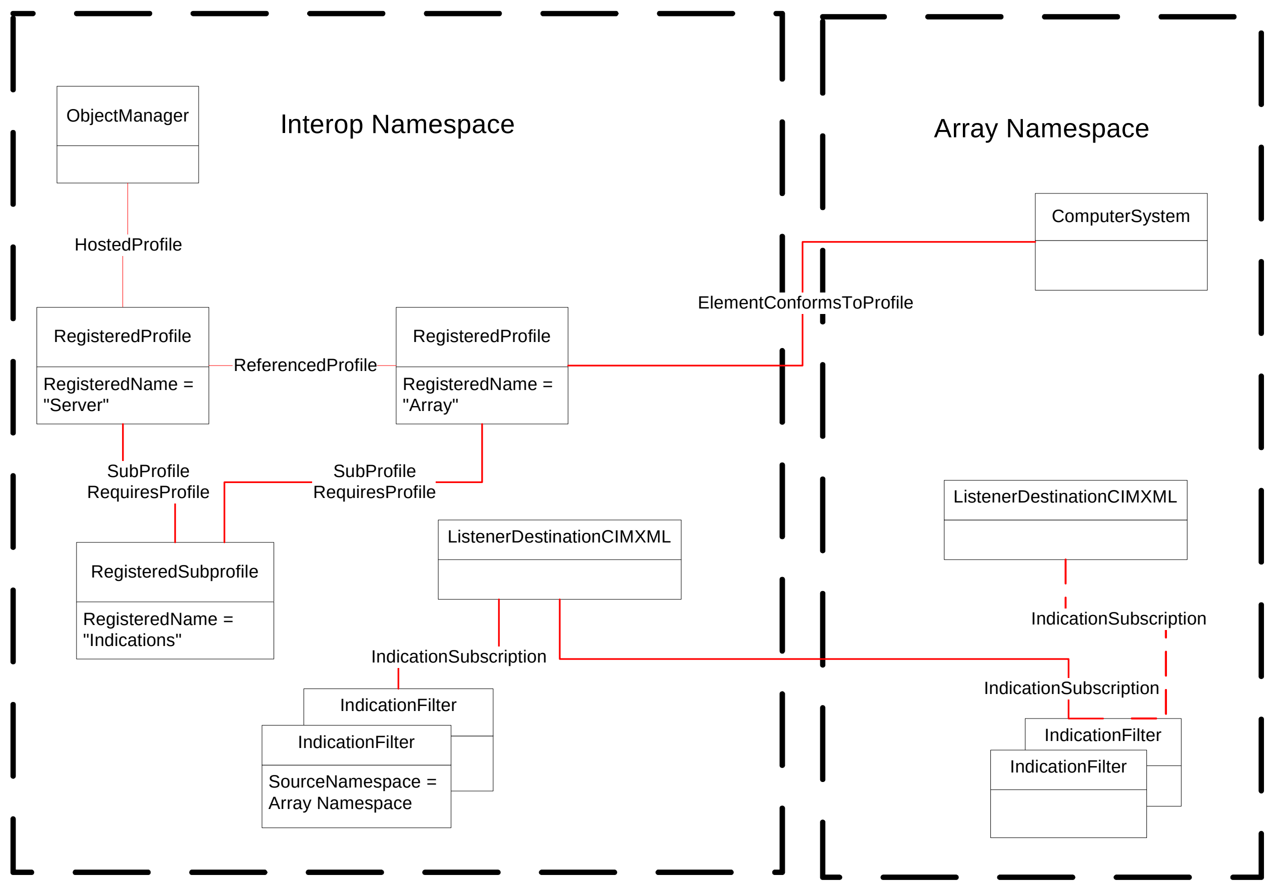
8.2.4.2.1 Description

Indications are support for unsolicited event notification. Each profile that supports event notification through CIM indications would support this subprofile and its classes and associations.

The Indications Subprofile is a subprofile of the Server Profile. It may also be a mandatory subprofile of any other profile (e.g., Array Profile).

Note: Refer to individual profile definitions to see whether or not the Indications Subprofile is mandatory or not. Figure 67: "Indications Subprofile and Namespaces" illustrates the structure of profiles, the Indications Subprofile and indication instances implied by an Array's support for the Indications Subprofile.

Figure 67: Indications Subprofile and Namespaces



Indication filters are defined in the context of the namespace in which they are implemented. In Figure 67: "Indications Subprofile and Namespaces", this is the Array's namespace. The indication filters shall be defined in two places: The InteropNamespace and the Namespace where the indications are intended to originate. For the Filters defined in the InteropNamespace, the SourceNamespace property shall be filled out to indicate the Namespace where the indications are to originate. For the IndicationFilters defined in the Array Namespace, this property may be null (indicating the indications originate in the array namespace).

The RegisteredProfile for the Array is associated to the ComputerSystem that is the top level system for the Array. This is done via the ElementConformsToProfile association, which is a cross namespace association (populated by the provider). The IndicationFilters may also be populated by the Provider (or they may be created by a client). In either case, they are created in both the Interop Namespace and the namespace of the array. The ListenerDestinationCIMXML class shall be in the Interop Namespace and may also be in the “source” namespace. And there would be two instantiations of the IndicationSubscription association: one in the Interop Namespace and one in the Array Namespace.

SMI-S profile and subprofile implementations that support indications shall support either the use of “predefined” indications filters, “client defined” indication filters or both. In the case of an implementation that supports “predefined” filters, the SMI-S Server would populate its model with indication filters that it supports. SMI-S Clients would select the indication filters to which they wish to subscribe from the list supplied by the SMI-S Server (enumeration of IndicationFilters in the appropriate namespace). In the case of an implementation that supports “client defined” filters, the SMI-S Server shall support filter creation (and deletion) by clients and it shall support creation of at least the filters defined by the profile.

Creation of an IndicationFilter will cause the creation of instances in both the InteropNamespace and the “Source” namespace. ListenerDestinationCIMXML instances should be created in the InteropNamespace, but may also be created in the “Source” Namespace (for IS24775-2006, *Storage Managemen*t, compatibility reasons). If a ListenerDestinationCIMXML instance is created in the “Source” Namespace, a duplicate instance will be instantiated in the InteropNamespace. However, if a ListenerDestinationCIMXML is created in the InteropNamespace, it may not be created in the “source” namespace.

Note: An implementation may support both “predefined” filters and “Client Defined” filters.

SMI-S Clients would subscribe to the indications for the events to which they wish to be notified. They would also supply an address (Indication listener) in which the indications are to be sent. SMI-S Clients shall use the subclass ListenerDestinationCIMXML when creating subscriptions.

In any given implementation Indication Filters are scoped by NameSpace. That is, a subscription to the change of operational status for a ComputerSystem can result in reporting of any change of operational status for ANY ComputerSystem managed within a Namespace. A client should inspect any indication to see if it is for an element that it manages.

All indication filters identified by a profile are mandatory for that profile. There is no notion of optional or recommended indication filters. A vendor implementation may support additional indication filters, but all the filters identified in SMI-S are mandatory (in the context of the implementing profile).

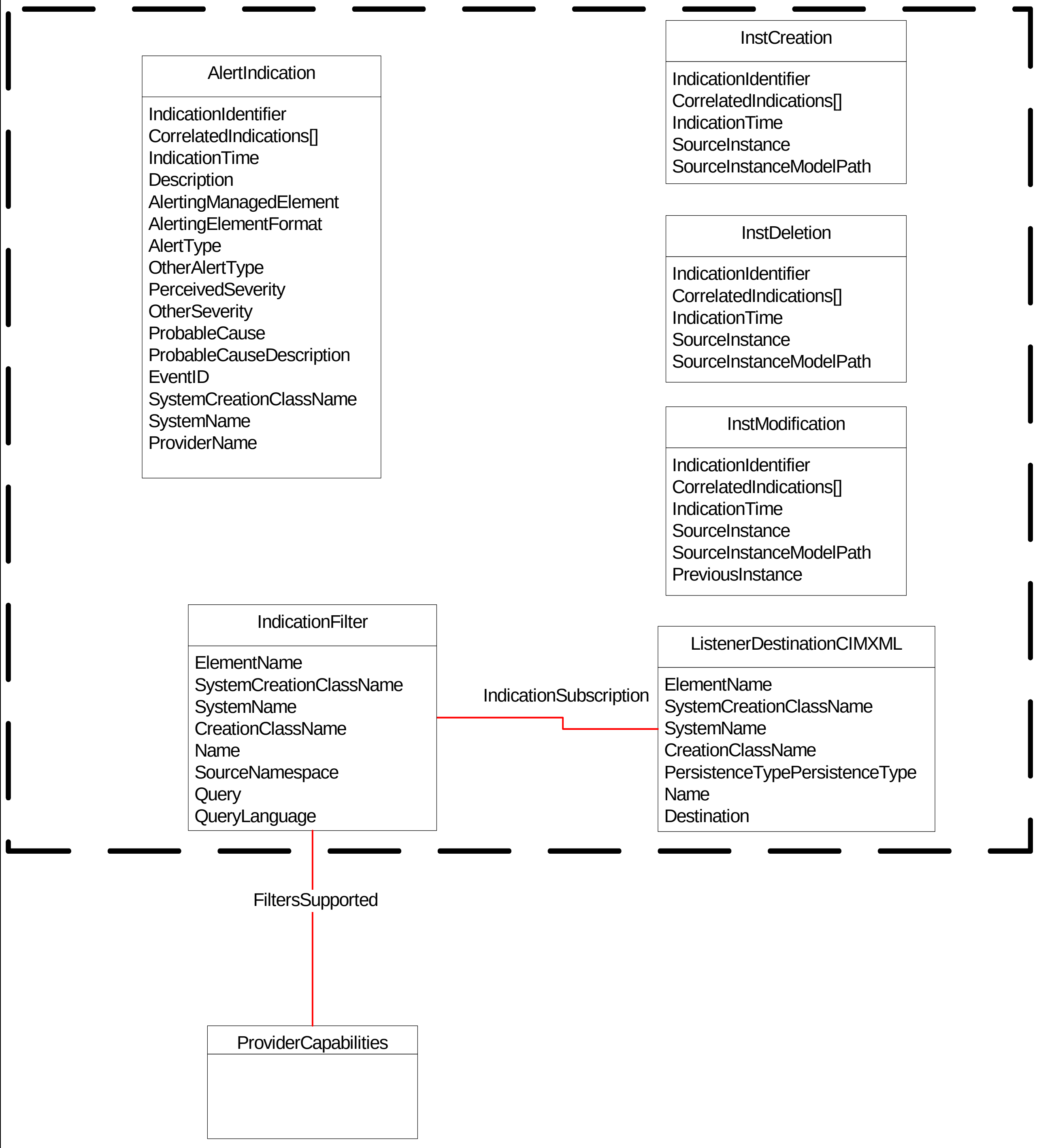
Note: Indication filters may correspond to optional features in a profile. When a provider supports an optional feature, all of the indications corresponding to the feature are mandatory. This means that the provider shall supply the filters or shall allow a client to define the filters. Indications corresponding to the filter shall be generated by the provider when a corresponding event occurs. On the other hand, if a profile implementation does not support a subprofile that defines mandatory indications, then the profile implementation does not need to support those indications.

8.2.4.2.1.1 Basic Indication Classes and Association

Figure 68: "Indications Subprofile Instance Diagram" illustrates the classes used in support of indications. Any given profile implementation may not include all of these classes. But they would at least support IndicationFilters (possibly predefined), ListenerDestinationsCIMXML and

IndicationSubscriptions. The actual types of indications supported can vary by profile (see the “CIM Element” section of the profile to determine the types of indications supported).

Figure 68: Indications Subprofile Instance Diagram



Clients request indications to be sent to them by subscribing to the indication filters. Subscriptions are stored in the SMI-S Server. A Subscription is expressed by the creation of a IndicationSubscription association instance that references an IndicationFilter (a filter) instance, and an ListenerDestination (for the handler of the indications) instance. A Filter contains the query that selects an indication class or classes.

SMI-S Servers that support SMI-S profiles that provide CIM indications support shall populate their models with the filters as defined by the profile(s) or allow clients to create the filters that are defined for the profile(s). Additional filters may also be created by indication consumers (e.g., SMI-S Clients), but this is not mandatory with SMI-S. The client would create these filters using CreateInstance intrinsic method.

The query property of the IndicationFilter is a string that specifies which indications are to be delivered to the client. There is also a query language property that defines the language of the query string. Example query strings are:

“SELECT * FROM AlertIndication”

“SELECT * FROM InstModification WHERE SourceInstance ISA ComputerSystem”

AlertIndication and InstModification are types of indications. The first query says to deliver all alert type indications to the client, and the second query says to deliver all instance modification indications to the client, where the instance being modified is a ComputerSystem (or any subclass thereof).

DEPRECATED

Note: For this version of SMI-S and future revisions, the preferred query language will be “DMTF:CQL”. Support for the Query Language specified in IS24775-2006, *Storage Managemen*t (SMI-S 1.0), is being deprecated.

DEPRECATED

A ListenerDestination specifies the means of delivering indications to the client. The subclass ListenerDestinationCIMXML provides for XML encoded indications to be sent to a specific URL, which is specified as a property of that class.

When a client receives an indication, it will receive some information with the indication, and then it may need to do additional queries to determine all of the consequences of the event.

Note: To avoid multiple calls to get additional data for an indication, profile designers (or clients, for client defined filters) should consider more elaborate Queries for Filters to return more information.

The instances of AlertIndications, InstCreation, InstDeletion and InstModification are temporary. They exist until they are delivered to the subscribing clients. The ListenerDestinationCIMXML, IndicationFilter and IndicationSubscription instance are permanent. That is, they persist until action is taken by client to delete them.

One final note on the indications supported. InstModification may or may not require the PreviousInstance property. A profile may be designed to require it or not. If the SMI-S profile defines an IndicationFilter on InstModification it shall specify whether or not PreviousInstance is required. It may always be recommended. If a profile defines PreviousInstance as optional, then an implementation may provide a previous instance (or not). However, if the SMI-S profile defines an IndicationFilter on InstModification with PreviousInstance required, then all implementations shall implement the PreviousInstance property.

8.2.4.2.1.2 AlertIndications

AlertIndications are used to by HFM and certain profiles for indicating process events. Unlike life cycle events, which report on changes to instances, AlertIndications report on process related events (such as error events) or events on aspects that are not part of the model. Since these indications are not necessarily identifiable by a CIM Instance, the properties shall convey the necessary information about the event.

The mandatory properties of an AlertIndication are:

- IndicationIdentifier - An identifier for the Indication that can be used for identification when correlating Indications (see the CorrelatedIndications array).
- IndicationTime - The time and date of creation of the Indication.
- AlertingManagedElement - The identifying information of the entity (i.e., the instance) for which this Indication is generated. The property contains the path of an instance, encoded as a string parameter - if the instance is modeled in the CIM Schema. If not a CIM instance, the property contains some identifying string that names the entity for which the Alert is generated.
- AlertingElementFormat - The format of the AlertingManagedElement property is interpretable based upon the value of this property. Values are defined as: “Unknown”, “Other”, “CIMObjectPath”
- AlertType - This is an integer property that is a value map. The values supported are: “Other”, “Communications Alert”, “Quality of Service Alert”, “Processing Error”, “Device Alert”, “Environmental Alert”, “Model Change”, “Security Alert”
- PerceivedSeverity - An enumerated value that describes the severity of the Alert Indication from the notifier's point of view. This is an integer property that is a value map. The values supported are: “Unknown”, “Other”, “Information”, “Degraded/Warning”, “Minor”, “Major”, “Critical”, “Fatal/Non Recoverable”.
- ProbableCause - This is an integer property that is a value map. There are many values that may be set (refer to the MOF for details).
- SystemCreationClassName - The scoping System's CreationClassName for the Provider generating this Indication.
- SystemName - The scoping System's Name for the Provider generating this Indication.
- ProviderName - The name of the Provider generating this Indication.

In addition, the following properties are recommended, but not mandatory:

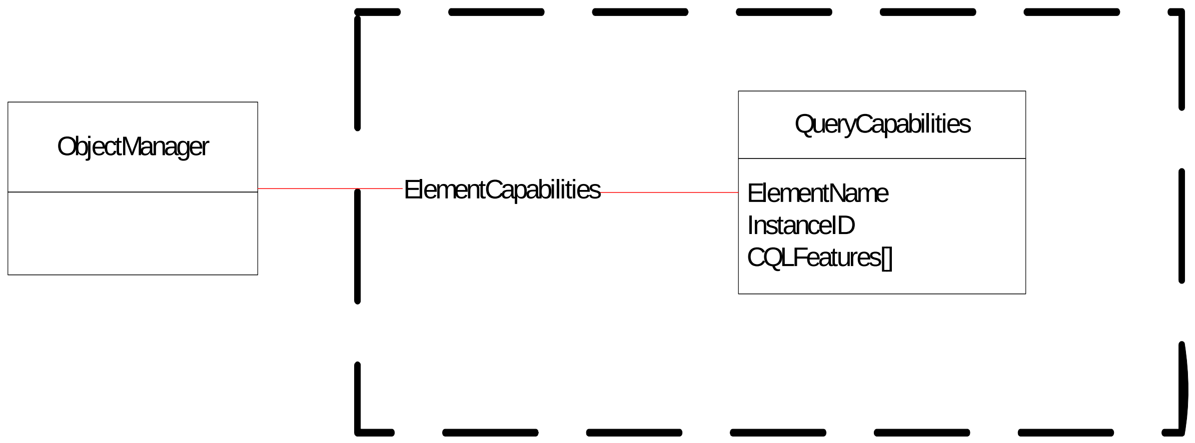
- CorrelatedIndications[]
- Description - A short description of the Indication.
- OtherAlertType - This property is mandatory if the AlertType is 1 (for “other”).
- OtherSeverity - This property is mandatory if the PerceivedSeverity is 1 (for “other”)
- ProbableCauseDescription - Provides additional information related to the ProbableCause.
- EventID - An instrumentation or provider specific value that describes the underlying \”real-world\” event represented by the Indication.

For descriptions of how these properties should be encoded, see the profile for specific alert indications that are supported.

8.2.4.2.1.3 Query Capabilities

If a profile supports “Client Defined” filters the CIM Server shall support CQL and the ObjectManager shall identify the CQL Features supported. This is done by associating an instance of QueryCapabilities to the ObjectManager using the ElementCapabilities association. This is illustrated in Figure 69: "QueryCapabilities for Client Defined Filters".

Figure 69: QueryCapabilities for Client Defined Filters



Note that QueryCapabilities are defined for the CIM Server (ObjectManager). The assumption is that any Query feature supported by CIM Server can be used in an IndicationFilter. The QueryFeatures property contains the list of features that are supported. The possible values are “Basic Query”, “Simple Join”, “Complex Join”, “Time”, “Basic Like”, “Full Like”, “Array Elements”, “Embedded Objects”, “Order By”, “Aggregations”, “Subduer”, “Satisfies Array”, “Distinct”, “First” and “Path Functions”. For a definition of what these values mean, see the *CIM Query Language Specification*.

8.2.4.2.1.4 Special handling for Multiple events of the same type

When a client creates a subscription (using CreateInstance), the provider may fill in the RepeatNotificationPolicy and related properties. This information describes the policy used by the implementation for reporting multiple events of the same type (multiple events for the subscription). If the RepeatNotificationPolicy is “None”, then the client will receive all indications. If the RepeatNotificationPolicy is “Suppress”, then all indications after the first ‘n’ (where ‘n’ is defined by the RepeatNotificationCount) are not sent (within the RepeatNotificationInterval time). If the RepeatNotificationPolicy is “Delay”, then indications are collected and notification is only sent after a certain number of events happen (as defined by RepeatNotificationCount) or the time interval (RepeatNotificationInterval) lapses.

8.2.4.2.1.5 Indication Delivery

In some cases, the Client (ListenerDestination) may not be available when an event occurs that requires delivery to the client. In such cases, the CIM Server should attempt delivery to the listener destination 3 times. If the delivery cannot be made within 3 attempts, the indication may be considered delivered.

If the ListenerDestinationCIMXML.PersistenceType is set to "3" (transient), the IndicationSubscription may be deleted after 3 attempts that fail. If the ListenerDestinationCIMXML.PersistenceType is set to "2" (permanent) the IndicationSubscription shall be retained.

8.2.4.2.1.6 Instrumentation Requirements

8.2.4.2.1.6.1 General Instrumentation Considerations

A SMI-S Server may allow a client to create indications filters. If the SMI-S Server does not support this option, then the server shall send a return code indicating a request to create an instance of a filter is unsupported. This allows the provider to inform clients which types of indications the provider supports.

For example, a provider that does not support SNMPTrapAlertIndications shall return unsupported for an indications filter create request.

8.2.4.2.1.6.2 SMI-S Dedicated Server Considerations

The dedicated server should supply more detailed queries as described in the profile sections.

A standard implementation of indications requires the server to accept client requests to create ListenerDestinations. The dedicated server implementation uses the Instance Manipulation functional group in addition to Basic Read.

8.2.4.2.1.6.3 Additional Indications

Most Indication Filters defined in the “CIM Elements” section of the specification are mandatory. However, a profile may also document additional Indication Filters as optional filters. A client can determine whether or not “additional” indication filters are supported by one of two techniques:

1) Enumerating Predefined Indication Filters – this will return all the indication filters that have been predefined by the provider for the Namespace.

2) CreateInstance of the desired “additional” Indication Filter – if the “additional” indication filter is supported, the CreateInstance will succeed.

DEPRECATED

8.2.4.2.1.6.4 Support for Query Language for IS24775-2006, *Storage Managemen*t (SMI-S 1.0) **(DEPRECATED)**

Support for 1.0 Query Language, as specified in IS24775-2006, *Storage Managemen*t, will continue until the next major version, at which time it will be withdrawn from the standard.

<u>Note:</u> IS24775-2006, *Storage Managemen*t identified QueryLanguage as a mandatory property, but did not specify what value to put in the property. However, the query language used in IS24775-2006, *Storage Managemen*t was based on a precursor to CQL called WQL. It supported a syntax that used OperationalStatus comparisons. Since OperationalStatus is an array, such comparisons are ambiguous. In CQL the syntax intended was OperationalStatus[*], meaning that any of the values of the array compare to the value, then the expression evaluates to true.

For IS24775-2006, *Storage Managemen*t, indication filters, OperationalStatus comparisons are treated as the CQL OperationalStatus[*] comparisons.

DEPRECATED

8.2.4.2.1.6.5 Timing of Delivery of Indications

There are no standards for how quickly an implementation shall deliver an indication. All reasonable attempts should be made by the implementation to deliver all indications at the CIM Server’s earliest convenience.

There are also no standard guidelines on how long or how many attempts should be made to deliver an indication. As a general guideline an implementation should make at least 3 attempts to deliver an indication before giving up trying to deliver the indication. Similarly, delivery of indications should allow at least 30 seconds to elapse before giving up trying to deliver the indication. The intent is to allow sufficient time to allow any network problems to clear.

8.2.4.2.1.6.6 Handling of Indication Storms

Occasionally an event may occur that causes many indication filters to evaluate to true (an trigger many indications). This situation is referred to as an “indication storm.” These can be very expensive and degrade the performance of the environment. To contain the impact of this an implementation can employ any one of three techniques:

- use the RepeatNotificationPolicy (and related properties) of the IndicationSubscription.

EXPERIMENTAL

- Use of Bellwether events (if they are defined by the profile)
- Use of batching

EXPERIMENTAL

Use of RepeatNotificationPolicy

The RepeatNotificationPolicy property defines the desired behavior for handling Indications that report the occurrence of the same underlying event (e.g., the disk is still generating I/O errors and has not yet been repaired). For SMI-S, this is extended to include multiple indications that are generated from a single IndicationFilter.

The related properties are RepeatNotificationCount, RepeatNotificationInterval, and RepeatNotificationGap. The defined semantics for these properties depend on the value of RepeatNotificationPolicy, but values for these properties shall be set if the property is defined for the selected policy.

If the value of RepeatNotificationPolicy is 2 (\”None\”), special processing of repeat Indications shall not be performed.

If the value is 3 (\”Suppress\”) the first RepeatNotificationCount Indications, describing the same event, shall be sent and all subsequent Indications for this event suppressed for the remainder of the time interval RepeatNotificationInterval. A new interval starts when the next Indication for this event is received.

If the value of RepeatNotificationPolicy is 4 (\”Delay\”) and an Indication is received, this Indication shall be suppressed if, including this Indication, RepeatNoticationCount or fewer Indications for this event have been received during the prior time interval defined by RepeatNotificationInterval. If this Indication is the RepeatNotificationCount + 1 Indication, this Indication shall be sent and all subsequent Indications for this event ignored until the RepeatNotificationGap has elapsed. A RepeatNotificationInterval may not overlap a RepeatNotificationGap time interval.

For SMI-S, a single indication filter that identifies a change in OperationalStatus on StorageVolumes would be subjected to the RepeatNotificationPolicy, even though the repeat notifications may be from multiple StorageVolumes.

The RepeatNotificationPolicy can vary by implementation (or even IndictationFilter). However, it shall be specified on any subscription. The valid values for an SMI-S implementation are:

- 2 (\”None\”),
- 3 (\”Suppress\”), or
- 4 (\”Delay\”)

An SMI-S profile may restrict this further for any given indication filter, but it cannot expand this to other policies without breaking interoperability. For example, a profile might restrict InstCreation filters for ComputerSystems to “None” and restrict InstModification filters on StorageVolume to “Suppress” or

“Delay.” But an SMI-S profile shall not define “unknown” as a valid SMI-S setting for the RepeatNotificationPolicy.

Note: RepeatNotificationPolicy set to 2 “none” is compatible with IS24775-2006, *Storage Managemen*t.

EXPERIMENTAL

Use of Bellwether Events

There are many state changes in the model for a device or application that results in changes in many CIM instances. For example, the addition of a device or application representation to a CIMOM should result in creation indications for every single member instance of that device or application. The activation of a ZoneSet from one of the member Switches in a fabric should result to indication listeners on another Switch's namespace creation indications for every instance of the new ZoneSet.

The worse case risk is that several of this type of situation may occur simultaneously and result in network storms and the sudden saturation of the LAN. Additionally, the use of computing resources of the device or application producing the indication or client receiving the indications may be unacceptably high.

Indications provide the most value when they are used by a client as a mechanism to pick a significant or small number of changes in CIMOMs of interest. In order to capture a wide variety of changes, any of which may be pertinent to the client application, the client is likely to create many indication subscriptions and keep them all active simultaneously. This approach is not problematic because the number of management related changes to any device or application in the network is usually very small.

As mentioned previously, there are several potential situations where an excessive number of indications can be produced, thereby potentially overloading the network, originating CIMOM, and receiving client's resources. There is no need to occur such a risk because it is likely that the client is not going to be interested in all things at all times. The interest of the client in instance changes usually follows the needs of the current users of that client application.

Bellwether indications are used by SMI-S designers and individual implementation to signal many instance changes with one event. A client can assume that some previously defined graph of associated CIM instances are affected when it receives a bellwether indication. It can then choose, if warranted, to fetch all or some of these instances. This design prevent the previously mentioned adverse side effects.

Some rules being considered are:

- When a device or application is added to a namespace and there are indication subscription that cover some or all of the graph of instances added by side effect of the addition, then only a create indication is produced for the top level object for the device or application, like ComputerSystem, provided that there is an indication subscription for changes in the top-level object. Similarly, if a device or application is deleted in the same situation, then only a delete indication will be produced.
- Bellwether indication are mandatory if they exist in SMI-S and will be easily identified as being bellwether events.
  - The classes associated to the bellwether indication will be part of the definition of the indication. The client can assume that instances of these classes will have been affected and can choose to harvest that data. The implementation is not required to produce instances of every class listed as per the requirements defined elsewhere in SMI-S.
- SMI-S Designer's are encouraged to define bellwether indications, which can be of any class of indication, for major state changes of a model. In the previous examples, the device creation

could be a life cycle indication where changes in ZoneSet change may be best communicated by an Alert Indication.

Bellwether Indications for ComputerSystem

It is important to not overload a SMI-S client when device or applications are added or removed from CIM Object Managers. The addition or removal of the representation of a device or application is attributed to the creation or deletion of a top-level computer system instance. This overloading would arise from a SMI-S Agent sending creation or deletion indications to every indication destination for all component or dependent instances to the top-level computer system. For this profile, when a top-level computer system instance is created in the model, the SMI-S agent shall not produce indications for indication subscriptions, on indications that do not reference the top-level computer system, that would otherwise receive InstCreation indications. Likewise, for this profile, when a top-level computer system is deleted from the model, the SMI-S agent shall not produce indications for indications subscriptions, on indications that do not reference the top-level computer system, that would otherwise receive InstDeletion indications.

Not defined in this standard.

EXPERIMENTAL

8.2.4.2.1.6.7 Clarification of indication generation

General Requirements

To minimize the use of stale object references by WBEM Clients, a WBEM Server shall generate instance deletion indications, where defined as mandatory profile elements, whenever a MSE instance is removed while the WBEM Server is operational. The indication shall be generated for all causes of removal, which include but are not limited to, explicit WBEM instance manipulation by some WBEM Client, internal implementation of the WBEM Server outside the scope of SMI-S, and a side effect of invoking some WBEM extrinsic method.

A WBEM Server should generate instance deletion indications, where defined as mandatory profile elements, whenever a MSE instance that was present before a failure of the device or application is no longer present when the device or application recovers from the failure. Note: SMI-S already requires WBEM Servers to persist WBEM Client subscription for indications.

A WBEM Server shall generate instance creation indications, where defined as mandatory profile elements, whenever a MSE instance is created while the WBEM Server is operational. A WBEM Server shall also generate instance creation indications, where defined as mandatory profile elements, whenever a MSE instance that was not present before a failure of the device or application is present when the device or application recovers from the failure.

Almost universally in SMI-S profiles, all MSE's can be linked by association back to a specific “top-level” MSE. In most profiles this is either a ComputerSystem or a AdminDomain. A WBEM Server that is providing information on multiple devices will have multiple MSE instances, one for each of the devices. The behavior of WBEM Operations in the face of a failure of the device or applications differs.

Definition of “failed” MSE

A MSE instance is defined to be failed if any of the following conditions hold:

1) Failure status are contained in the OperationalStatus attribute, when present, and OperationalStatus array does not contain “OK”

2) EnumerateInstances, EnumerateInstanceNames, Associators, AssociatorNames, References, ReferenceNames WBEM Operations might return meaningless or no information for any mandatory profile element. OperationalStatus when present in the class will have meaningful data and

will have a failure status. Explicit values for “unknown” or “undetermined” are completely meaningful when defined for a profile element.

3) WBEM extrinsic operations that ERR_FAILED may indicate that this instance is failed.

4) CIM Instances that were returned before the failure of the MSE might not be returned after the failure. Indications representing the OperationalStatus change to a failure status were produced for the this 'top-level' CIM Instance or 'top-level' parent CIM Instance. The combination of these two situations define failure in this case

A MSE with an OperationalStatus of “Lost Communications” or “No Contact” obviously shall be considered failed because no WBEM operations can succeed.

An OperationalStatus of “Starting”, “Stopping”, or “Stopped” does not mandate failure. The detailed behavior of the MSE with regard to the conditions given above, determines whether these status's indicate failure. The WBEM Client should be warned of a possible failure scenario when receiving these status.

Minimal function for failed MSEs

Any failed instance represented by any WBEM Server shall support the following functionality. If the WBEM Server is not able to support the functionality on a failed instance, it shall delete the instance.

1) EnumerateInstances, EnumerateInstanceNames, Associators, AssociatorNames, References, and RefererenceNames WBEM Operations that include the failed instance as part of the return set will complete without error. The Key and the OperationalStatus attributes, when present, shall be properly provided.

2) When a GetInstance WBEM Operation is attempted on the failed instance, CIM_ERR_FAILED shall be returned with a message describing or indicating the failure of the device or application.

3) Failed instance names shall be returned from WBEM Operations that return Object Names. Failed instances shall be returned for WBEM Operations that return Instances but only the keys and OperationalStatus, when present, are mandatory.

4) Method invocations on failed MSEs will fail with the CIM_ERR_FAILED error.

Isolation of failed top-level MSE's

For efficiency and consistency of navigation, a WBEM Client should not be able to retrieve false or meaningless information from the WBEM Server about a MSE instance.

A WBEM Server can take one of two actions in the Failed MSE case and top-level MSE instances. It shall set the OperationalStatus on the top-level MSE instance to reflect the failed state and forward the related CIM Indications as required. It may also remove all directly or indirectly associated instances, generating the corresponding indications.

A WBEM Client shall be prepared to deal with a WBEM Object CIM_ERR_NOT_FOUND error, indicating the use of a stale object reference not avoided by timely receipt and processing of an instance deletion indication. A WBEM Client shall also consider the OperationalStatus of any MSE for which OperationalStatus is a mandatory profile element before treating the other attributes and associations of the instance as meaningful.

8.2.4.2.2 Health and Fault Management Considerations

Elements Reporting Health

The Indications Subprofile has no classes that report health information. However, indications are a means available for reporting changes in health status.

Health State Transformations and Dependencies

No Indications class have OperationalStatus or HealthState properties.

Standard Errors Produced

All manipulation of Indication classes and associations are done using intrinsic methods. The errors produced are those listed for intrinsic methods.

Cause and effect associations

Cause and effect associations are defined as part of the Health and Fault Management Package.

8.2.4.2.3 Cascading Considerations

Not Applicable.

8.2.4.2.4 Supported Subprofiles and Packages

None

8.2.4.2.5 Methods of the Profile

8.2.4.2.5.1 Extrinsic Methods of the Profile

No extrinsics are specified on the Indication Subprofile.

8.2.4.2.5.2 Intrinsic Methods of the Profile

The Indication Subprofile is mostly populated by providers and is accessible to clients using basic read and association traversal. However, there are two constructs that would be created by Clients. These are the ListenerDestinationCIMXML and the IndicationSubscription. In addition, a client may be able to create an IndicationFilter. In addition to being able to create them, client may delete them (except “pre-defined” filters which cannot be deleted), and a client may modify any IndicationFilter that was client created. These functions are performed using the intrinsics:

Table 379: Indications Subprofile Methods that Cause Instance Creation, Deletion or Modification

Method	CreatedInstances	Deleted Instances	Modified Instances
CreateInstance	ListenerDestinationCIMXML	N/A	N/A
CreateInstance	IndicationSubscription	N/A	N/A
CreateInstance	IndicationFilter	N/A	N/A
DeleteInstance	N/A	ListenerDestinationCIMXML	N/A
DeleteInstance	N/A	IndicationSubscription	N/A
DeleteInstance	N/A	IndicationFilter	N/A
ModifyInstance	N/A	N/A	IndicationFilter

CreateInstance - for ListenerDestinationCIMXML, IndicationSubscription and IndicationFilter

```
<instanceName>CreateInstance (
        [IN] <instance> NewInstance
 )
```

If successful, the return value defines the object path of the new CIM Instance relative to the target Namespace (i.e., the Model Path), created by the CIM Server.

Note that for CreateInstance of an IndicationSubscription requires that the ListenerDestinationCIMXML instance and the IndicationFilter exist.

If unsuccessful, one of the following status codes shall be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

CIM_ERR_ACCESS_DENIED, CIM_ERR_NOT_SUPPORTED, CIM_ERR_INVALID_NAMESPACE, CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters), CIM_ERR_INVALID_CLASS (the CIM Class of which this is to be a new Instance does not exist), CIM_ERR_ALREADY_EXISTS (the CIM Instance already exists), CIM_ERR_FAILED (some other unspecified error occurred).

Note that a ListenerDestinationCIMXML instance should be created in the Interop namespace. However, they may be created in the “Source” namespace. If the client creates a ListenerDestinationCIMXML instance in the “Source” namespace, then a duplicate ListenerDestinationCIMXML instance will be created in the Interop Namespace.

Note: The inverse is not true. If the client creates the ListenerDestinationCIMXML instance in the Interop Namespace, no instance will be created in another namespace (there is nothing that would indicate which Namespace would be the Source namespace).

IndicationFilters shall be created in either the Interop Namespace or the Namespace in which the indications are to originate. In either case, the Client only needs to create one instance (and providers will automatically create the corresponding instance in the other namespace).

Note: If a client attempts to create an IndicationFilter that already exists (has the same key fields), but other properties are different, then the request will fail. If the Client attempts to create an IndicationFilter that has identical properties to an existing IndicationFilter instance, it will succeed and CreateInstance need not treat the instance as a separate instance.

When a client creates an IndicationSubscription the client only needs to create a subscription to one of the IndicationFilters (the provider will automatically generate the corresponding subscription to the other filter instance). Even though there are two instance of the IndicationFilter created (and two instances of the subscription) duplicate indications will not be sent to the ListenerDestination.

Indeed, in general, redundant subscriptions need not produce duplicate indications (that is, if the same listener subscribes to two filters that are equivalent, then an implementation need not produce two indications).

DeleteInstance - for ListenerDestinationCIMXML, IndicationSubscription and IndicationFilter

```
void DeleteInstance (
        [IN] <instanceName> InstanceName
)
```

The InstanceName input parameter defines the name (model path) of the Instance to be deleted.

If successful, the specified Instance (ListenerDestinationCIMXML, IndicationSubscription or IndicationFilter) shall have been removed by the CIM Server.

The deletion of a ListenerDestinationCIMXML or an IndicationFilter instance will cause the automatic deletion of any associated IndicationSubscription instances. Deletion of an IndicationSubscription will not cause the deletion of any corresponding ListenerDestinationCIMXML or IndicationFilter instances. For example, the deletion of an instance may cause the automatic deletion of all associations that reference that instance. Or the deletion of an instance may cause the automatic deletion of instances (and their associations) that have a Min(1) relationship to that instance.

If unsuccessful, one of the following status codes shall be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

CIM_ERR_ACCESS_DENIED, CIM_ERR_NOT_SUPPORTED, CIM_ERR_INVALID_NAMESPACE, CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters), CIM_ERR_INVALID_CLASS (the CIM Class does not exist in the specified namespace), CIM_ERR_NOT_FOUND (the CIM Class does exist, but the requested CIM Instance does not exist in the specified namespace), CIM_ERR_FAILED (some other unspecified error occurred).

Note: Deleting the instance of an IndicationFilter in the Interop Namespace will cause the corresponding IndicationFilter in the "SourceNamespace" to also be deleted (and vice versa). Deletion of an indication filter will also cause all subscriptions to that filter to be deleted. However, deletion of a filter will not cause the deletion of any listener destination.

Note: Deleting the instance of an IndicationSubscription in the InteropNamespace will cause the corresponding IndicationSubscription in the "SourceNamespace" to also be deleted (and vice versa). However, deleting a subscription will not delete filters or listener destinations.

Note: Deleting the instance of ListenerDestinationCIMXML in either the InteropNamespace or the "source" namespace will cause the corresponding instance (if one exists) to be deleted.

ModifyInstance - for IndicationFilters

```
void ModifyInstance (
        [IN] <namedInstance> ModifiedInstance,
        [IN, Optional, NULL] string propertyList[] = NULL
 )
```

The ModifiedInstance input parameter identifies the name of the Instance to be modified, and defines the set of changes to be made to the current Instance definition.

The only Property that may be specified in the PropertyList input parameter is the Query property. Modification of all other properties is not specified by SMI-S.

If successful, the specified Instance shall have been updated by the CIM Server.

If unsuccessful, one of the following status codes shall be returned by this method, where the first applicable error in the list (starting with the first element of the list, and working down) is the error returned. Any additional method-specific interpretation of the error in is given in parentheses.

CIM_ERR_ACCESS_DENIED, CIM_ERR_NOT_SUPPORTED, CIM_ERR_INVALID_NAMESPACE, CIM_ERR_INVALID_PARAMETER (including missing, duplicate, unrecognized or otherwise incorrect parameters), CIM_ERR_INVALID_CLASS (the CIM Class of which this is to be a new Instance does not exist), CIM_ERR_NOT_FOUND (the CIM Instance does not exist), CIM_ERR_FAILED (some other unspecified error occurred)

8.2.4.2.6 Client Considerations and Recipes

8.2.4.2.6.1 Use of Profile Specific Recipes

See Recipes in related profile sections.

8.2.4.2.6.2 General Client Considerations

The indication filters that a client subscribes to are either "predefined" and populated by the profile, or they are created by the client. If the profile supports "predefined" indication filters the client can find them via an enumeration. If the client cannot find the filter it is looking for, it may attempt to create the desired indication filter. If this fails, the client should fall back to creating a filter exactly as it exists in SMI-S. This shall work. The "predefined" indication filters in this specification shall be populated in the profile or it shall be possible to create it.

8.2.4.2.6.3 Discovery of Implementation variations

A client will need to discovery the variations that are allowed in SMI-S profile implementations. A profile implementation has the following degrees of variability:

- Client defined IndicationFilters, pre-defined IndicationFilters or both
- InstModification, with or without PreviousInstance
- Additional Indications

To determine if an implementation supports Client Defined filters, the client should attempt to create an SMI-S specified filter. If it succeeds, the implementation supports client defined filters. At this point, the client can attempt to create a filter of its own choice or making (e.g., using the client's desired query). If it fails, this means the implementation does not support an indication based on the query used. The client may refer to the QueryCapabilities to ensure that it is using features that are supported by the CIM Server.

If the attempt to create an SMI-S specified indication filter fails, this means client defined queries are not supported. At this point, the client should look for pre-defined filters. This can be done by enumerating filters in the namespace of the profile the client wishes to monitor.

An implementation may (or may not) support PreviousInstance, when the SMI-S specification for the profile identifies InstModification as the indication filter and PreviousInstance is identified as optional. If a client wishes to determine whether or not the implementation actually supports PreviousInstance, it can only tell by receiving an InstModification indication.

Additional Indications are IndicationFilters that are supported by the implementation, but not mandatory with SMI-S. If the implementation supports pre-defined Filters, these can easily be discovered in the enumeration of IndicationFilters. If the implementation does not support pre-defined filters, then the only way a client can discover these is through trial and error (or specific knowledge of the implementation).

8.2.4.2.6.4 Client Defined Filters

Clients need to avoid Filters that generate excessive events. Subscriptions to a general-purpose Server should be specific to the provider – for example "select * from CompanyCorp_InstCreation" rather than "select * from CIM_InstCreation".

8.2.4.2.6.5 Indications Status

```
// DESCRIPTION
// Determine if the indication subscription requested already exists. If
// not, then attempt to create the indication subscription passed in. If
// the CIM Server does not support the addition of indication, then the
// CIM Client will need to poll for these instance changes. This recipes
// does not handle the issue of providing the target URL for indications.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.The namespace of interest has previously been identified and
//   defined in the #SomeNameSpace variable
// 2.The list of filters of interest has been previously built in the
//   #filters[] array. Each element is this array is the query filter itself

// FUNCTION: createIndication
```

```
sub createIndication ($Filter)
{
     try {
         <create indications as per SMIS specification>
     } catch(CIM Exception $Exception) {
         if($Exception.CIMStatusCode == CIM_ERROR_NOT_SUPPORTED) {
             // The implementation does not allow the creation of indication filters
             // Normally this should not happen because the filter being created
             // is the one required by SMI-S.
             <client polls for changes rather than listening to indications>‘
         } else {
             throw $Exception
         }
     }
}

// MAIN
$ExistingInstances[] = EnumerateInstances(#SomeNameSpace, “CIM_IndicationFilter”)
#found = false
for #i in #filters[]
{
     for #j in $ExistingInstances[]
     {
         if(compare($ExistingInstances[#j].Query, #filters[#i])
         {
             #found = true
         }
     }
     if(!#found) {
         &createIndiciation(#filters[#i])
     } else { // #found == true
         #found = false
     }
}
```

8.2.4.2.6.6 Listenable Instance Notification

```
// DESCRIPTION
// Create an indication subscription for every indication that is
// required by the profile.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1.The namespace of interest has previously been identified and
//   defined in the #SomeNameSpace variable

#filters[] = <array of SMIS filters for the target profile>
@{Determine if Indications already exist or have to be created} #filters
```

8.2.4.2.6.7 Life Cycle Event Subscription Description

```
// DESCRIPTION
// Create an indication subscription for the operational status for a
// computer systems defined within a given CIM agent and namespace. This
// subscription   will only be made in those CIM agents that have SAN
// devices or applications of   interest defined in them. The client will
// have to determine once having received the indication, whether the
// computer system related to this indication (AlertingManagedElement
// attribute) is of interest. This recipe does not handle the target URL
// for the indication.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// None

#filter[0] = “SELECT * FROM CIM_InstModification
    WHERE SourceInstance ISA CIM_ComputerSystem
      AND SourceInstance.OperationalStatus[0] <>
          PreviousInstance.OperationalStatus[0]”
@{Determine if Indications already exist or have to be created} #filter
```

8.2.4.2.6.8 Subscription for alert indications

```
// DESCRIPTION
// Create an indication subscription for every indication
// that isrequired by the profile
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.The namespace of interest has previously been identified and
//   defined in the #SomeNameSpace variable

#filters[] = <array of SMIS filters for the target profile>
@{Determine if Indications already exist or have to be created} #filters
```

8.2.4.2.6.9 Listenable Interface Modification Notification

```
// DESCRIPTION
// Create an indication subscription for every indication
// that isrequired by the profile
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.The namespace of interest has previously been identified and
//   defined in the #SomeNameSpace variable

#filters[] = <array of SMIS filters for the target profile>
@{Determine if Indications already exist or have to be created} #filters
```

8.2.4.2.6.10 Subscribe for Lifecycle Events where OperationalStatus Changes

```
// DESCRIPTION
// Create an indication subscription for the operational
// status for a computer systems defined within a given CIM agent and
// namspace.  This subscription will only be made in those CIM agents
// that have SAN devices or applications of interest defined in them. The
// client will have to determine once having received the indication,
// whether the computer system related tothis indication
// (AlertingManagedElement attribute) is of interest.  This recipe does
// not handle the target URL for the indication.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// None

#filter[0] = “SELECT * FROM InstModification
  WHERE SourceInstance ISA CIM_ComputerSystem
    AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]”
@{Determine if Indications already exist or have to be created} #filter
```

8.2.4.2.7 Registered Name and Version

Indication version 1.1.0

8.2.4.2.8 CIM Server Requirements

Table 380: CIM Server Requirements for Indication

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.4.2.9 CIM Elements

Table 381: CIM Elements for Indication

Element Name	Description
Mandatory Classes	
CIM_IndicationSubscription (8.2.4.2.9.5)	This association defines a subscription to a specific IndicationFilter instance by a specific indication handler (as represented by a ListenerDestinationCIMXML instance).
CIM_ListenerDestinationCIMXML (8.2.4.2.9.9)	A CIM_ListenerDestinationCIMXML describes the destination for CIM Export Messages to be delivered via CIM-XML. ListenerDestinationCIMXML is subclassed from ListenerDestination.
Optional Classes	
CIM_AlertIndication (8.2.4.2.9.1)	This Indication is used to capture events that occur in the profile, but may not be related to a specific part of the model.
CIM_ElementCapabilities (8.2.4.2.9.2)	This associates the QueryCapabilities to the Object-Manager.
CIM_IndicationFilter (8.2.4.2.9.3)	This is for "pre-defined" CIM_IndicationFilter instances.CIM_IndicationFilter defines the criteria for generating an Indication and what data should be returned in the Indication.
CIM_IndicationFilter (8.2.4.2.9.4)	This is for "client defined" CIM_IndicationFilter instances.CIM_IndicationFilter defines the criteria for generating an Indication and what data should be returned in the Indication.
CIM_InstCreation (8.2.4.2.9.6)	CIM_InstCreation is an indication of the creation of a CIM instance. It would be generated when an instance of the SourceInstance class is created (either explicitly or implicitly).
CIM_InstDeletion (8.2.4.2.9.7)	CIM_InstDeletion is an indication of the Deletion of a CIM instance. It would be generated when an instance of the SourceInstance class is deleted from the model (either explicitly or implicitly).
CIM_InstModification (8.2.4.2.9.8)	CIM_InstModification is an indication of the modification or change to a CIM instance. It would be generated when an instance of the SourceInstance class is modified or changed (either explicitly or implicitly).
CIM_QueryCapabilities (8.2.4.2.9.10)	OPTIONAL: Defines the Query execution capabilities of the profile or CIMOM.

8.2.4.2.9.1 CIM_AlertIndication

A CIM_AlertIndication is a specialized type of CIM_Indication that contains information about the severity, cause, recommended actions and other data of a real world event.

CIM_AlertIndication is subclassed from CIM_ProcessIndication.

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

Class Mandatory: false

Table 382: SMI Referenced Properties/Methods for CIM_AlertIndication

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IndicationIdentifier		string	An identifier for the Indication used for correlated indications.
IndicationTime	N	datetime	The time and date of creation of the Indication. The property may be set to NULL if it cannot be determined.
AlertingManagedElement		string	The identifying information of the entity for which this Indication is generated.
AlertingElementFormat		uint16	Valid SMI-S values are "Unknown", "Other", "CIMObjectPath"
AlertType		uint16	Values { "Other", "Communications Alert", "Quality of Service Alert", "Processing Error", "Device Alert", "Environmental Alert", "Model Change", "Security Alert" }
PerceivedSeverity		uint16	Values { "Unknown", "Other", "Information", "Degraded/Warning", "Minor", "Major", "Critical", "Fatal/NonRecoverable" }
ProbableCause		uint16	Many possible values in a value map. See MOF.
SystemCreationClassName		string	
SystemName		string	
ProviderName		string	
Optional Properties/Methods			
CorrelatedIndications		string[]	IndicationIdentifiers whose notifications are correlated with this one.
Description		string	Recommendation.ITU|X733.Additional text
OtherAlertType		string	
OtherSeverity		string	
ProbableCauseDescription		string	
EventID		string	

8.2.4.2.9.2 CIM_ElementCapabilities

CIM_ElementCapabilities represents the association between ManagedElements (i.e.,CIM_ObjectManager) and their Capabilities (e.g., CIM_QueryCapabilities).

CIM_ElementCapabilities is not subclassed from anything.

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

Class Mandatory: false

Table 383: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element (ObjectManager)
Capabilities		CIM_Capabilities	The CIM_QueryCapabilities instance associated with the element.

8.2.4.2.9.3 CIM_IndicationFilter

CIM_IndicationFilter instances that are "pre-defined" are IndicationFilters that are be populated automatically by the profile provider. If a profile implementation cannot support client defined IndicationFilters, the implementation can populate its model with "pre-defined" IndicationFilter instances. "Pre-defined" filters shall include those that are required by the profile, but may also contain additional filters supported by the implementation.

CIM_IndicationFilter is subclassed from CIM_ManagedElement.

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 384: SMI Referenced Properties/Methods for CIM_IndicationFilter (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
Query		string	
QueryLanguage		string	This should be DMTF:CQL, but may be WQL or SMI-S V1.0. WQL and SMI-S V1.0 are deprecated in favor of DMTF:CQL.
Optional Properties/Methods			
SourceNamespace	N	string	For instances in the Interop-Namespace, this shall be the namespace where the indications are to originate. For instances in the namespace where the indications are to originate (e.g., the namespace of the profile that supports the filter), this may be NULL to indicate the Filter is registered in the Namespace where the indications originate.
ElementName	N	string	This should be NULL for pre-defined indication filters.

8.2.4.2.9.4 CIM_IndicationFilter

CIM_IndicationFilter instances that are "client defined" are IndicationFilters that are be created by a client using CreateInstance. If a profile implementation can support client defined IndicationFilters, the implementation would support "client defined" IndicationFilter instances. The implementation shall support "client defined" filters that are defined by SMI-S profile as mandatory, but may also support additional filters supported by the implementation (See QueryCapabilities).

CIM_IndicationFilter is subclassed from CIM_ManagedElement.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 385: SMI Referenced Properties/Methods for CIM_IndicationFilter (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
Query		string	
QueryLanguage		string	This should be DMTF:CQL, but may be WQL or SMI-S V1.0. WQL and SMI-S V1.0 are deprecated in favor of DMTF:CQL.
Optional Properties/Methods			
SourceNamespace	N	string	The path to a local namespace where the Indications originate. If NULL, the namespace of the Filter registration is assumed.
ElementName		string	A Client Defined user-friendly string that identifies the Indication Filter.

8.2.4.2.9.5 CIM_IndicationSubscription

A CIM_IndicationSubscription is not subclassed from anything.

Created By : CreateInstance
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: true

Table 386: SMI Referenced Properties/Methods for CIM_IndicationSubscription

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Filter		CIM_IndicationFilter	
Handler		CIM_ListenerDestination	

Table 386: SMI Referenced Properties/Methods for CIM_IndicationSubscription

Property	Flags	Type	Description & Notes
RepeatNotificationPolicy		uint16	SMI-S supports a restricted set of values. ValueMap { "2", "3", "4" }, Values { "None", "Suppress", "Delay" }
Optional Properties/Methods			
RepeatNotificationInterval		uint64	Mandatory if the RepeatNotificationPolicy is "Suppress" or "Delay".
RepeatNotificationGap		uint64	Mandatory if the RepeatNotificationPolicy is "Delay".
RepeatNotificationCount		uint16	Mandatory if the RepeatNotificationPolicy is "Suppress" or "Delay".

8.2.4.2.9.6 CIM_InstCreation

CIM_InstCreation notifies a handler when a new instance (of a class defined in the Filter QueryString) is created.

CIM_InstCreation is subclassed from CIM_InstIndication.

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 387: SMI Referenced Properties/Methods for CIM_InstCreation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IndicationIdentifier		string	An identifier for the Indication used for correlated indications.
IndicationTime		datetime	The time and date of creation of the Indication. The property may be set to NULL if it cannot be determined.
SourceInstance		string	A copy of the instance that changed to generate the Indication. SourceInstance contains the current values of the properties selected by the Indication Filter's Query.
SourceInstanceModelPath		string	The Model Path of the SourceInstance.
Optional Properties/Methods			
CorrelatedIndications		string[]	IndicationIdentifiers whose notifications are correlated with this one.

8.2.4.2.9.7 CIM_InstDeletion

CIM_InstDeletion notifies a handler when a new instance (of a class defined in the Filter QueryString) is deleted.

CIM_InstDeletion is subclassed from CIM_InstIndication.

Created By : External

Modified By : External
Deleted By : External
Class Mandatory: false

Table 388: SMI Referenced Properties/Methods for CIM_InstDeletion

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IndicationIdentifier		string	An identifier for the Indication used for correlated indications.
IndicationTime		datetime	The time and date of creation of the Indication. The property may be set to NULL if it cannot be determined.
SourceInstance		string	A copy of the instance that changed to generate the Indication. SourceInstance contains the current values of the properties selected by the Indication Filter's Query.
SourceInstanceModelPath		string	The Model Path of the SourceInstance.
Optional Properties/Methods			
CorrelatedIndications		string[]	IndicationIdentifiers whose notifications are correlated with this one.

8.2.4.2.9.8 CIM_InstModification

CIM_InstModification notifies a handler when a new instance (of a class defined in the Filter QueryString) is modified or changed. To avoid undue effort on Providers, the select list (in the query filter) for this indication should only call for properties that are needed.

CIM_InstModification is subclassed from CIM_InstIndication.

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 389: SMI Referenced Properties/Methods for CIM_InstModification

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IndicationIdentifier		string	An identifier for the Indication used for correlated indications.
IndicationTime		datetime	The time and date of creation of the Indication. The property may be set to NULL if it cannot be determined.
SourceInstance		string	A copy of the instance that changed to generate the Indication. SourceInstance contains the current values of the properties selected by the Indication Filter's Query.
SourceInstanceModelPath		string	The Model Path of the SourceInstance.

Table 389: SMI Referenced Properties/Methods for CIM_InstModification

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
CorrelatedIndications		string[]	IndicationIdentifiers whose notifications are correlated with this one.
PreviousInstance		string	A copy of the 'previous' instance whose change generated the Indication. PreviousInstance contains 'older' values of an instance's properties (as compared to SourceInstance), selected by the IndicationFilter's Query.

8.2.4.2.9.9 CIM_ListenerDestinationCIMXML

CIM_ListenerDestinationCIMXML is subclassed from CIM_ListenerDestination.

Created By : CreateInstance
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: true

Table 390: SMI Referenced Properties/Methods for CIM_ListenerDestinationCIMXML

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A client defined user-friendly string that identifies the CIMXML Listener destination.
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
PersistenceType		uint16	For SMI-S, this shall be "2" (permanent) or "3" (transient)
Destination		string	The destination URL to which CIM-XML Export Messages are to be delivered. The scheme prefix shall be consistent with the DMTF CIM-XML specifications.If a scheme prefix is not specified, the scheme "http:" shall be assumed.

8.2.4.2.9.10 CIM_QueryCapabilities

This class defines the capabilities of the Object Manager or Provider associated via ElementCapabilities.

CIM_QueryCapabilities is subclassed from CIM_Capabilities.

An instance of this class may or may not exist. If the profile supports client defined indication filters, then an instance shall exist.

Created By : Static
Modified By : Static

Deleted By : Static
Class Mandatory: false

Table 391: SMI Referenced Properties/Methods for CIM_QueryCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	This is a user-friendly name of the capabilities instance.
CQLFeatures		uint16[]	Enumeration of CQL features supported by an Object Manager or Provider associated via ElementCapabilities. (See DSP0202 CIM Query Language Specification for a normative definition of each feature.) Values {"Basic Query", "Simple Join", "Complex Join", "Time", "Basic Like", "Full Like", "Array Elements", "Embedded Objects", "Order By", "Aggregations", "Subquery", "Satisfies Array", "Distinct", "First", "Path Functions"}

8.2.4.2.10 Related Standards

Table 392: Related Standards for Indication

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.9.1	DMTF
CIM Query Specification	1.0	DMTF

8.2.4.3 Object Manager Adapter Subprofile

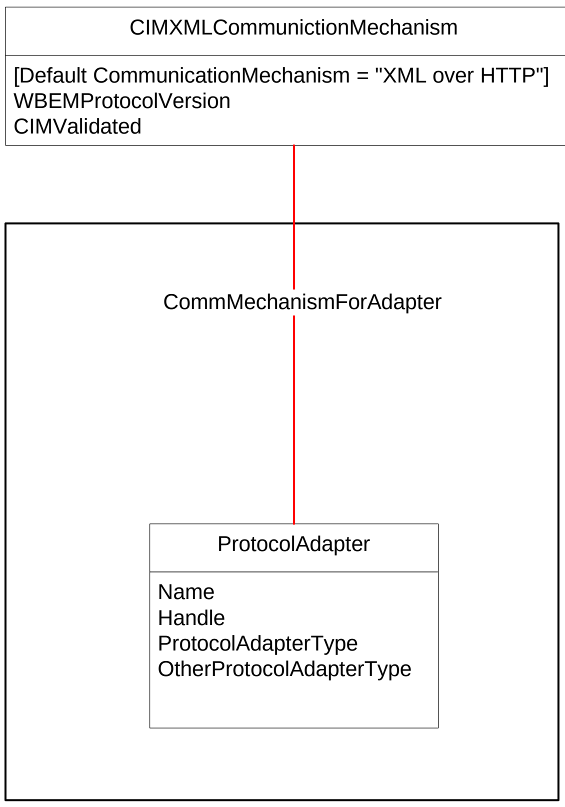
8.2.4.3.1 Description

The ObjectManagerAdapter model defines the protocol adapters that are supported for a CIM Server. This model is optional for the CIM Server Profile. If implemented, the ObjectManagerAdapterModel shall adhere to the “required elements” table.

Instance Diagram

ObjectManagerAdapter subprofile is not advertised.

Figure 70: ObjectManagerAdapter Subprofile Model



8.2.4.3.2 Health and Fault Management

Not defined in this standard.

8.2.4.3.3 Cascading Considerations

Not defined in this standard.

8.2.4.3.4 Supported Subprofiles and Packages

None.

8.2.4.3.5 Methods of the Profile

None.

8.2.4.3.6 Client Considerations and Recipes

None.

8.2.4.3.7 Registered Name and Version

Object Manager Adapter version 1.1.0

8.2.4.3.8 CIM Server Requirements

Table 393: CIM Server Requirements for Object Manager Adapter

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.4.3.9 CIM Elements

Table 394: CIM Elements for Object Manager Adapter

Element Name	Description
Mandatory Classes	
CIM_CommMechanismForObjectManagerAdapter (8.2.4.3.9.1)	
CIM_ObjectManagerAdapter (8.2.4.3.9.2)	

8.2.4.3.9.1 CIM_CommMechanismForObjectManagerAdapter

Class Mandatory: true

Table 395: SMI Referenced Properties/Methods for CIM_CommMechanismForObjectManagerAdapter

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ObjectManagerAdapter	The specific ObjectManagerAdapter whose communication mechanism with the CIM Object Manager is described.
Dependent		CIM_ObjectManagerCommunicationMechanism	The encoding/protocol/set of operations that may be used to communicate between the Object Manager and the referenced ObjectManagerAdapter.

8.2.4.3.9.2 CIM_ObjectManagerAdapter

Class Mandatory: true

Table 396: SMI Referenced Properties/Methods for CIM_ObjectManagerAdapter

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ElementName		string	
Handle		string	
AdapterType		uint16	
OperationalStatus		uint16[]	
Started		boolean	
StartService()			
StopService()			
Optional Properties/Methods			
OtherAdapterTypeDescription		string	
StatusDescriptions		string[]	This shall not be NULL if "Other" is identified in OperationalStatus

8.2.4.3.10 Related Standards

Table 397: Related Standards for Object Manager Adapter

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.5 Security Profiles and Subprofiles

EXPERIMENTAL

8.2.5.1 Security Profile

8.2.5.1.1 Description

Overview

Security requirements can be divided into four major categories: authentication, authorization, confidentiality, and integrity (including non-repudiation), brief definitions follow. Authentication is verifying the identity of an entity (client or server). Authorization is deciding if an entity is allowed to perform a given operation. Confidentiality is restricting information to only those intended recipients. Integrity is guaranteeing that information, passed between entities, has not been modified.

This top level Security Profile primarily addresses authentication; 8.2.4.1.1.4, "HTTP Security" of the Server Profile addresses confidentiality; and authorization is addressed by 8.2.5.2, "Authorization Subprofile".

Issues not covered include threat models, protection against specific attack vectors, (such as denial of service, replay, buffer overflow, man in the middle, etc.), topics related to key management, and data integrity. Development of threat models, and specific attack countermeasures required for robust security elements, such as integrity has been left for future work.

Security concerns occur in three areas of an SMI-S implementation.

First an SMI-S Server may also be a client of other services, (sometimes conceptualized as a devices.). Those services, (or devices), may require a login before discovery or operations are allowed to be performed. The information needed to perform this login is generically referred to as “credentials”, (or in the case of devices as “device credentials”). An SMI-S server or provider needs to obtain these credentials in order to talk to the service, and they should be provided confidentially.

Second, an SMI-S Server may need to authenticate an SMI-S Client. Not all Clients may be allowed to query the object model, and not all Clients may be allowed to perform operations on objects in the model. The SMI-S Server is responsible for the process of authenticating credentials received from an SMI-S Client. Successful authentication establishes a trust relationship, which is represented on the SMI-S Server by an authenticated Identity. Authenticating the client is the first step in determining what that Client is allowed to do.

Thirdly, should implementers of an SMI-S Server be unaware of secure development practices, attackers may be able to exploit insecurely developed implementations. (Note, potential attacks might include, but not be limited to buffer overflows, obtaining secure information handled by the SMI-S implementation, like passwords, etc.) In an effort to increase the general knowledge of SMI-S developers, for secure development practices, one resources is referenced: Building Secure Software by Gary McGraw and John Viega (ISBN: 020172152X).

Security Subprofiles

This profile describes minimum requirements on Authentication and Authorization services of an SMI-S Server, where an authenticated Identity is assumed to be authorized. This capability is then extended and constrained by various subprofiles. These are summarized in Table 398, “Security Subprofiles”.

Table 398: Security Subprofiles

Security Subprofile	Depends on	References	Description
3rdPartyAuthentication	IdentityManagement Security	CredentialManagement	Specifies additional requirements on an SMI-S Server when it is also a client of a 3rd party authentication service
Authorization	Security		Specifies additional requirements on an SMI-S Server that supports an authorization service
CredentialManagement	Security		Specifies additional requirements on an SMI-S Server that is also a client of some other service that enforces security
IdentityManagement	Security		Specifies additional requirements on an SMI-S Server that supports the management of Identities, including establishing Accounts, and defining User and Organizational entities and Groups of those entities.
RBAC	Authorization Security		Specifies additional requirements on an SMI-S Server that supports Role Based Access Control.
ResourceOwnership	Authorization Security	RBAC	Specifies additional requirements on an SMI-S Server that supports the capability to restrict authorization rights.

The purpose of the Security profile is to enable the monitoring and management an entity's rights to act on, (including to view or detect), the operational or management aspects of particular objects within a System. Such an entity is known in CIM by an instance of Identity. With respect to the particular objects, at any point in time an entity is either authenticated or not. This is tracked in the Identity instance as CurrentlyAuthenticated. An Identity with CurrentlyAuthenticated set to True represents a security principal. Authentication is a key criteria for Authorization, where authenticated entities are granted rights to act on particular objects.

Support for this profile declares the ability to discover Identities maintained on an SMI-S Server. Unless modified by a subprofile, entities represented by authenticated Identities are granted all rights to all objects within the scope that the identified entity is known.

This profile contains a number of options. It is up to the profile or subprofile that depends on this profile to specify which options are acceptable

Selecting an Identity

To act on a system which enforces security, a requestor needs to be authenticated. The process of authentication maps a requestor to a well-defined Identity. From a management point of view, rights to act on particular resources of a system are granted to Identities.

Figure 71: "Identity" shows that an Identity instance may be associated with the entity being identified via AssignedIdentity. Commonly this ManagedElement will be an instance of UserContact. UserContact provides information about a user, including UserID.

If AssignedIdentity is not used, an alternative is to use a subclass of Identity with additional properties and to algorithmically equate those properties to a requesting entity in a known way. StorageHardwareID instances are an example of the second option. Each StorageHardwareID contains a StorageID that uniquely identifies a requesting port.

An Identity is only valid within some scope. This is defined by an IdentityContext association, typically to a System or RemoteServiceAccessPoint. If there is more than one System or if there are RemoteServiceAccessPoint instances in the Profile namespace, then IdentityContext is mandatory for this profile.

In all cases, the InstanceID of an Identity should be treated as opaque.

Two options are available for managing the Authentication process within a System.

One option is to use the Identity aspect of Account via ConcreteIdentity. The UserID and UserPassword properties of Account are matched to the authentication information provided by a requestor and the associated Identity instances are selected.

The other option is to associate an AuthenticationRule via PolicySetAppliesToElement.

An Account may be used together with an AuthenticationRule.

See the Security Identity Management subprofile for specification of the ability to add Accounts, UserContacts, and Identities to an SMI-S Server.

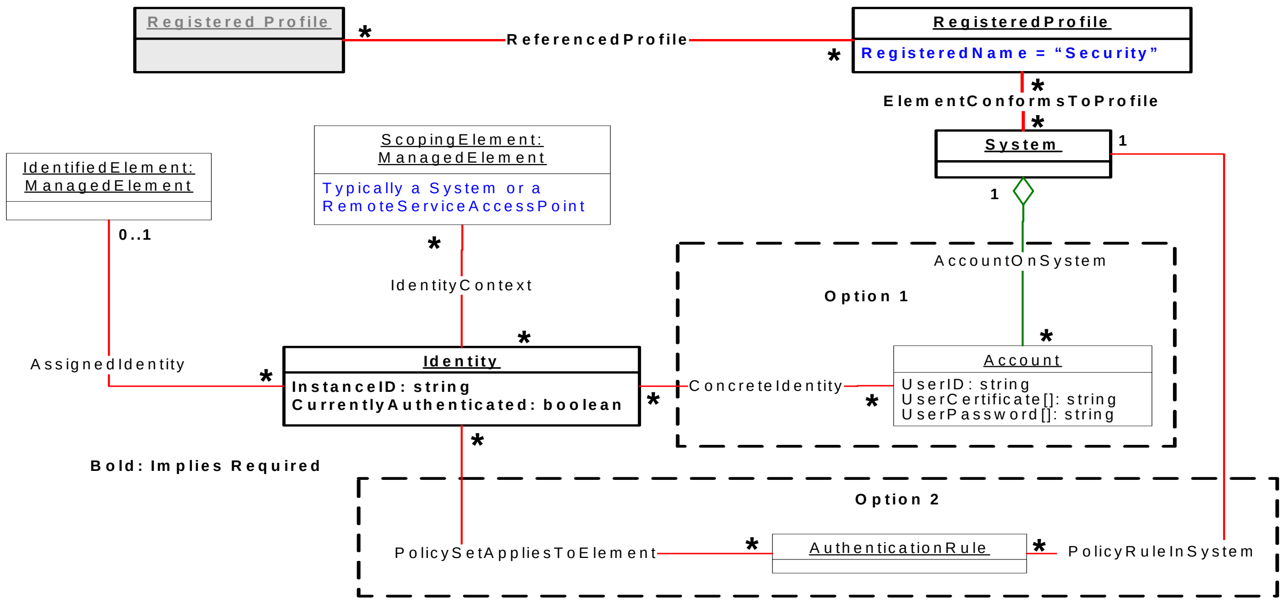


Figure 71: Identity

Authentication Policy

If an AuthenticationRule is not associated with an Identity, then CurrentlyAuthenticated property of Identity is set to True whenever a requestor authenticates to an Identity, and False otherwise.

An AuthenticationRule may be associated with Identity via PolicySetAppliesToElement.

If specified, it further defines or constrains the authentication for the associated Identity. For instance, a PolicyTimePeriodCondition may be associated to the AuthenticationRule via PolicySetValidationPeriod. Additionally, there are a number of specific subclasses of AuthenticationCondition which may be used to further qualify the AuthenticationRule. The CurrentlyAuthenticated property of one of these Identity instances is set to True whenever a requestor matches to an Identity and the conditions of the AuthenticationRule are met, and is set to False otherwise.

The incorporating profile or subprofile shall specify which subclasses of Identity and AuthenticationRule are allowable.

Authorization

Unless further constrained by a subprofile or by an incorporating profile, if the CurrentlyAuthenticated property of Identity is set to True, then the identified requesting entity is granted permission to perform any supported action on all elements of the System that conforms to this profile.

See the Security Authorization and Security RBAC subprofiles for additional specification of SMI-S conformant authorization rules.

8.2.5.1.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.5.1.3 Cascading Considerations

Not defined in this standard.

8.2.5.1.4 Supported Subprofiles and Packages

Table 399: Supported Subprofiles for Security

Registered Subprofile Names	Mandatory	Version
Security CredentialManagement	No	1.1.0
Security IdentityManagement	No	1.1.0
Security Authorization	No	1.1.0

8.2.5.1.5 Methods of the Profile

None.

8.2.5.1.6 Client Considerations and Recipes

Included is one recipe to list and classify Identities.

8.2.5.1.6.1 List and classify Identities

```
// DESCRIPTION
// This recipe describes how to identify existing Identities and classify them
// by type. The current authentication status of each Identity is determined.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
```

```
// 1. The name of a top-level System instance in the Security Profile has
// previously been discovered via SLP and is known as $System->.

// MAIN
// Step 1. Locate the known Identities on the system.
$Identities[] = Associators($System->,
     “CIM_IdentityContext”,
     “CIM_Identity”,
     “ElementProvidingContext”,
     “ElementInContext”,
     false,
     false,
     {“CurrentlyAuthenticated”})
// Verify that one or more Identities exist on the system.
if ($Identities[] == null || $Identities[].length < 1) {
    <ERROR! No known Identities on the system>
}

// Step 2. Create a list entry for each Identity and classify it by type.
#IdentityType[]// contains {“HardwareID”, “Entity”, “Unknown”}
#IdentityUserID[]// contains UserID if the Identity is for an Account.
for (#i in $Identities[]) {

    #IsAuthenticated[#i] = $Identities[#i].CurrentlyAuthenticated

    $Identity-> = $Identities[#i].getObjectPath()
    if ($Identity-> ISA CIM_StorageHardwareID) {
     #IdentityType[#i] = “HardwareID”
     #IdentityUserID[#i] = ““
    } else if ($Identity-> ISA CIM_IPNetworkID) {
     #IdentityType[#i] = “IPNetworkID”
     #IdentityUserID[#i] = ““
    } else {

     // Determine the matching entity type
     $Entity[] = Associators($Identity->,
        “CIM_AssignedIdentity”,
        “CIM_ManagedElement”,
        “IdentityInfo”,
        “ManagedElement”,
        false,
        false,
        {“UserID”})

     // There will be at most one matching entity
     if ($Entity[] == null || $Entity[].length == 0) {
         // Not enough information present to determine type of Identity
```

```
              #IdentityType[#i] = “Unknown”
              #IdentityUserID[#i] = ““
          } else {
              // Determine the matching entity type.
              if ($Identity[#i] ISA CIM_UserContact) {
             // Identity of a User
             #IdentityType[#i] = “User”
             #IdentityUserID[#i] = $Entity[0].UserID
              } else {
             // Identity of some other type of Entity
             #IdentityType[#i] = “Entity”
             #IdentityUserID[#i] = ““
              }
          }
         }
         // Determine if there is an associated Account.
         $Entity[] = Associators($Identity->,
              “CIM_ConcreteIdentity”,
              “CIM_Account”,
              “SameElement”,
              “SystemElement”,
              null,
              null,
              {“UserID”})
         if ($Entity[] != null && $Entity[].length = 1) {
          #IdentityUserID[#i] = Entity[1].UserID
         }
     }
```

8.2.5.1.7 Registered Name and Version

Security version 1.1.0

8.2.5.1.8 CIM Server Requirements

Table 400: CIM Server Requirements for Security

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.5.1.9 CIM Elements

Table 401: CIM Elements for Security

Element Name	Description
Mandatory Classes	
CIM_RegisteredProfile (8.2.5.1.9.11)	Describes support for the Security Profile
CIM_System (8.2.5.1.9.12)	System containing elements supporting Authentication and basic Authorization
Optional Classes	
CIM_Account (8.2.5.1.9.1)	Represents information about an entity that may act on resources
CIM_AccountOnSystem (8.2.5.1.9.2)	Identifies the conformant element
CIM_AssignedIdentity (8.2.5.1.9.3)	Identifies the conformant element
CIM_AuthenticationRule (8.2.5.1.9.4)	A policy the defines the rules for authenticating an Identity
CIM_ConcreteIdentity (8.2.5.1.9.5)	Identifies the conformant element
CIM_Identity (8.2.5.1.9.6)	Represents an entity that may act on resources
CIM_IdentityContext (8.2.5.1.9.7)	Identifies the conformant element
CIM_ManagedElement (8.2.5.1.9.8)	Represents either an entity or a resource
CIM_PolicyRuleInSystem (8.2.5.1.9.9)	Identifies the System which supports the associated PolicyRule.
CIM_PolicySetAppliesToElement (8.2.5.1.9.10)	Identifies the conformant element
Mandatory Indications	
SELECT * FROM CIM_InstMethodCall WHERE ANY element in Error[*] SATISFIES element.CIMStatusCode = 2	Deprecated WQL - Capture all Access Denied errors
SELECT * FROM CIM_InstMethodCall WHERE ANY element in Error[*] SATISFIES element.CIM_InstMethodCall::CIMStatusCode = 2	CQL - Capture all Access Denied errors

8.2.5.1.9.1 CIM_Account

Represents information about an entity that may act on resources

Class Mandatory: false

Table 402: SMI Referenced Properties/Methods for CIM_Account

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key
UserID		string	
UserPassword		string[]	
OrganizationName		string[]	

8.2.5.1.9.2 CIM_AccountOnSystem

Identifies the conformant element
Class Mandatory: false

Table 403: SMI Referenced Properties/Methods for CIM_AccountOnSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	Key
PartComponent		CIM_Account	Key

8.2.5.1.9.3 CIM_AssignedIdentity

Identifies the conformant element
Class Mandatory: false

Table 404: SMI Referenced Properties/Methods for CIM_AssignedIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	Key
IdentityInfo		CIM_Identity	Key

8.2.5.1.9.4 CIM_AuthenticationRule

A policy the defines the rules for authenticating an Identity
Class Mandatory: false

Table 405: SMI Referenced Properties/Methods for CIM_AuthenticationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.1.9.5 CIM_ConcreteIdentity

Identifies the conformant element
Class Mandatory: false

Table 406: SMI Referenced Properties/Methods for CIM_ConcreteIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SameElement		CIM_ManagedElement	Key
SystemElement		CIM_ManagedElement	Key

8.2.5.1.9.6 CIM_Identity

Represents an entity that may act on resources
Class Mandatory: false

Table 407: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	Indicates whether or not an entity has been authenticated to use this Identity.

8.2.5.1.9.7 CIM_IdentityContext

Identifies the conformant element
Class Mandatory: false

Table 408: SMI Referenced Properties/Methods for CIM_IdentityContext

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementProvidingContext		CIM_ManagedElement	Key
ElementInContext		CIM_Identity	Key

8.2.5.1.9.8 CIM_ManagedElement

Represents either an entity or a resource
Class Mandatory: false
No specified properties or methods.

8.2.5.1.9.9 CIM_PolicyRuleInSystem

Identifies the System which supports the associated PolicyRule.
Class Mandatory: false

Table 409: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_PolicyRule	Key

8.2.5.1.9.10 CIM_PolicySetAppliesToElement

Identifies the conformant element
Class Mandatory: false

Table 410: SMI Referenced Properties/Methods for CIM_PolicySetAppliesToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	Key
PolicySet		CIM_PolicySet	Key

8.2.5.1.9.11 CIM_RegisteredProfile

Describes support for the Security Profile
Class Mandatory: true

Table 411: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization		uint16	SNIA
RegisteredName		string	The Profile name.
RegisteredVersion		string	

8.2.5.1.9.12 CIM_System

System containing elements supporting Authentication and basic Authorization
Class Mandatory: true

Table 412: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.1.10 Related Standards

Table 413: Related Standards for Security

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
Representation of CIM using XML	2.2.0	DMTF
WBEM Discovery using SLP	1.0.0	DMTF
WBEM URI Specification	1.0.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.2 Authorization Subprofile

8.2.5.2.1 Description

The Authorization subprofile extends the Security profile. The Authorization subprofile specifies base support to enable management of the rights of particular subjects to perform specific operations on selected target elements within a CIM Service.

Authorization

Assuming successful authentication, the system needs to assure that the requestor is authorized to perform the request. Figure 72: "Authorization" shows the elements needed to manage authorization. This subprofile constrains the Security profile. When applied, authenticated requestors are not automatically granted all rights. Instead, this subprofile automatically denies all rights unless specifically granted. See “Authorization Rights” in 8.2.5.2.1for a detailed description of rights.

Rights to act on a resource are granted or denied to entities using the ChangeAccess method of a PrivilegeManagementService instance. Resources and entities are represented by ManagedElements and Identities, respectively. Granted rights are displayed using the ShowAccess method.

In complex environments two additional associations are used to select the correct PrivilegeManagementService:

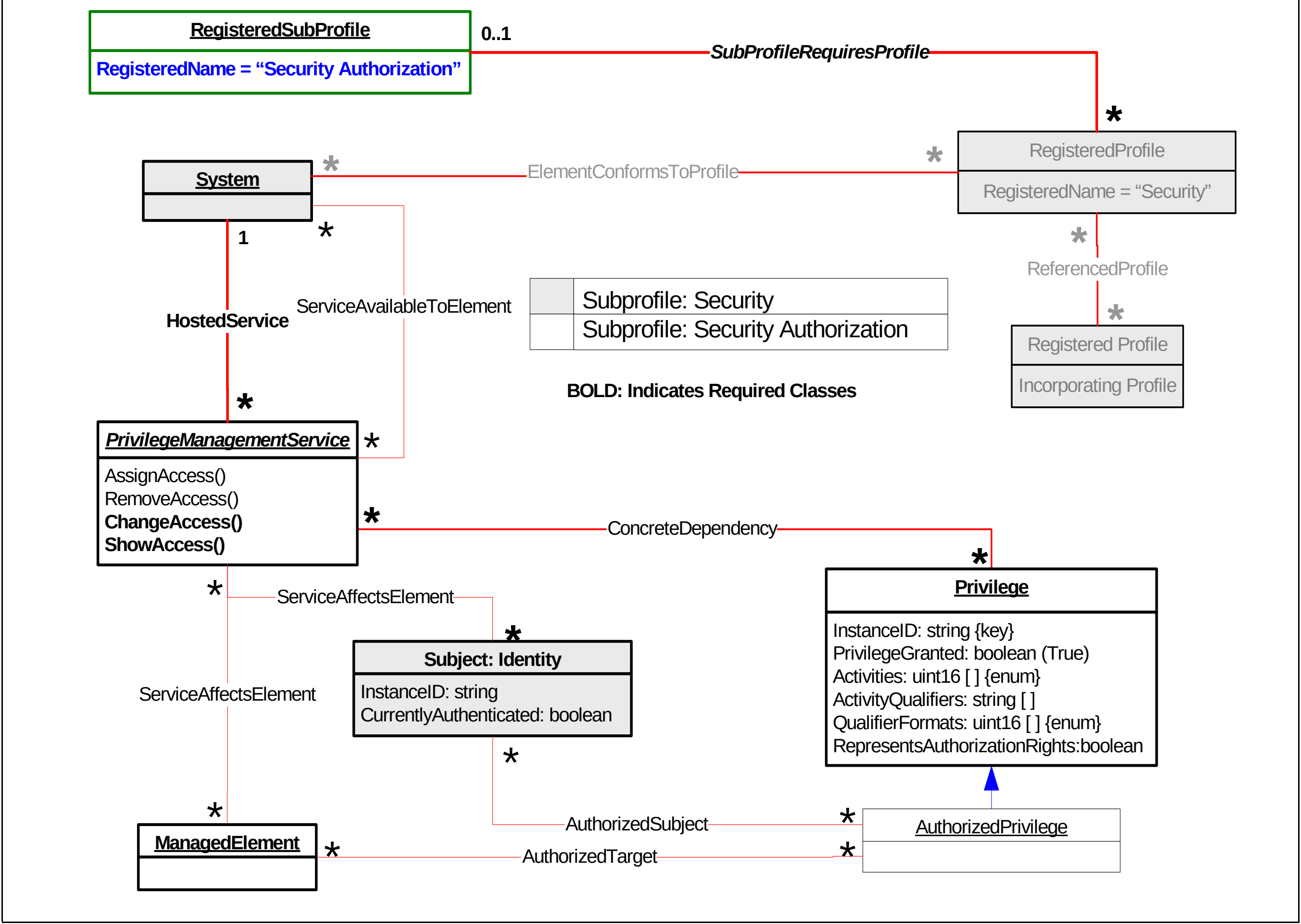
- The first is ServiceAvailableToElement, which is not mandatory unless there are more than one System instances in the profile namespace. If there are more than one System, then a ServiceAvailableToElement association between the applicable System and the PrivilegeManagementService is mandatory.
- The second is ServiceAffectsElement associations, which are not mandatory unless there are more than one PrivilegeManagementService instances in the profile namespace. If there are more than one PrivilegeManagementService, then a ServiceAffectsElement association between the PrivilegeManagementService and elements that it can operate on is mandatory.

Sets of rights are represented by Privilege instances. An implementation may publish Privilege instances to use as templates for granting rights. This is done by associating Privilege instances to a PrivilegeManagementService instance via ConcreteDependency.

When a set of rights are granted, the implementation may make this concrete by instantiating an AuthorizedPrivilege instance to represent the set of rights and then using AuthorizedSubject and

AuthorizedTarget to associate the authorized Identity and resource. Profiles that incorporate this subprofile may require these associations to be made explicit.

Figure 72: Authorization



A request is made to act on some element. In the case of Intrinsic Methods, this is first a Namespace, which may or may not be modeled, and which may propagate sub-requests to one or more other ManagedElements published in that Namespace. In the case of Extrinsic Methods, the element shall be the ManagedElement which supports the method.

If it is desired to place restrictions on all elements within a Namespace, then modeling the Namespace is required. The Namespace instance is used as the "ManagedElement" instance shown in Figure 72: "Authorization".

Good practice requires the implementation of each ManagedElement to enforce authorization. A simpler, but less robust model allows the ObjectManager or the Provider of the ManagedElement to authorize the request. Since enforcement at either the ObjectManager or Provider level does not assure there are no back-doors to the implementation, and since the ObjectManager has limited semantic information about the model elements, (and therefore the meaning of the rights passed in Privilege instances,) these simpler schemes are not always applicable. As a result, this Profile RECOMMENDS the more general model.

When the request is delivered, the Identity of the requestor shall be available to the AuthorizationService. The Provider for a ManagedElement can then ask the AuthorizationService to verify that the requested action is allowed. The AuthorizationService maps the request to the rights specified by the Activities, ActivityQualifiers, and QualifierFormat properties of AuthorizedPrivilege. The means for the Provider of a ManagedElement to ask this question of the AuthorizationService is not specified by this Profile.

The client shall use either ChangeAccess (recommended), or AssignAccess and RemoveAccess to grant or deny rights.

Authorization Rights

Rights are encoded within the properties of Privilege, two of which operate on all rights defined by the Privilege instance and three of which define a set of rights. The Privilege global properties are:

- PrivilegeGranted: This boolean controls whether the rights defined by the instance are granted or denied[1]. The default is TRUE.
- RepresentsAuthorizationRights: This boolean controls whether the rights defined by the instance specifies access rights or authorization rights. Access rights grant a subject access to a target. Authorization rights grant a subject the right to assign, change, or remove the specified rights for a target to other subjects. The default is FALSE.

The properties which define rights are each an indexed array. Corresponding array entries across all three represent a single access or authorization right. These properties are:

- Activities: Each entry is an enumeration that specifies whether the corresponding right is “Read”. “Write”, “Execute”, “Create”, “Delete”, or “Detect”.
- ActivityQualifiers: Each entry is a string that qualifies the corresponding Activity entry. For instance, if the Activities is “Execute”, then the corresponding entry might be a comma separated list of method names. An entry may be NULL which specifies that the corresponding Activity is not qualified.
- QualifierFormats: Each entry is an enumeration that specifies the format of the string in the corresponding ActivityQualifiers entry. If an ActivityQualifiers entry is not NULL then the corresponding QualifierFormats entry shall be specified. Otherwise it shall be NULL. Possible enumerations are: “Class Name”, “<Class.>Property”, “<Class.>Method”, “Object Reference”, “Namespace”, “URL”, “Directory/File Name”, “Command Line Instruction”, “SCSI Command”, and “Packet”. In the “Execute” example above, the QualifierFormats entry shall be “<Class.>Method”.

Specification of allowable combinations of rights is left to the profiles or subprofiles that incorporate this subprofile.

Authorization Policy

The default authorization policy is to deny all requests that are not explicitly granted via either an AuthorizationPolicy or by an explicit ChangeAccess or AssignAccess method.

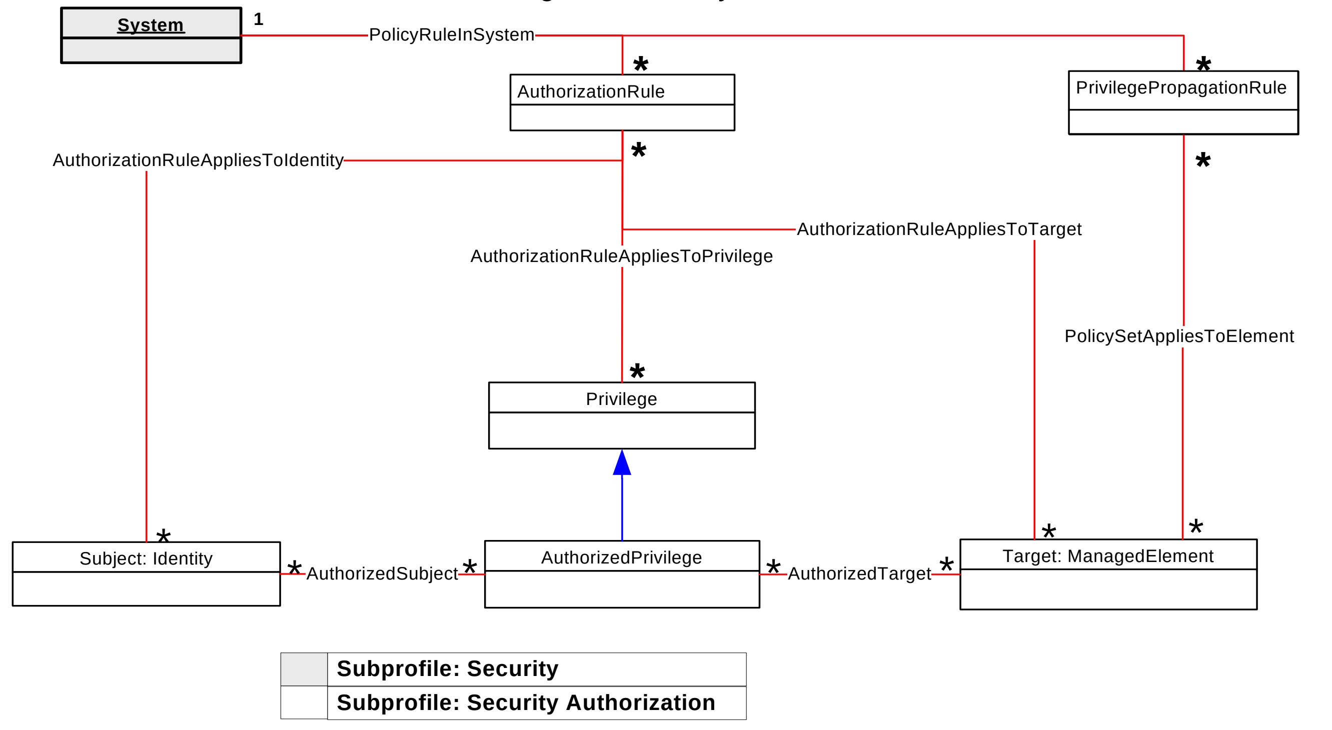
An AuthorizationRule may be specified as part of a ChangeAccess method. The AuthorizationRule may then grant rights implicitly.

Identities, Privileges, and target ManagedElements may be associated to an AuthorizationRule by AuthorizationRuleAppliesToIdentity, AuthorizationRuleAppliesTo-AuthorizedPrivilege, and AuthorizationRuleAppliesToTarget respectively. This is shown in Figure 73: "Policy Rules". When an AuthorizedPrivilege, is added to the AuthorizationRule, an AuthorizedSubject or AuthorizedTarget may be instantiated.

1. When used with ChangeAccess, the meaning of PrivilegeGranted changes to specify whether the rights defined by the instance are added or subtracted.

The details of the specification of AuthorizationRules are left to the profiles and subprofiles that reference this subprofile.

Figure 73: Policy Rules



Privilege Propagation Policies

In most instances, the propagation rules for a particular type of target element are clear and apply to all subjects. In this case, the semantics of the target element can imply a particular propagation policy. When a subject may select from multiple possible propagation strategies for a target element, there needs to be a means to specify the propagation strategy. Subclasses of PrivilegePropagationRule provide this ability. When associated with a target element via PolicySetAppliesToElement, the PrivilegePropagationRule specifies the default policy to apply. When associated to an AuthorizedPrivilege, via PolicySetAppliesToElement, the PrivilegePropagationRule specifies the policy used to propagate the named rights.

When an AuthorizedPrivilege instance representing propagated rights is returned, it will have the IsPropagated boolean set to True.

The details of the specification of PrivilegePropagationRules are left to the profiles and subprofiles that reference this subprofile.

For illustrative purposes only, the following example illustrates the creation of a PrivilegePropagationRule using QueryCondition (not shown) and MethodAction (not shown) classes associated via PolicyConditionInPolicyRule (not shown) and PolicyActionInPolicyRule (not shown) respectively. The QueryLanguage property of the QueryCondition and MethodAction instances shall be set to “2”, meaning “CQL”. Assume the QueryCondition.QueryResultName is set to “SNIA_AuthorizationConditionExample” and its Query property set to

```
“SELECT (M.SourceInstanceHost || “/” || M.SourceInstanceModelPath) AS PMSPath,
                     M.MethodParameters.Subject,
                     ObjectPath(E) AS Target,
                     M MethodParameters.Privileges
FROM
     CIM_InstMethodCall M,
     CIM_Collection C,
     CIM_MemberOfCollection MoC,
     CIM_ManagedElement E
     CIM_PolicySetAppliesToElement PSATE
     CIM_PolicyConditionInPolicyRule PCIPR
     CIM_PrivilegePropagationRule PPR
WHERE
     M.MethodName = “ChangeAccess”
AND M.ReturnValue = 0
AND M.PreCall = FALSE
AND M.MethodParameters.Target ISA CIM_Collection
AND M.Target = MoC.Collection
AND ObjectPath(E) = MoC.Element
AND ObjectPath() = PCIPR.PartComponent
AND ObjectPath(PPR) = PCIPR.GroupComponent
AND ObjectPath(PPR) = PSATE.PolicySet
AND ObjectPath(E) = PSATE.ManagedElement”
```

This assures that this query is being run on behalf of a PrivilegePropagationRule that is applied to the Collection. This assures that propagation does not pass through collections that are not appropriate.

The corresponding MethodAction instance would have its Query property set to

```
  “SELECT (Ex.PMSPath || “.” || “ChangeAccess”) AS Methadone,
                    Ex.Subject AS Subject,
                    Ex.Target AS Target,
                    NULL AS PropagationPolicies,
                    Ex.Privileges AS Privileges
  FROM SNIA_AuthorizationConditionExample Ex”
```

The ChangeAccess method enables a client to specify a PrivilegePropagationRule to use while assigning rights. (See Figure 73: "Policy Rules".)

Reporting Granted Rights

Granted rights are reported using the ShowAccess method. (See Figure 72: "Authorization".) This method takes as input one or both of a subject Identity and target ManagedElement. Output is a list of Identity, Privilege, target triples that represent granted Privileges. This output shall reflect a consistent current state at the time of the call, regardless of whether or not corresponding instances of AuthorizedPrivilege, AuthorizedTarget, and AuthorizedSubject have been instantiated.

8.2.5.2.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.5.2.3 Cascading Considerations

Not defined in this standard.

8.2.5.2.4 Supported Subprofiles and Packages

None.

8.2.5.2.5 Methods of the Profile

None.

8.2.5.2.6 Client Considerations and Recipes

8.2.5.2.6.1 Show access rights

```
// DESCRIPTION
// This recipe describes how to identify the authorized subjects and their
// rights for a specified resource.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The name of a top-level System instance in the Security Profile has
// previously been discovered via SLP and is known as $System->.
// 2. The name of a managed element on $System-> whose authorized subjects and
// rights has previously been discovered and is known as $Resource->.

// This function locates the PrivilegeManagementService that manages the
// specified managed element. If no such service is located, null is returned.
sub CIMObjectPath GetPrivilegeServiceForElement(CIMObjectPath[] $Services->[],
     CIMObjectPath $Resource->) {

    $Service-> = null
    // Verify that there is one or more instance of PrivilegeManagementService
    // hosted by the system.
    if ($Services->[] != null && $Services->[] > 0) {
     // Locate the service that manages the privileges of the specified
     // managed element.
     $ResourceServices->[] = AssociatorNames($Resource->,
        “CIM_ServiceAffectsElement”,
        “CIM_PrivilegeManagementService”,
        “UserOfService”,
        “ServiceProvided”)
     if ($ResourceServices->[] != null || $ResourceServices->[].length > 0) {
         for (#i in $ResourceServices->[]) {
        for (#j in $Services->[]) {
            if ($ResourceServices->[#i] == $Services->[#j])) {
           $Service-> = Services->[#j]
           break
            }
        }
         }
     }
    }
    return $Service->
```

```
}

// MAIN
// Step 1. Locate the PrivilegeManagementServices on the system.
$PrivilegeServices->[] = AssociatorNames($System->,
     “CIM_HostedService”,
     “CIM_PrivilegeManagementService”,
     “Antecedent”,
     “Dependent”)

// There must be exactly one PrivilegeManagementService for the managed element.
$PrivilegeService-> = &GetPrivilegeServiceForElement($PrivilegeServices->[],
     $Resource->)
if ($PrivilegeService-> == null) {
    <EXIT! The required PrivilegeManagementService was not found>
}

// Step 2. Retrieve the authorized subjects and their rights for the specified
// resource.
%InArgs[“Subject”] = null
%InArgs[“Target”] = $Resource->
#Result = InvokeMethod($PrivilegeService->,
     “ShowAccess”,
     %InArgs[],
     %OutArgs[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Retrieving access for the specified resource failed>
}

// Step 3. Retrieve the references to the Identities (or other subjects)
// authorized for the resource.
$OutSubjects->[] = %OutArgs[“OutSubjects”]

// Step 4. Retrieve the references to the Privileges corresponding to the
// subject entries.
$OutPrivileges->[] = %OutArgs[“Privileges”]
```

8.2.5.2.6.2 Grant an access right

```
// DESCRIPTION
// This recipe describes how to apply a set of rights to a given resource
// and subject.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The name of a top-level System instance in the Security Profile has
// previously been discovered via SLP and is known as $System->.
// 2. The name of a managed element on $System-> has previously been
// discovered and is known as $Resource->.
// 3. The name of a subject has previously been discovered and is known as
// $Subject->.
// 4. A container of activities to be granted or denied is known as #Activity[].
// 5. A container of additional information related to the activities is known
// as #ActivityQualifiers[].
// 6. A container of sematic descriptions of the formats of the elements in
// #ActivityQualifiers[] is known as #QualifierFormats[].

// MAIN
// Step 1. Locate the PrivilegeManagementServices on the system.
$PrivilegeServices->[] = AssociatorNames($System->,
     “CIM_HostedService”,
     “CIM_PrivilegeManagementService”,
     “Antecedent”,
     “Dependent”)

// There must be exactly one PrivilegeManagementService for the managed element.
$PrivilegeService-> = &GetPrivilegeServiceForElement($PrivilegeServices->[],
     $Resource->)
if ($PrivilegeService-> == null) {
    <EXIT! The required PrivilegeManagementService was not found>
}

// Step 2. Create an Access Privilege
$Privilege = newInstance(“CIM_Privilege”)
$Privilege.PrivilegeGranted = true
$Privilege.RepresentsAuthorizationRights = false
$Privilege.Activity[] = #Activity[]
$Privilege.ActivityQualifiers[] = #ActivityQualifiers[]
$Privilege.QualifierFormats[] = #QualifierFormats[]

// Step 3. Add the right and get the resultant rights.
%InArgs[“Subject”] = $Subject->
%InArgs[“Target”] = $Resource->
%InArgs[“PropagationPolicies”] = null
$Privileges[0] = $Privilege
%InArgs[“Privileges”] = $Privileges[]
#Result = InvokeMethod($PrivilegeService->,
     “ChangeAccess”,
     %InArgs[],
     %OutArgs[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Changing access for the specified resource failed>
}

// Step 4. Retrieve the references to the Privileges that represent the
// resulting rights between the subject and target instances.
$OutPrivileges->[] = %OutArgs[“Privileges”]
```

8.2.5.2.6.3 Deny a right

```
// DESCRIPTION
// This recipe describes how to remove a right from a given resource.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The name of a top-level System instance in the Security Profile has
// previously been discovered via SLP and is known as $System->.
// 2. The name of a managed element on $System-> has previously been
// discovered and is known as $Resource->.
// 3. The name of a subject has previously been discovered and is known as
// $Subject->.
// 4. A container of activities to be granted or denied is known as #Activity[].
// 5. A container of additional information related to the activities is known
// as #ActivityQualifiers[].
// 6. A container of sematic descriptions of the formats of the elements in
// #ActivityQualifiers[] is known as #QualifierFormats[].

// MAIN
// Step 1. Locate the PrivilegeManagementServices on the system.
$PrivilegeServices->[] = AssociatorNames($System->,
     “CIM_HostedService”,
     “CIM_PrivilegeManagementService”,
     “Antecedent”,
     “Dependent”)

// There must be exactly one PrivilegeManagementService for the managed element.
$PrivilegeService-> = &GetPrivilegeServiceForElement($PrivilegeServices->[],
     $Resource->)
if ($PrivilegeService-> == null) {
    <EXIT! The required PrivilegeManagementService was not found>
}

// Step 2. Create an Access Privilege
$Privilege = newInstance(“CIM_Privilege”)
$Privilege.PrivilegeGranted = false
$Privilege.RepresentsAuthorizationRights = false
$Privilege.Activity[] = #Activity[]
$Privilege.ActivityQualifiers[] = #ActivityQualifiers[]
$Privilege.QualifierFormats[] = #QualifierFormats[]

$Privilege[1] = $Privilege
```

```
// Step 3. Remove the right and get the resultant rights.
%InArgs[“Subject”] = $Subject->
%InArgs[“Target”] = $Resource->
%InArgs[“PropagationPolicies”] = null
$Privileges[0] = $Privilege
%InArgs[“Privileges”] = $Privileges[]
#Result = InvokeMethod($PrivilegeService->,
     “ChangeAccess”,
     %InArgs[],
     %OutArgs[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Changing access for the specified resource failed>
}

// Step 4. Retrieve the references to the Privileges that represent the
// resulting rights between the subject and target instances.
$OutPrivileges->[] = %OutArgs[“Privileges”]
```

8.2.5.2.7 Registered Name and Version

Security Authorization version 1.1.0

8.2.5.2.8 CIM Server Requirements

Table 414: CIM Server Requirements for Security Authorization

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	Yes
Schema Manipulation	No

8.2.5.2.9 CIM Elements

Table 415: CIM Elements for Security Authorization

Element Name	Description
Mandatory Classes	
CIM_ElementConformsToProfile (8.2.5.2.9.9)	
CIM_HostedService (8.2.5.2.9.10)	
CIM_PrivilegeManagementService (8.2.5.2.9.16)	
CIM_RegisteredSubProfile (8.2.5.2.9.19)	
CIM_ServiceAvailableToElement (8.2.5.2.9.21)	
CIM_SubProfileRequiresProfile (8.2.5.2.9.22)	
CIM_System (8.2.5.2.9.23)	
Optional Classes	
CIM_AuthorizationRule (8.2.5.2.9.1)	
CIM_AuthorizationRuleAppliesToIdentity (8.2.5.2.9.2)	
CIM_AuthorizationRuleAppliesToPrivilege (8.2.5.2.9.3)	
CIM_AuthorizationRuleAppliesToTarget (8.2.5.2.9.4)	
CIM_AuthorizedPrivilege (8.2.5.2.9.5)	
CIM_AuthorizedSubject (8.2.5.2.9.6)	
CIM_AuthorizedTarget (8.2.5.2.9.7)	
CIM_ConcreteDependency (8.2.5.2.9.8)	
CIM_Identity (8.2.5.2.9.11)	
CIM_ManagedElement (8.2.5.2.9.12)	
CIM_PolicyRuleInSystem (8.2.5.2.9.13)	
CIM_PolicySetAppliesToElement (8.2.5.2.9.14)	
CIM_Privilege (8.2.5.2.9.15)	
CIM_PrivilegePropagationRule (8.2.5.2.9.17)	
CIM_RegisteredProfile (8.2.5.2.9.18)	
CIM_ServiceAffectsElement (8.2.5.2.9.20)	

8.2.5.2.9.1 CIM_AuthorizationRule

Class Mandatory: false

Table 416: SMI Referenced Properties/Methods for CIM_AuthorizationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.2.9.2 CIM_AuthorizationRuleAppliesToIdentity

Class Mandatory: false

Table 417: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Identity	Key

8.2.5.2.9.3 CIM_AuthorizationRuleAppliesToPrivilege

Class Mandatory: false

Table 418: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToPrivilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Privilege	Key

8.2.5.2.9.4 CIM_AuthorizationRuleAppliesToTarget

Class Mandatory: false

Table 419: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.2.9.5 CIM_AuthorizedPrivilege

Class Mandatory: false

Table 420: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key

Table 420: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Property	Flags	Type	Description & Notes
RepresentsAuthorizationRights		boolean	Must be an Access right for this sub-profile.
PrivilegeGranted		boolean	Only Grant type privileges are allowed.

8.2.5.2.9.6 CIM_AuthorizedSubject

Class Mandatory: false

Table 421: SMI Referenced Properties/Methods for CIM_AuthorizedSubject

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	Key
PrivilegedElement		CIM_ManagedElement	Key

8.2.5.2.9.7 CIM_AuthorizedTarget

Class Mandatory: false

Table 422: SMI Referenced Properties/Methods for CIM_AuthorizedTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	Key
TargetElement		CIM_ManagedElement	Key

8.2.5.2.9.8 CIM_ConcreteDependency

Class Mandatory: false

Table 423: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Key
Dependent		CIM_ManagedElement	Key

8.2.5.2.9.9 CIM_ElementConformsToProfile

Class Mandatory: true

Table 424: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.2.9.10 CIM_HostedService

Class Mandatory: true

Table 425: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.2.9.11 CIM_Identity

Class Mandatory: false

Table 426: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	The Identified entity is authenticated or not

8.2.5.2.9.12 CIM_ManagedElement

Class Mandatory: false

No specified properties or methods.

8.2.5.2.9.13 CIM_PolicyRuleInSystem

Class Mandatory: false

Table 427: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_PolicyRule	Key

8.2.5.2.9.14 CIM_PolicySetAppliesToElement

Class Mandatory: false

Table 428: SMI Referenced Properties/Methods for CIM_PolicySetAppliesToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_PolicySet	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.2.9.15 CIM_Privilege

Class Mandatory: false

Table 429: SMI Referenced Properties/Methods for CIM_Privilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RepresentsAuthorizationRights		boolean	Indicates the privilege is to assign the named rights to subjects.
Optional Properties/Methods			
PrivilegeGranted		boolean	Only Grant type privileges are allowed.

8.2.5.2.9.16 CIM_PrivilegeManagementService

Class Mandatory: true

Table 430: SMI Referenced Properties/Methods for CIM_PrivilegeManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key
Optional Properties/Methods			
ChangeAccess()			
ShowAccess()			

8.2.5.2.9.17 CIM_PrivilegePropagationRule

Class Mandatory: false

Table 431: SMI Referenced Properties/Methods for CIM_PrivilegePropagationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.2.9.18 CIM_RegisteredProfile

Class Mandatory: false

Table 432: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key

Table 432: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	Parent subprofile

8.2.5.2.9.19 CIM_RegisteredSubProfile

Class Mandatory: true

Table 433: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.2.9.20 CIM_ServiceAffectsElement

Class Mandatory: false

Table 434: SMI Referenced Properties/Methods for CIM_ServiceAffectsElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AffectingElement		CIM_Service	Key
AffectedElement		CIM_ManagedElement	Key

8.2.5.2.9.21 CIM_ServiceAvailableToElement

Class Mandatory: true

Table 435: SMI Referenced Properties/Methods for CIM_ServiceAvailableToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UserOfService		CIM_ManagedElement	Key
ServiceProvided		CIM_Service	Key

8.2.5.2.9.22 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 436: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.2.9.23 CIM_System

Class Mandatory: true

Table 437: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.2.10 Related Standards

Table 438: Related Standards for Security Authorization

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.3 Security Resource Ownership Subprofile

8.2.5.3.1 Description

.

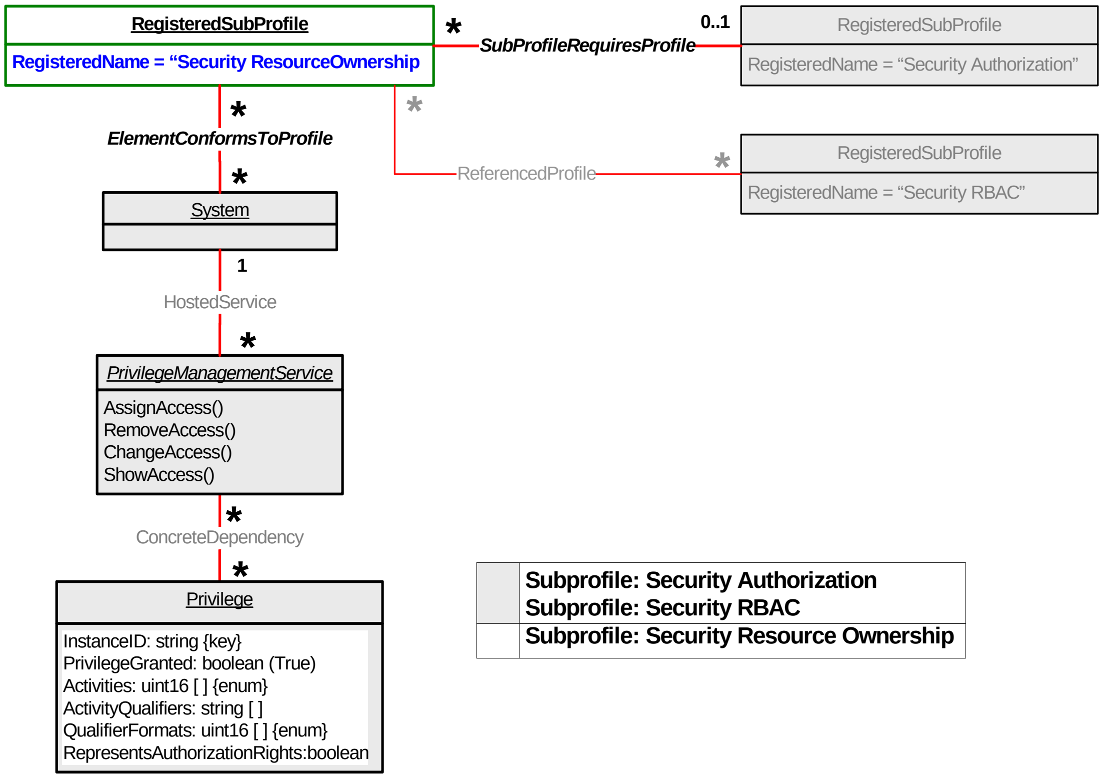

RegisteredSubProfile
RegisteredName = "Security ResourceOwnership
SubProfileRequiresProfile
0..1
RegisteredSubProfile
RegisteredName = "Security Authorization"
ElementConformsToProfile
ReferencedProfile
RegisteredSubProfile
RegisteredName = "Security RBAC"
System
1
HostedService
PrivilegeManagementService
AssignAccess()
RemoveAccess()
ChangeAccess()
ShowAccess()
ConcreteDependency
Privilege
InstanceID: string {key}
PrivilegeGranted: boolean (True)
Activities: uint16 [ ] {enum}
ActivityQualifiers: string [ ]
QualifierFormats: uint16 [ ] {enum}
RepresentsAuthorizationRights:boolean
Subprofile: Security Authorization
Subprofile: Security RBAC
Subprofile: Security Resource Ownership

Figure 74: Security Resource Ownership

This subprofile[2] provides the means to model restrictions on CIM operations associated with exclusive use of a resource, For instance, a storage volume in an array. It 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 its use. Specific examples include use of a volume by storage 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, although this is not strictly necessary. The Security Resource Ownership Subprofile is optional.

The model is permission-based (i.e., represents allowed operations, as opposed to forbidden ones). Where used, the policy is to deny all rights except those explicitly granted. Specific details of how the Security Resource Ownership Subprofile is applied are specified in the Resource Ownership Considerations subsection of the Cascading Considerations section of the including profile; this includes definition of the contents of the Privilege instances and definition of any propagation rules. The key class in Security Resource Ownership is the Privilege class that is used to grant rights to subjects (for instance, the identity of an embedded CIM client) to act on targets (resources that can be manipulated.)

2. The Security 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.

Support for the ShowAccess method is mandatory. It is used to extract which rights have been granted to a subject entity for a particular target resource. The implementation may also make this explicit by instantiating AuthorizedPrivilege instances with appropriate AuthorizedSubject and AuthorizedTarget associations.

An important aspect of this class is the RepresentsAuthorizationRights property:

- A Privilege with RepresentsAuthorizationRights = FALSE is an access privilege that controls invocation of CIM operations. The basic operation of an access privilege is that only the authorized subject identities can perform the Activities (including qualifiers) in the privilege on the authorized target(s).
- A Privilege with RepresentsAuthorizationRights = TRUE is a resource ownership privilege that controls the ability to associate access privileges with objects. The basic operation of an ownership privilege is to control the association of access privileges to target resource; for the Activities (including qualifiers) listed in the ownership privilege. Only authorized subjects of the ownership privilege are permitted to associate an access privilege containing any of those Activities with any target of the ownership privilege. An object that is an authorized target of an ownership privilege is called an owned resource.

An object can be subject to operation restrictions imposed by this subprofile only when it is an owned resource (i.e., the target of a resource ownership privilege). The algorithm is:

1) In the absence of an ownership privilege on a resource, any client may assign access privileges to that resource.
2) If an object is an owned resource (the target of a resource ownership privilege) then only subjects represented by owning Identities may assign access rights covered by the ownership Privilege instance to that resource.
3) In the absence of an access privilege on a resource, all clients are granted Read and Detect access (see the CIM Authorization model for information on the intrinsic operations covered by Read and Detect). All other access is denied.
4) All object reference parameters of each extrinsic method shall be checked; it is not sufficient to check only the first object reference parameter on the theory that the extrinsic is invoked on that object.
5) When Security Resource Ownership is in use, the CIM Client shall authenticate to the CIMOM to prevent misuse of Identity; an unauthenticated CIM Client will not be able to invoke any operation that is restricted by an access privilege.

For an object to be both owned and manageable via the controlling CIM Client, that object needs to be the target of a resource ownership privilege (for the ownership rights) and an access privilege (to allow management operations).

To enable future flexibility and (hopefully) minimize the opportunity for client programming errors, a resource supporting the Security Resource Ownership Subprofile shall either:

1) Instantiate one or more ownership Privilege instances containing allowable sets of rights to be granted. These are associated to the PrivilegeManagementService via ConcreteDependency associations. To assign ownership, the RepresentsAuthorizationRights property shall be set to TRUE in a copy of a Privilege instance passed in the ChangeAccess method. Otherwise, access rights are defined.
2) Instantiate one or more Role instances having ownership Privilege instances associated via Mem-berOfCollection. As above, these Privilege instances contain allowable sets of rights to be granted. Unless the Role applies to all resources in the System, the Role instances shall be associated to

applicable resources via RoleLimitedToTarget. The infrastructure may restrict the ability of the client to modify Role instances, including associations and associated Privileges. To assign ownership, a Role with Privileges, associated by MemberOfCollection, that have RepresentsAuthorizationRights set to TRUE, shall be associated via MemberOfCollection to one or more Identity instances. Each selected Identity instance shall be associated via ServiceAffectsElement to PrivilegeManagementService that is also associated to the resource via ServiceAffectsElement.

Privilege propagation rules, as defined by an instance of PrivilegePropagationRule, is a means of specifying how rights are propagated by a ChangeAccess call. The infrastructure may publish available propagation strategies via instances of PrivilegePropagationRule associated to a resource via PolicySetAppliesToElement associations. Alternatively, a Profile may define a set of “well-known” PrivilegePropagationRules that apply to particular types of resources and which may be discovered via enumeration. In either case, these available rules may be referenced in a ChangeAccess method.

Design Considerations

ServiceAffectsElement associations are assumed between Services and affected elements. (See Figure 75: "Service Associations".) This subprofile does not require an implementation to present these associations unless there is more than one PrivilegeManagementService in the profiled Namespace.

ServiceAvailableToElement associations are assumed between Services and using elements (See Figure 75: "Service Associations".) This subprofile does not require an implementation to present these associations unless there is more than one System in the profiled Namespace.

AuthorizedPrivilege instances are assumed when a Privilege is granted to a subject or assigned to a target. (See Figure 76: "AuthorizedPrivilege".) 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 75: Service Associations

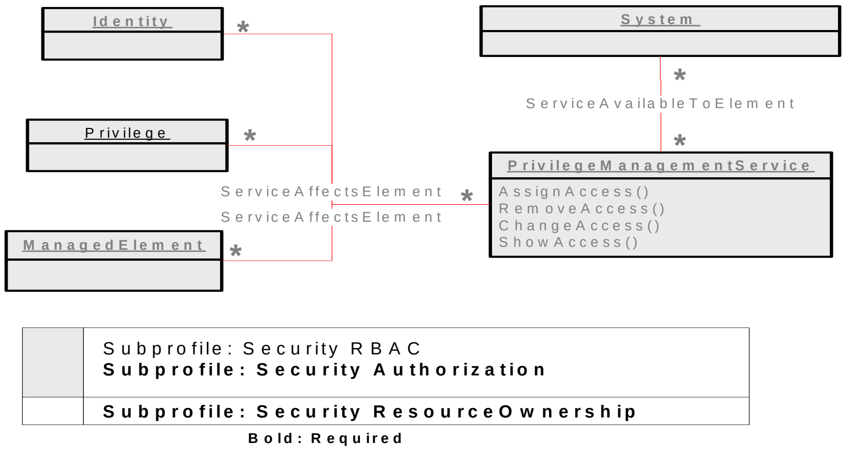
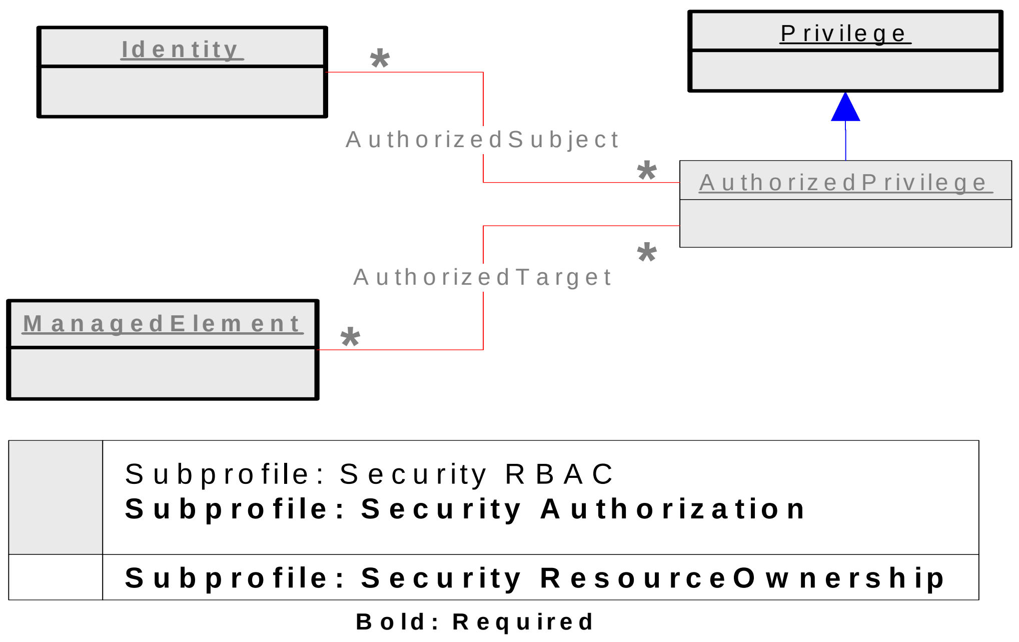

Identity
Privilege
ManagedElement
System
ServiceAvailableToElement
ServiceAffectsElement
ServiceAffectsElement
PrivilegeManagementService
AssignAccess()
RemoveAccess()
ChangeAccess()
ShowAccess()
*
Subprofile: Security RBAC
Subprofile: Security Authorization
Subprofile: Security ResourceOwnership
Bold: Required

Figure 76: AuthorizedPrivilege



8.2.5.3.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.5.3.3 Cascading Considerations

Not defined in this standard.

8.2.5.3.4 Supported Subprofiles and Packages

None.

8.2.5.3.5 Methods of the Profile

None.

8.2.5.3.6 Client Considerations and Recipes

8.2.5.3.6.1 Show Ownership Rights

```
// DESCRIPTION
// List the Subjects that have authorization rights to a resource.
// These subjects have ownership for the associated privileges.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// $Resource-> contains a reference to a resource (Any Managed Element)
// $PMS-> contains a reference to the PrivilegeManagementService
//
```

```
// Get Privileges for resource
//
#result = $PMS->ShowAccess(,$Resource->, $OutSubject->[], null, $OutPrivilege[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Show access for the specified resource failed>
}

// Filter out the non authorization rights
//
#k = 0
for #j in $OutPrivilege[] {
    if ($OutPrivilege[#j].RepresentsAuthorizationRights = True) {
     #k++
     $Subject->[#k] = $OutSubject->[#j]
     $Privilege->[#k] = $OutPrivilege->[#j]
    }
}



//
// $Resource-> contains resource
// $Subject->[] contains array of references to Identities (or other subjects),
//   with Authorization rights to a resource
// $Privilege[] contains array of Privileges, corresponding to the subject
entries.
//
```

8.2.5.3.6.2 Deny ownership rights

```
// DESCRIPTION
// Remove a set of authorization rights, (represented by a Privilege), from a named
//    Subject for a resource.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// The calling subject MUST be an owner for the named set of rights.
// Note: A resource is typically represented by an instance of some type of
//      CIM_ManagedElement.  Conceptually, a resource could also be an association
instance.
//       It is up to referencing Profiles to apply any additional constraints on
the types of
//       instances that are considered to be resource.
//
// $Identity-> contains a reference to a subject Identity
// $Resource-> contains a reference to a resource
// $Privilege contains a Privilege
// $PMS-> contains a reference to the PrivilegeManagementService
//
// This recipe is NOT dealing with Privilege Propagation.
//

// Set the Privilege to eliminate all rights
//
$Privilege[1] = $Privilege
$Privilege[1].PrivilegeGranted = False
$Privilege[1].RepresentsAuthorizationRights = True

// Eliminate all rights to the resource.
// Note that we don’t care whether someone else did it already.
//
#result = $PMS->ChangeAccess($Identity->,$Resource->,null,$Privilege[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Changing access for the specified resource failed>
}

// $Privilege[] contains the result array of Privileges between the subject and
target
//
```

8.2.5.3.6.3 Grant ownership rights

```
// DESCRIPTION
// Give a named Subject a set of authorization rights,
//    (represented by a Privilege) for a resource.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// The calling subject MUST be an owner.
//  This call also makes the named subject an owner.
//  The assumption is that the calling subject trusts the named subject.
//
// $Identity-> contains a reference to a subject Identity
// $Resource-> contains a reference to a resource
// $Privilege contains a Privilege to be granted
// $PMS-> contains a reference to the PrivilegeManagementService
//
// This recipe is NOT dealing with Privilege Propagation.
//

// Set the Privilege
//
$Privilege[1] = $Privilege
$Privilege[1].PrivilegeGranted = True
```

```
$Privilege[1].RepresentsAuthorizationRights = True

#result = $PMS->ChangeAccess($Identity->,$Resource->,null,$Privilege[])

// Verify that the operation performed successfully.
if (#Result != 0) {
    <EXIT! Changing access for the specified resource failed>
}

// $Privilege[] contains the result array of Privileges between the subject and
target
//
```

8.2.5.3.7 Registered Name and Version

Security Resource Ownership version 1.1.0

8.2.5.3.8 CIM Server Requirements

Table 439: CIM Server Requirements for Security Resource Ownership

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	Yes
Schema Manipulation	No

8.2.5.3.9 CIM Elements

Table 440: CIM Elements for Security Resource Ownership

Element Name	Description
Mandatory Classes	
CIM_ElementConformsToProfile (8.2.5.3.9.10)	
CIM_HostedService (8.2.5.3.9.11)	
CIM_PrivilegeManagementService (8.2.5.3.9.19)	
CIM_RegisteredSubProfile (8.2.5.3.9.23)	
CIM_RegisteredSubProfile (8.2.5.3.9.24)	
CIM_ServiceAvailableToElement (8.2.5.3.9.29)	
CIM_SubProfileRequiresProfile (8.2.5.3.9.30)	
CIM_System (8.2.5.3.9.31)	
Optional Classes	
CIM_AuthorizationRule (8.2.5.3.9.1)	
CIM_AuthorizationRuleAppliesToIdentity (8.2.5.3.9.2)	
CIM_AuthorizationRuleAppliesToPrivilege (8.2.5.3.9.3)	
CIM_AuthorizationRuleAppliesToRole (8.2.5.3.9.4)	
CIM_AuthorizationRuleAppliesToTarget (8.2.5.3.9.5)	
CIM_AuthorizedPrivilege (8.2.5.3.9.6)	
CIM_AuthorizedSubject (8.2.5.3.9.7)	
CIM_AuthorizedTarget (8.2.5.3.9.8)	
CIM_ConcreteDependency (8.2.5.3.9.9)	
CIM_Identity (8.2.5.3.9.12)	
CIM_ManagedElement (8.2.5.3.9.13)	
CIM_MemberOfCollection (8.2.5.3.9.14)	
CIM_OwningCollectionElement (8.2.5.3.9.15)	
CIM_PolicyRuleInSystem (8.2.5.3.9.16)	
CIM_PolicySetAppliesToElement (8.2.5.3.9.17)	

Table 440: CIM Elements for Security Resource Ownership

Element Name	Description
CIM_Privilege (8.2.5.3.9.18)	
CIM_PrivilegePropagationRule (8.2.5.3.9.20)	
CIM_ReferencedProfile (8.2.5.3.9.21)	
CIM_RegisteredProfile (8.2.5.3.9.22)	
CIM_RegisteredSubProfile (8.2.5.3.9.25)	
CIM_Role (8.2.5.3.9.26)	
CIM_RoleLimitedToTarget (8.2.5.3.9.27)	
CIM_ServiceAffectsElement (8.2.5.3.9.28)	

8.2.5.3.9.1 CIM_AuthorizationRule

Class Mandatory: false

Table 441: SMI Referenced Properties/Methods for CIM_AuthorizationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.3.9.2 CIM_AuthorizationRuleAppliesToIdentity

Class Mandatory: false

Table 442: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Identity	Key

8.2.5.3.9.3 CIM_AuthorizationRuleAppliesToPrivilege

Class Mandatory: false

Table 443: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToPrivilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Privilege	Key

8.2.5.3.9.4 CIM_AuthorizationRuleAppliesToRole

Class Mandatory: false

Table 444: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToRole

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Role	Key

8.2.5.3.9.5 CIM_AuthorizationRuleAppliesToTarget

Class Mandatory: false

Table 445: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.3.9.6 CIM_AuthorizedPrivilege

Class Mandatory: false

Table 446: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RepresentsAuthorizationRights		boolean	Must be an Access right for this sub-profile.
PrivilegeGranted		boolean	Only Grant type privileges are allowed.

8.2.5.3.9.7 CIM_AuthorizedSubject

Class Mandatory: false

Table 447: SMI Referenced Properties/Methods for CIM_AuthorizedSubject

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	Key
PrivilegedElement		CIM_ManagedElement	Key

8.2.5.3.9.8 CIM_AuthorizedTarget

Class Mandatory: false

Table 448: SMI Referenced Properties/Methods for CIM_AuthorizedTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	Key
TargetElement		CIM_ManagedElement	Key

8.2.5.3.9.9 CIM_ConcreteDependency

Class Mandatory: false

Table 449: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Key
Dependent		CIM_ManagedElement	Key

8.2.5.3.9.10 CIM_ElementConformsToProfile

Class Mandatory: true

Table 450: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.3.9.11 CIM_HostedService

Class Mandatory: true

Table 451: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.3.9.12 CIM_Identity

Class Mandatory: false

Table 452: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	

8.2.5.3.9.13 CIM_ManagedElement

Class Mandatory: false
No specified properties or methods.

8.2.5.3.9.14 CIM_MemberOfCollection

Class Mandatory: false

Table 453: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	Key
Member		CIM_ManagedElement	Key

8.2.5.3.9.15 CIM_OwningCollectionElement

Class Mandatory: false

Table 454: SMI Referenced Properties/Methods for CIM_OwningCollectionElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
OwnedElement		CIM_Collection	Key
OwningElement		CIM_ManagedElement	Key

8.2.5.3.9.16 CIM_PolicyRuleInSystem

Class Mandatory: false

Table 455: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_PolicyRule	Key

8.2.5.3.9.17 CIM_PolicySetAppliesToElement

Class Mandatory: false

Table 456: SMI Referenced Properties/Methods for CIM_PolicySetAppliesToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_PolicySet	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.3.9.18 CIM_Privilege

Class Mandatory: false

Table 457: SMI Referenced Properties/Methods for CIM_Privilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RepresentsAuthorizationRights		boolean	Must be an Access right for this sub-profile.
PrivilegeGranted		boolean	Only Grant type privileges are allowed.

8.2.5.3.9.19 CIM_PrivilegeManagementService

Class Mandatory: true

Table 458: SMI Referenced Properties/Methods for CIM_PrivilegeManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key
Optional Properties/Methods			
ChangeAccess()			

8.2.5.3.9.20 CIM_PrivilegePropagationRule

Class Mandatory: false

Table 459: SMI Referenced Properties/Methods for CIM_PrivilegePropagationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.3.9.21 CIM_ReferencedProfile

Class Mandatory: false

Table 460: SMI Referenced Properties/Methods for CIM_ReferencedProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.3.9.22 CIM_RegisteredProfile

Class Mandatory: false

Table 461: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization		uint16	
RegisteredName		string	

8.2.5.3.9.23 CIM_RegisteredSubProfile

Class Mandatory: true

Table 462: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.3.9.24 CIM_RegisteredSubProfile

Class Mandatory: true

Table 463: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization		uint16	
RegisteredName		string	

8.2.5.3.9.25 CIM_RegisteredSubProfile

Class Mandatory: false

Table 464: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization		uint16	
RegisteredName		string	

8.2.5.3.9.26 CIM_Role

Class Mandatory: false

Table 465: SMI Referenced Properties/Methods for CIM_Role

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.3.9.27 CIM_RoleLimitedToTarget

Class Mandatory: false

Table 466: SMI Referenced Properties/Methods for CIM_RoleLimitedToTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
DefiningRole		CIM_Role	Key
TargetElement		CIM_ManagedElement	Key

8.2.5.3.9.28 CIM_ServiceAffectsElement

Class Mandatory: false

Table 467: SMI Referenced Properties/Methods for CIM_ServiceAffectsElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AffectingElement		CIM_Service	Key
AffectedElement		CIM_ManagedElement	Key

8.2.5.3.9.29 CIM_ServiceAvailableToElement

Class Mandatory: true

Table 468: SMI Referenced Properties/Methods for CIM_ServiceAvailableToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UserOfService		CIM_ManagedElement	Key
ServiceProvided		CIM_Service	Key

8.2.5.3.9.30 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 469: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.3.9.31 CIM_System

Class Mandatory: true

Table 470: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.3.10 Related Standards

Table 471: Related Standards for Security Resource Ownership

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.4 Security Role Based Access Control Subprofile

8.2.5.4.1 Description

8.2.5.4.1.1 Overview

The Role Based Access Control (RBAC.) subprofile enables management of authorization using RBAC Roles, (see Figure 77: "Role-Based Access Control"). The Security RBAC subprofile is a subprofile of the Security Authorization subprofile.

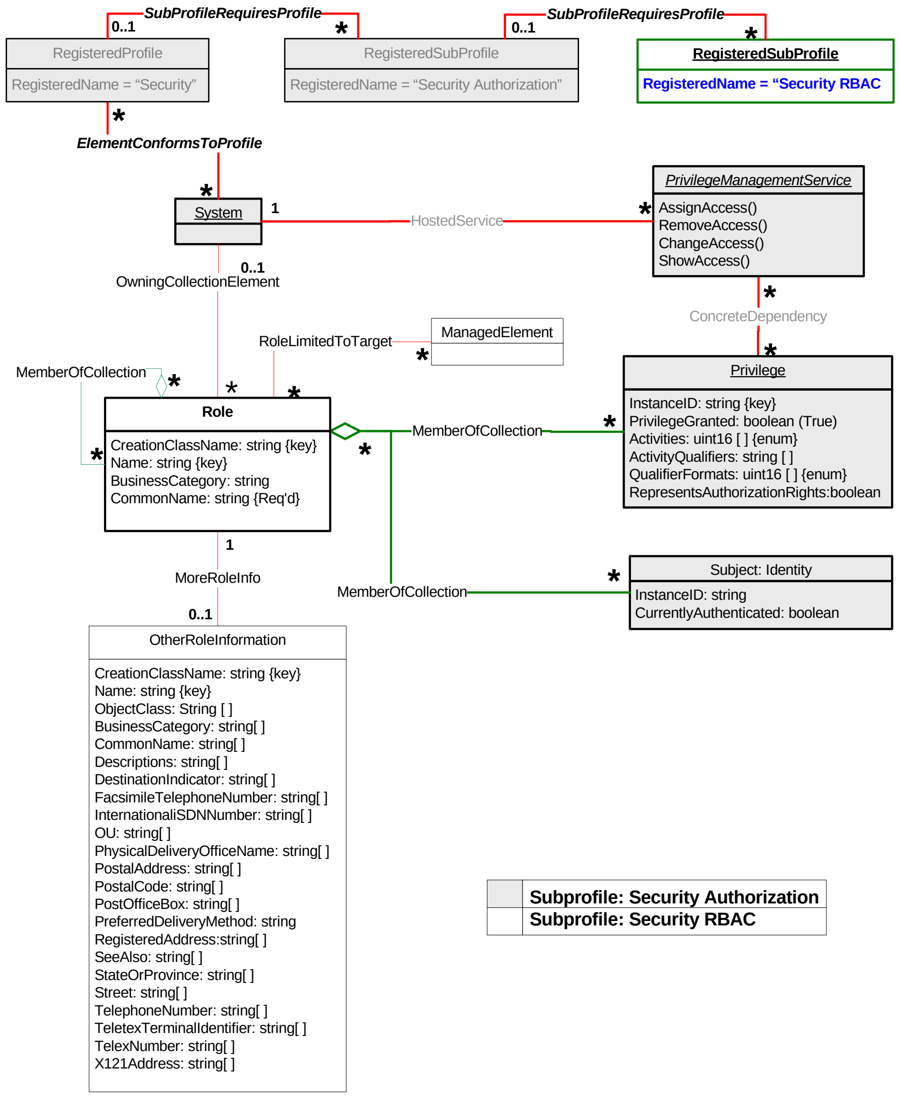
If this subprofile is supported, the CIM Server may publish some number of Roles via OwningCollectionElement associations to the top level System. Rights are granted to a Role by Privilege instances associated via MemberOfCollection. Target resources are associated to a Role via RoleLimitedToTarget associations.

If a subject Identity is associated to a Role via MemberOfCollection and if CurrentlyAuthenticated is true, then the entity named by the Identity is authorized to exercise all rights granted by the Role to target resources.

If there are no RoleLimitedToTarget associations, then the Role applies to all resources in the System. If there are RoleLimitedToTarget association, then those associations identify the target resources of the role.

A Role may collect other Roles via MemberOfCollection. Privileges of the included Role are granted to Identities of the including Role for those resources that are scoped to both Roles.

Figure 77: Role-Based Access Control



SubProfileRequiresProfile
0..1
*
RegisteredProfile
RegisteredName = "Security"
SubProfileRequiresProfile
0..1
*
RegisteredSubProfile
RegisteredName = "Security Authorization"
RegisteredSubProfile
RegisteredName = "Security RBAC
*
ElementConformsToProfile
*
System
1
HostedService
*
PrivilegeManagementService
AssignAccess()
RemoveAccess()
ChangeAccess()
ShowAccess()
0..1
OwningCollectionElement
*
ConcreteDependency
*
ManagedElement
RoleLimitedToTarget
*
MemberOfCollection
*
*
*
Role
CreationClassName: string {key}
Name: string {key}
BusinessCategory: string
CommonName: string {Req'd}
*
MemberOfCollection
*
*
Privilege
InstanceID: string {key}
PrivilegeGranted: boolean (True)
Activities: uint16 [ ] {enum}
ActivityQualifiers: string [ ]
QualifierFormats: uint16 [ ] {enum}
RepresentsAuthorizationRights:boolean
1
MoreRoleInfo
MemberOfCollection
*
Subject: Identity
InstanceID: string
CurrentlyAuthenticated: boolean
0..1
OtherRoleInformation
CreationClassName: string {key}
Name: string {key}
ObjectClass: String [ ]
BusinessCategory: string[ ]
CommonName: string[ ]
Descriptions: string[ ]
DestinationIndicator: string[ ]
FacsimileTelephoneNumber: string[ ]
InternationaliSDNNumber: string[ ]
OU: string[ ]
PhysicalDeliveryOfficeName: string[ ]
PostalAddress: string[ ]
PostalCode: string[ ]
PostOfficeBox: string[ ]
PreferredDeliveryMethod: string
RegisteredAddress:string[ ]
SeeAlso: string[ ]
StateOrProvince: string[ ]
Street: string[ ]
TelephoneNumber: string[ ]
TeletexTerminalIdentifier: string[ ]
TelexNumber: string[ ]
X121Address: string[ ]

▨	**Subprofile: Security Authorization**
☐	**Subprofile: Security RBAC**

8.2.5.4.1.2 Default Authorization

The ChangeAccess method is not used to grant or deny authorization via Roles. Rather, this subprofile uses CIM Intrinsic methods CreateInstance and DeleteInstance on appropriate associations and on the Role class itself. The following list describes the rules.

- All resources of a system conforming to this subprofile are scoped to any Role with no RoleLimitedToTarget associations.
- Only resources associated by RoleLimitedToTarget are scoped to a Role with RoleLimitedToTarget associations. CreateInstance and DeleteInstance are used to add or delete RoleLimitedToTarget associations.
- MemberOfCollection associations are used to grant Privileges to a Role. CreateInstance and DeleteInstance are used to add or delete MemberOfCollection associations between Privilege and Role instances.
- MemberOfCollection associations are used to place Identities into a Role. CreateInstance and DeleteInstance are used to add or delete MemberOfCollection associations between Identity and Role instances.
- Every Identity in a Role is authorized with all rights defined by all Privileges granted to the Role for all resources scoped to the Role. The set of authorized rights is adjusted dynamically as a result of CreateInstance and DeleteInstance operations on the MemberOfCollection and RoleLimitedToTarget associations described above.
- MemberOfCollection associations are used to incorporate one Role into another Role. CreateInstance and DeleteInstance are used to add or delete MemberOfCollection associations between Role instances. The following additional rules apply:
    - Identities of the incorporating Role are authorized with all rights defined by all Privileges granted to the incorporated Role for all resources that are scoped to the intersection of the set of resources scoped to each Role.
    - This process is recursive through the MemberOfCollection association between Roles with the added conditions that:
        - At each level, the intersecting set of resources found at level *n* is intersected with the set of resources scoped to level *n+1*.
        - This intersection forms the set of resources to which the Identities of level *1* are authorized with the Privileges of level *n+1*.

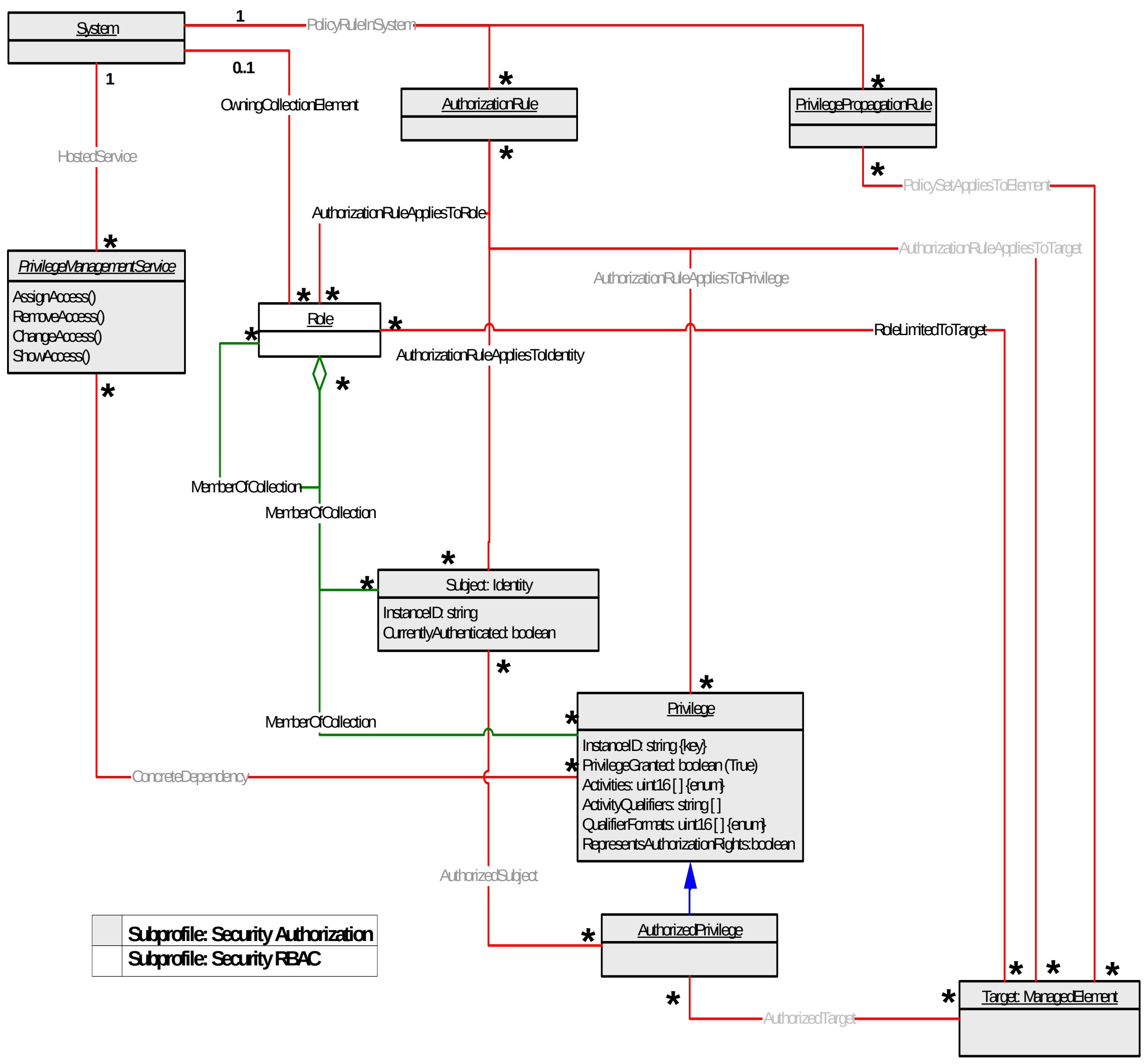
8.2.5.4.1.3 Authorization Policy

This subprofile extends the Authorization Policy defined in the Security Authorization subprofile.

In addition associations specified in the Security Authorization subprofile. AuthenticationRuleAppliesToRole may be used to incorporate a Role into an AuthenticationRule. This is shown in Figure 78: "Policy Rules".

The details of the specification of AuthorizationRules are left to the profiles and subprofiles that reference this subprofile.

Figure 78: Policy Rules


System
PolicyRuleInSystem
OwningCollectionElement
HostedService
AuthorizationRule
PrivilegePropagationRule
PolicySetAppliesToElement
AuthorizationRuleAppliesToRole
AuthorizationRuleAppliesToTarget
AuthorizationRuleAppliesToPrivilege
PrivilegeManagementService
AssignAccess()
RemoveAccess()
ChangeAccess()
ShowAccess()
Role
RoleLimitedToTarget
AuthorizationRuleAppliesToIdentity
MemberOfCollection
Subject: Identity
InstanceID: string
CurrentlyAuthenticated: boolean
Privilege
InstanceID: string {key}
PrivilegeGranted: boolean (True)
Activities: uint16 [ ] {enum}
ActivityQualifiers: string [ ]
QualifierFormats: uint16 [ ] {enum}
RepresentsAuthorizationRights: boolean
ConcreteDependency
AuthorizedSubject
AuthorizedPrivilege
AuthorizedTarget
Target: ManagedElement
Subprofile: Security Authorization
Subprofile: Security RBAC

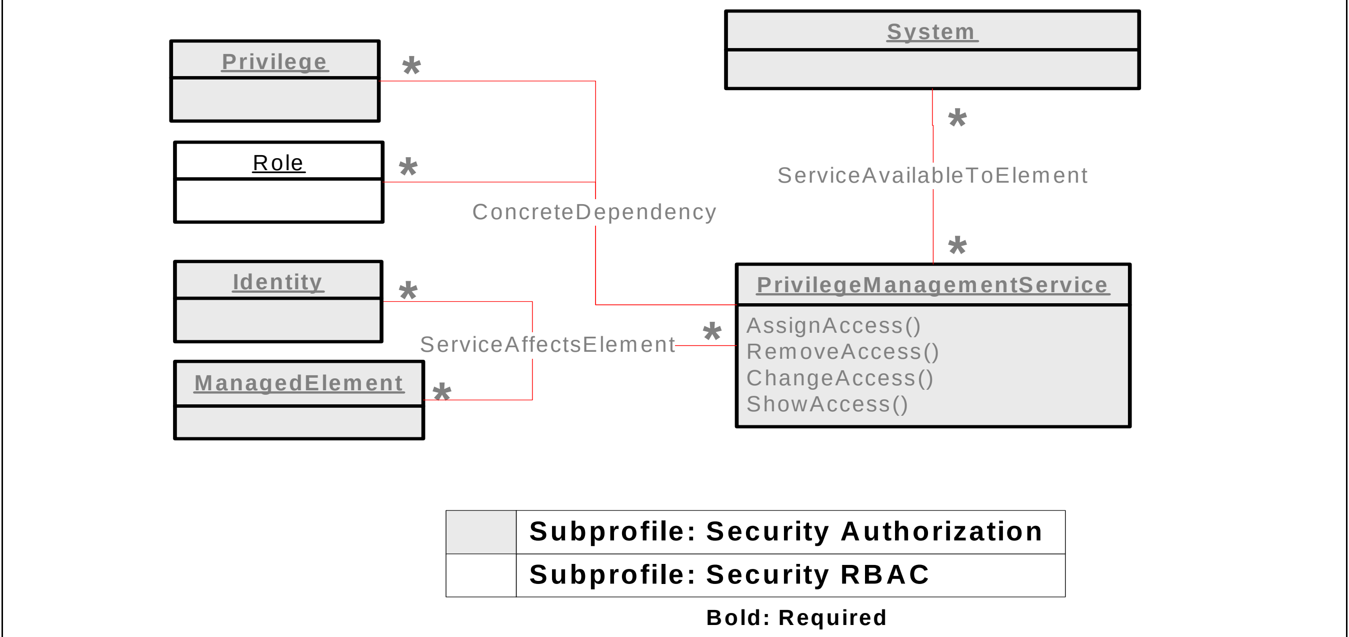
8.2.5.4.1.4 Design Considerations

ConcreteDependency associations are assumed between Services and the elements that they directly manage (See Figure 79: "Service Associations".) This subprofile does not REQUIRE an implementation to present these associations unless there is more than one PrivilegeManagementService in the profiled Namespace.

ServiceAffectsElement associations are assumed between Services and affected elements. (See Figure 79: "Service Associations".) This subprofile does not REQUIRE an implementation to present these associations unless there is more than one PrivilegeManagementService in the profiled Namespace.

ServiceAvailableToElement associations are assumed between Services and using elements (See Figure 79: "Service Associations".) This subprofile does not REQUIRE an implementation to present these associations unless there is more than one System in the profiled Namespace.

Figure 79: Service Associations



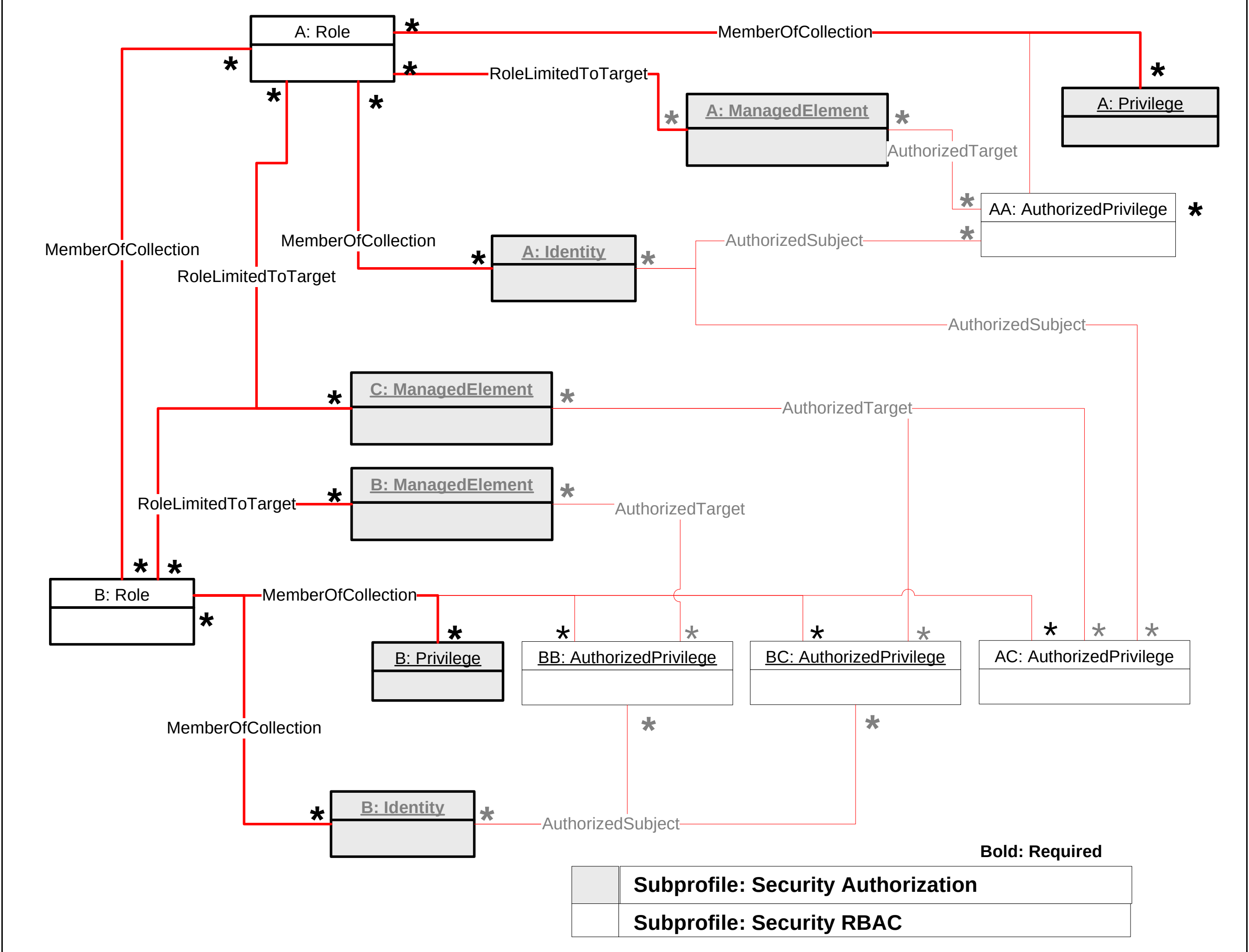
This subprofile does not REQUIRE the implementation to make AuthorizedPrivilege instances explicit. However, there existence is assumed whenever a Role containing one or more Privileges is associated by MemberOfCollection to an Identity.

- In the case where there is no RoleLimitedToTarget association, then all ManagedElements are implicitly authorized to the collected Identity instances.
- If RoleLimitedToTarget associations are used, then only those ManagedElements are authorized. Figure 80: "AuthorizedPrivilege" show this case.
- Additionally Figure 80: "AuthorizedPrivilege" shows the case where a Role is collected into another role. Only the intersection of target resources between the included and including Roles are granted permission for Identities of the including Role. For example, in Figure 80: "AuthorizedPrivilege", note that Identity B does not become authorized to ManagedElement A.

However, Identity A does become authorized to ManagedElement AB.This subprofile relies on the ShowAccess method to display rights the rights granted by membership in a Role

.

Figure 80: AuthorizedPrivilege



8.2.5.4.2 Health and Fault Management Consideration

Not defined in this standard.

8.2.5.4.3 Cascading Considerations

Not defined in this standard.

8.2.5.4.4 Supported Subprofiles and Packages

None.

8.2.5.4.5 Methods of the Profile

None.

8.2.5.4.6 Client Considerations and Recipes

8.2.5.4.6.1 List the Roles associated with an Identity

```
// DESCRIPTION
// For a specific Identity, this recipe lists all associated Roles
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// $Identity-> contains a reference to an Identity
//

//==================================================================
// Subroutines of SecurityRBAC 1
//==================================================================
Sub GetMemberRoles($StartRoles->[], $Roles->[])
{
    // Get Member Roles
    for #i in $StartRoles->[]
    {
        $MemberRoles->[] = AssociatorNames($StartRoles->[#i],

“CIM_MemberOfCollection”,”CIM_Role”,Collection,)

        // Append Member Roles to Roles output.
     // Note that on the first iteration size of Roles is 0.
     //       On the next interation Roles.size is now size of previous
MemberRoles.
        //
        #i = $Roles->[].size
        for #j in $MemberRoles->[]
        {
            $Roles->[(#i+#j] = $MemberRoles->[#j]
        }
     //  Get Members of Members
     //
        &GetMemberRoles($MemberRoles->[], Roles->[])
}


//==================================================================
//SecurityRBAC 1 Recipe starts here
//==================================================================

// Find the first-level Roles of an Identity.
//
$Roles->[] = AssociatorNames($Identity>,
“CIM_MemberOfCollection”,”CIM_Role”,Member,)

//Append Member Roles
```

```
&GetMemberRoles($Roles->[], $Roles->[])

// ON OUTPUT
//
// $Roles->[] contains a list of pointers to Roles
//
```

8.2.5.4.6.2 List the Privileges of a Role

```
// DESCRIPTION
// For a specific Role, this recipe lists all associated Privileges obtained
// via membership in various Roles.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// $Role-> contains a reference to a Role
//

//===============================================================
// Subroutines of SecurityRBAC 2
//===============================================================
Sub GetMemberPrivileges($StartRoles->[], $Roles->[], $Privileges->[])
{
    // Get Member Roles
    for #i in $StartRoles->[]
    {
        $MemberRoles->[] = AssociatorNames($StartRoles->[#i],

“CIM_MemberOfCollection”,”CIM_Role”,Collection,)

        // Append Member Roles to Roles output.
     // Note that on the first iteration size of Roles is 0.
     //      On the next interation Roles.size is now size of previous
     //  MemberRoles.
        //
        #i = $Roles->[].size
        for #j in $MemberRoles->[]
        {
            $Roles->[#i+#j] = $MemberRoles->[#j]

            // Now append the Privileges for each member
         //
            $MemberPrivs->[] = AssociatorNames(&MemberRoles-[#j],
        “CIM_MemberOfCollection”,”CIM_Privilege”,Collection,)
        #k = $Privileges->[].size
            for #l in $MemberPrivs->[]
            {
                $Privileges->[#k+#l] = $MemberPrivs->[#l]
            }
        }
     //  Get Members of Members
     //
        &GetMemberRoles($MemberRoles->[], Roles->[], $Privileges->[])
}


//===============================================================
//SecurityRBAC 2 Recipe starts here
//===============================================================

// Find the first-level Privileges
//
    $Privileges->[] = AssociatorNames($Role->,

“CIM_MemberOfCollection”,”CIM_Privilege”,Collection,)

//Append Member Privileges
$Roles->[1] = $Role->&GetMemberPrivileges($Roles->[], $Roles->[], $Privileges->[])

// ON OUTPUT
//
// $Roles->[] contains a list of pointers to Roles in the Role hierarchy
// $Privileges->[] contains a list of pointers to Privileges from the Role
hierachy
//
```

8.2.5.4.7 Registered Name and Version

Security RBAC version 1.1.0

8.2.5.4.8 CIM Server Requirements

Table 472: CIM Server Requirements for Security RBAC

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	Yes
Schema Manipulation	No

8.2.5.4.9 CIM Elements

Table 473: CIM Elements for Security RBAC

Element Name	Description
Mandatory Classes	
CIM_ConcreteDependency (8.2.5.4.9.3)	
CIM_ElementConformsToProfile (8.2.5.4.9.4)	
CIM_HostedService (8.2.5.4.9.5)	
CIM_RegisteredSubProfile (8.2.5.4.9.16)	
CIM_RegisteredSubProfile (8.2.5.4.9.17)	
CIM_SubProfileRequiresProfile (8.2.5.4.9.20)	
CIM_System (8.2.5.4.9.21)	
Optional Classes	
CIM_AuthorizationRule (8.2.5.4.9.1)	
CIM_AuthorizationRuleAppliesToRole (8.2.5.4.9.2)	
CIM_Identity (8.2.5.4.9.6)	
CIM_ManagedElement (8.2.5.4.9.7)	
CIM_MemberOfCollection (8.2.5.4.9.8)	
CIM_MoreRoleInfo (8.2.5.4.9.9)	
CIM_OtherRoleInformation (8.2.5.4.9.10)	
CIM_OwningCollectionElement (8.2.5.4.9.11)	
CIM_PolicyRuleInSystem (8.2.5.4.9.12)	
CIM_Privilege (8.2.5.4.9.13)	
CIM_PrivilegeManagementService (8.2.5.4.9.14)	
CIM_RegisteredProfile (8.2.5.4.9.15)	
CIM_Role (8.2.5.4.9.18)	
CIM_RoleLimitedToTarget (8.2.5.4.9.19)	

8.2.5.4.9.1 CIM_AuthorizationRule

Class Mandatory: false

Table 474: SMI Referenced Properties/Methods for CIM_AuthorizationRule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
PolicyRuleName		string	Key

8.2.5.4.9.2 CIM_AuthorizationRuleAppliesToRole

Class Mandatory: false

Table 475: SMI Referenced Properties/Methods for CIM_AuthorizationRuleAppliesToRole

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PolicySet		CIM_AuthorizationRule	Key
ManagedElement		CIM_Role	Key

8.2.5.4.9.3 CIM_ConcreteDependency

Class Mandatory: true

Table 476: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Key
Dependent		CIM_ManagedElement	Key

8.2.5.4.9.4 CIM_ElementConformsToProfile

Class Mandatory: true

Table 477: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.4.9.5 CIM_HostedService

Class Mandatory: true

Table 478: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.4.9.6 CIM_Identity

Class Mandatory: false

Table 479: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	Entity is authenticated to use this Identity.

8.2.5.4.9.7 CIM_ManagedElement

Class Mandatory: false
No specified properties or methods.

8.2.5.4.9.8 CIM_MemberOfCollection

Class Mandatory: false

Table 480: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	Key
Member		CIM_ManagedElement	Key

8.2.5.4.9.9 CIM_MoreRoleInfo

Class Mandatory: false

Table 481: SMI Referenced Properties/Methods for CIM_MoreRoleInfo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Role	
Dependent		CIM_OtherRoleInformation	

8.2.5.4.9.10 CIM_OtherRoleInformation

Class Mandatory: false

Table 482: SMI Referenced Properties/Methods for CIM_OtherRoleInformation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key, Must match that of Role

8.2.5.4.9.11 CIM_OwningCollectionElement

Class Mandatory: false

Table 483: SMI Referenced Properties/Methods for CIM_OwningCollectionElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
OwningElement		CIM_ManagedElement	Key
OwnedElement		CIM_Collection	Key

8.2.5.4.9.12 CIM_PolicyRuleInSystem

Class Mandatory: false

Table 484: SMI Referenced Properties/Methods for CIM_PolicyRuleInSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_PolicyRule	Key

8.2.5.4.9.13 CIM_Privilege

Class Mandatory: false

Table 485: SMI Referenced Properties/Methods for CIM_Privilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RepresentsAuthorizationRights		boolean	Rights are to assign rights.
PrivilegeGranted		boolean	Instantiated Privileges will only be granted.

8.2.5.4.9.14 CIM_PrivilegeManagementService

Class Mandatory: false

Table 486: SMI Referenced Properties/Methods for CIM_PrivilegeManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.4.9.15 CIM_RegisteredProfile

Class Mandatory: false

Table 487: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	Parent subprofile

8.2.5.4.9.16 CIM_RegisteredSubProfile

Class Mandatory: true

Table 488: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.4.9.17 CIM_RegisteredSubProfile

Class Mandatory: true

Table 489: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization		uint16	
RegisteredName		string	

8.2.5.4.9.18 CIM_Role

Class Mandatory: false

Table 490: SMI Referenced Properties/Methods for CIM_Role

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.4.9.19 CIM_RoleLimitedToTarget

Class Mandatory: false

Table 491: SMI Referenced Properties/Methods for CIM_RoleLimitedToTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
DefiningRole		CIM_Role	Key
TargetElement		CIM_ManagedElement	Key

8.2.5.4.9.20 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 492: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.4.9.21 CIM_System

Class Mandatory: true

Table 493: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.4.10 Related Standards

Table 494: Related Standards for Security RBAC

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.5 IdentityManagement Subprofile

8.2.5.5.1 Description

This subprofile of the Security profile provides support for adding and managing users of a system and for mapping those users to accounts, people and organizations.

Users are assumed to have Identity instances to represent their ability to be authenticated. Identity instances may stand alone or may be linked to Accounts, Organizations, OrgUnits, UserContacts, Persons, Groups or Roles.

All Identity instances shall be unique within the namespace of the conformant System.

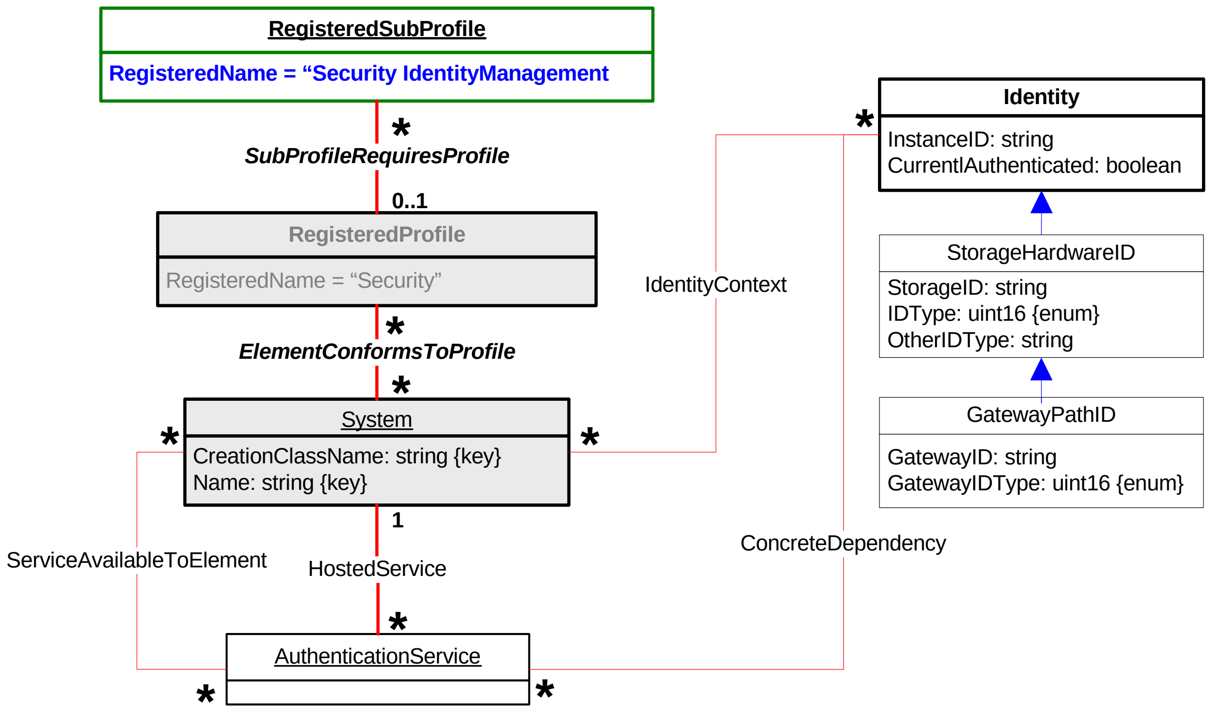
8.2.5.5.2 Identities

Identities represent a user of a system and when authenticated, represent a security principal.'

Authentication is performed by an authentication service which may be represented as an AuthenticationService. If represented, this specification relies on the implementation to instantiate appropriate ServiceAffectsElement associations between the AuthenticationService and an Identities.

If there are multiple Systems in the namespace, and the Identity is scoped to a particular System, then IdentityContext associations shall be instantiated between the Identity and the scoping System. CreateInstance and DeleteInstance may be used to instantiate IdentityContext associations. IdentityContext instances shall be deleted by the infrastructure as a side-affect of deleting an Identity.

Figure 81: Identities



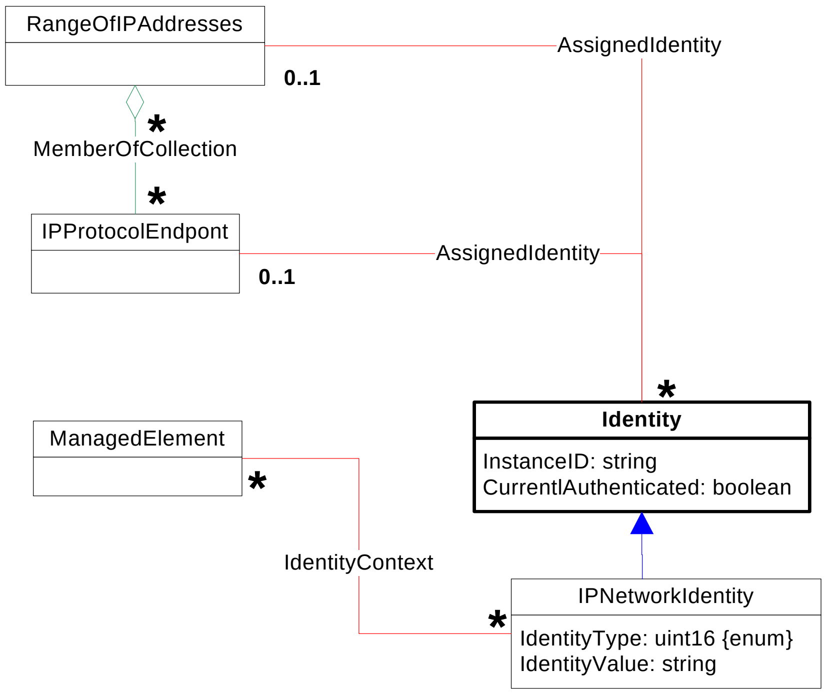
8.2.5.5.2.1 Stand-alone Identities

Two types of stand-alone Identities may be instantiated, StorageHardwareID and GatewayPathID. Use CreateInstance and DeleteInstance to instantiate stand-alone Identities. The detailed specification for use of StorageHardwareID and GatewayPathID instances is deferred to profiles or subprofiles that reference this subprofile.

8.2.5.5.2.2 Network Identities

NetworkIdentities represent a particular IPProtocolEndpoint or a collection of IPProtocolEndpoints.

Figure 82: IPNetworkIdentity



8.2.5.5.3 Accounts

Accounts are used for the purpose of authenticating Identities and may additionally be used to as a basis for tracking other information about the use of a system by a particular Identity. Account is essentially another aspect of Identity and is associated via ConcreteIdentity.

When creation of Accounts is supported, the implementation shall present an AccountManagementService instance together with HostedService and ServiceAvailableToElement associations.

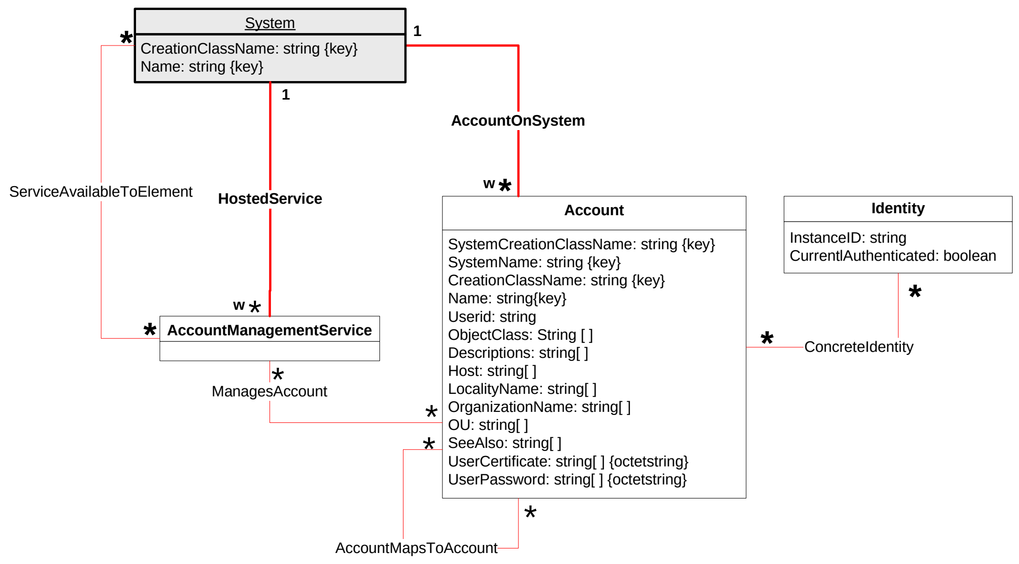
If an AccountManagementService is present, instances of Account may be added or deleted using the CreateInstance and DeleteInstance intrinsic methods. The key properties: SystemCreationClassName, SystemName, CreationClassName, and Name of each Account shall be fully specified at creation time. The implementation shall add or delete the AccountOnSystem associations automatically.

Modeling one or more AccountManagementService instances is optional for this subprofile. If there is only one AccountManagementService with a ServiceAvailableToElement association to the named System, then a ManagesAccount association may be implied or the implementation may automatically instantiate one. However if there is more than one AccountManagementService with a ServiceAvailableToElement association to the named System, an instance of ManagesAccount shall be added by a CreateInstance of an Account. The choice of which AccountManagementService to associate to is made intrinsically by the implementation. ManagesAccount instances are deleted automatically when an Account is deleted.

For each Account instance, this subprofile recommends a corresponding Identity instance, associated by ConcreteIdentity. When the Account is created, UserID is set and the UserPassword is specified in clear-text. The creation request is expected to be performed over a secure channel. This subprofile REQUIRES that the UserPassword property shall be write only.

UserContact and Person instances that are associated to an Account via a common Identity instance may have the same, non-null UserID. Setting UserID, UserCertificate or UserPassword properties on such related Account, UserContact or Person instances shall also set the corresponding entries in matching instances.

Figure 83: Account Management



8.2.5.5.4 Organizational Directories

There are three basic types of OrganizationalEntities that may be stored in a namespace:

- Organization instances describe top-level entities, like organizations. (See 8.2.5.5.4.1, "Organizations".)
- OrgUnit instances describe sub-units of organizations. (See 8.2.5.5.4.1, "Organizations".)
- UserEntity instances describe people. (See 8.2.5.5.4.2, "OPeople".)

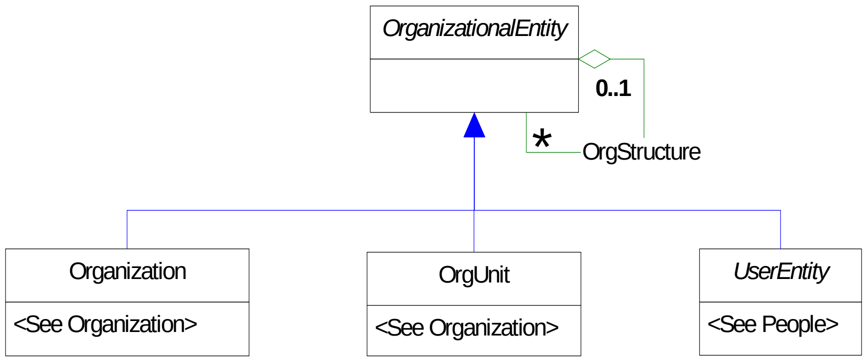
Any OrganizationalEntity may aggregate any number of Collections, such as Groups or Roles. This is managed by CreateInstance or DeleteInstance of CollectionInOrganization associations. This association may be used to associate a Collection to at most one OrganizationalEntity.

Any System may aggregate any number of Collections. This is managed by CreateInstance or DeleteInstance of OwningCollectionElement associations. This association may be used to associate a Collection to at most one System.

A single Collection shall not have both OwningCollectionElement and CollectionInOrganization associations.

Any OrganizationalEntity may aggregate any number of other OrganizationalUnits. For example, a Company may have Business Units and Business Units may have Departments. This is managed by CreateInstance or DeleteInstance of OrgStructure associations. An OrganizationalUnit may belong to at most one OrganizationalUnit.

Figure 84: OrganizationalEntities



8.2.5.5.4.1 Organizations

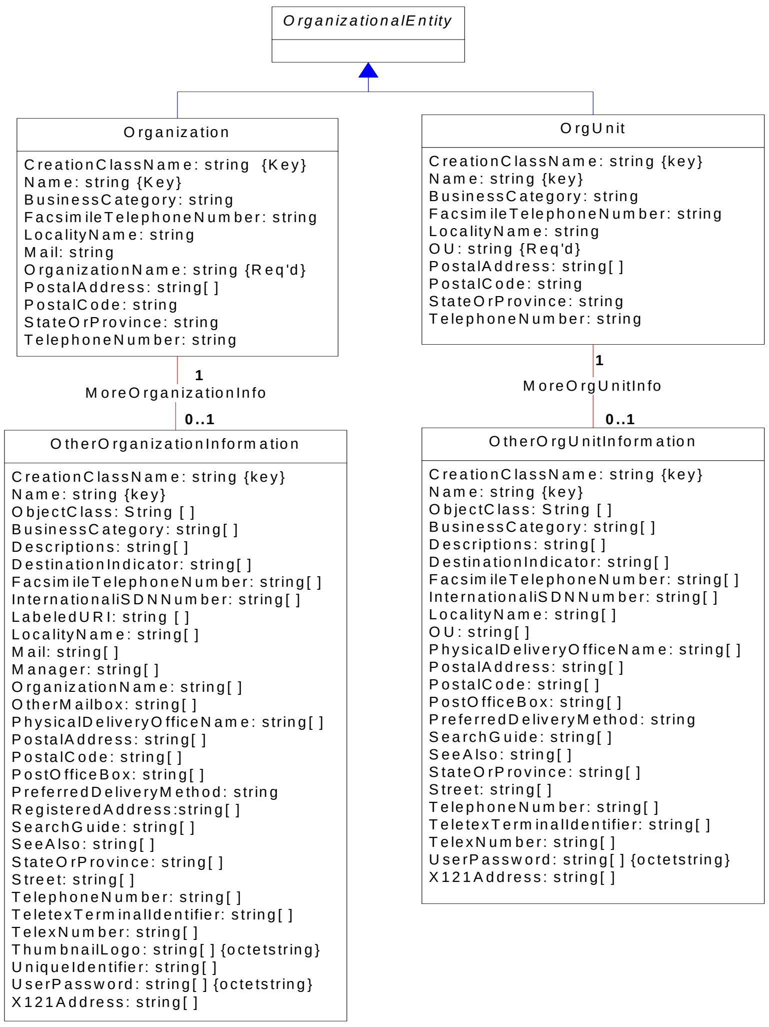
There are two types of OrganizationalEntities. (See Figure 8.2.5.5.4.2: "OPeople".) The difference between the two is largely subjective, however this subprofile RECOMMENDS that Organization instances be used to describe businesses, clubs, families, or governments and that OrgUnit instances be used to describe sub-units within Organizations. To make the amount of information provided in these classes more manageable, much of the less commonly used information is defined by properties of the OtherOrganizationInfo and OtherOrgUnitInfo classes respectively,

The key properties: CreationClassName and Name of each shall be fully specified at creation time. Name defines a namespace unique name for the instance of the class. Additionally, the OrganizationName or OU properties are also required and name the OrganizationalEntity.

Either or these classes may be added or deleted by the CreateInstance or DeleteInstance intrinsic methods.

At most one OtherOrganizationalInfo or OtherOrgUnitInfo instance per respective Organization or OrgUnit may be instantiated using CreateInstance or DeleteInstance. When instantiated a MoreOrganizationInfo or MoreOrgUnitInfo association is instantiated to the corresponding Organization or OrgUnit with the same Name. It is an error if either there is no matching instance or there is already an instance of this type with the same Name.

Figure 85: Organizations and OrgUnits



8.2.5.5.4.2 OPeople

A person may be represented by either an Account instance, (see Figure 8.2.5.5.3: "Accounts"), or by a UserContact instance, (see Figure 86: "People".) Subjectively, Accounts are used to authenticate and track user of a system, where UserContacts are used to represent a person to clients of a system.

The Person class subclasses from UserContact and provides additional information about a person. This is further enhanced by OtherPersonInformation.

A Person instance together with an OtherPersonInformation instance provides UserID and Password. As such, the pair could be used for authentication, in place of Account. However this subprofile RECOMMENDS that Account instances be used for authentication and that UserContact instances be used to describe directory information about a person.

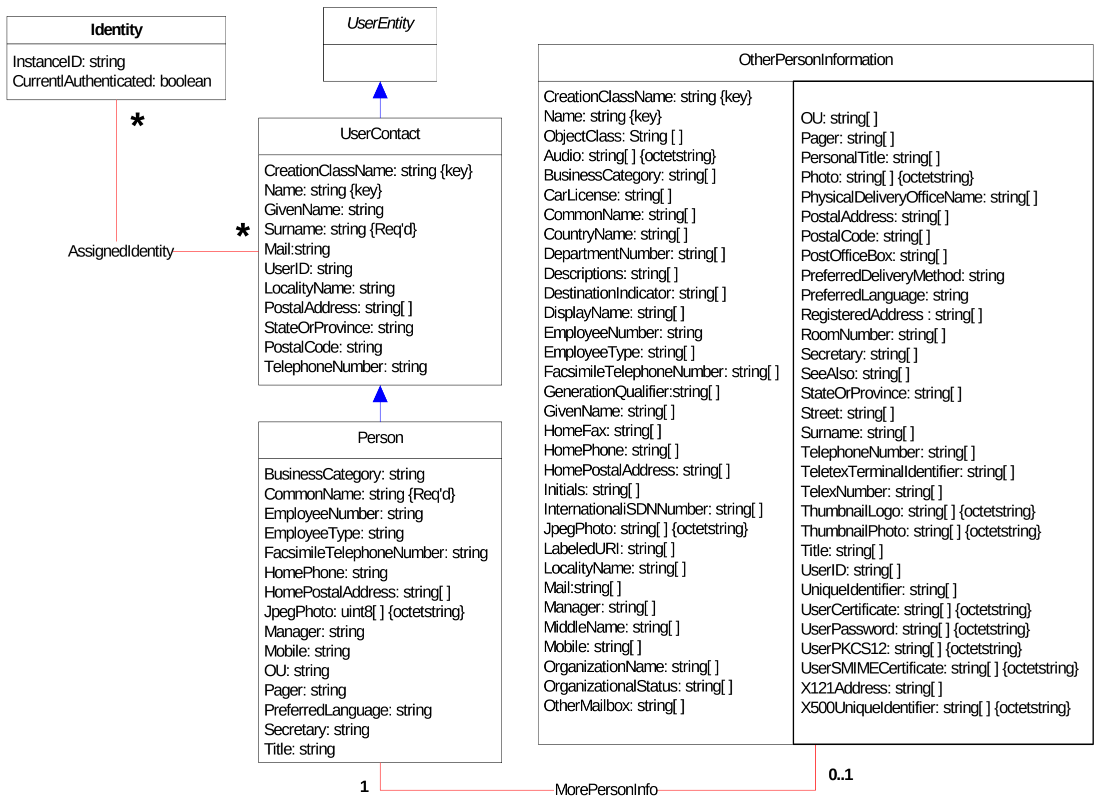
Instances of UserContact, Person, OtherPersonInformation, and MorePersonInfo may be added or deleted using CreateInstance and DeleteInstance intrinsic methods. The key properties of each shall be fully specified at creation time. Additionally the Surname property is required for UserContact or Person instances.

There shall be exactly one Person instance with the same Name property for each instantiated OtherPersonInfo instance.

UserContact and Person instances associated to the same Identity match an Account instance with the same, non-null UserID. Setting UserID, UserCertificate or UserPassword properties on Account, UserContact or Person instances shall also set the corresponding entries in matching instances.

For this subprofile, when a UserContact or Person instance is created, it is mandatory to create an Identity associated via AssignedIdentity.

Figure 86: People



8.2.5.5.5 Groups

A Group is an aggregation of ManagedElements. These shall be Identities. An Identity is assigned to a Group via AssignedIdentity in order to assign privileges to a Group or to incorporate a Group into a Role. Unless otherwise specified, the Authentication policy for the Group Identity is that a successful authentication of a MemberOfCollection Identity also authenticates the Group Identity for that user.

Both Groups and Roles may be aggregated via OwningCollectionElement into an OrganizationalEntity instance.

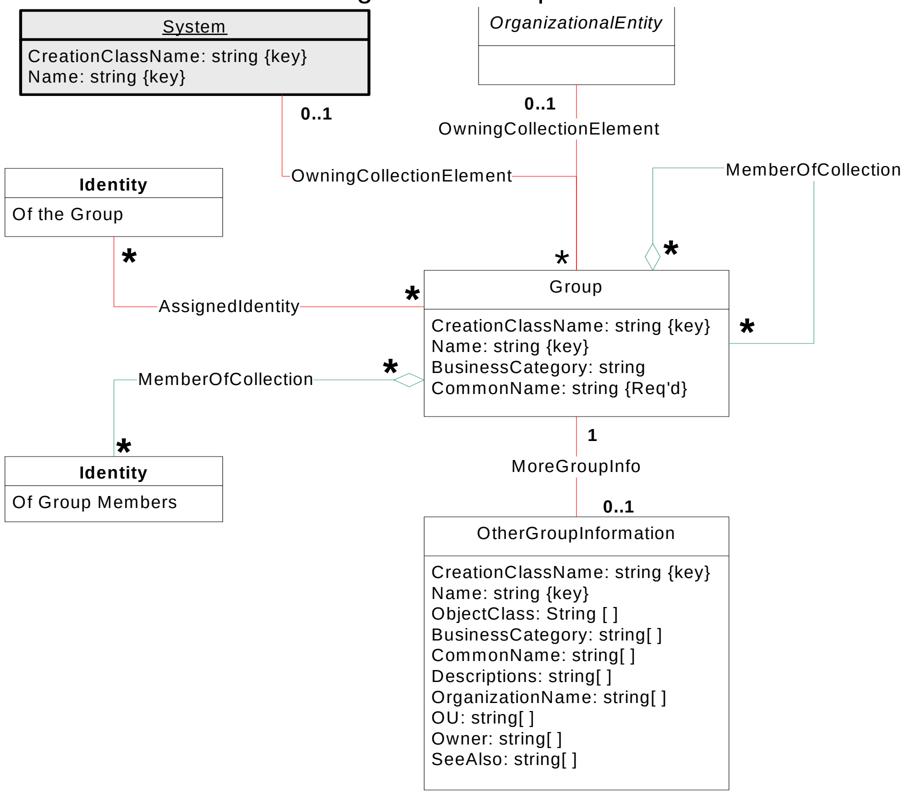
Member information defined by an OtherGroupInformation instance may be associated to a Group via the MoreGroupInfo association.

All of these associations, OwningCollectionElement, MemberOfCollection, AssignedIdentity, and MoreGroupInformation, may be added or deleted via CreateInstance or DeleteInstance intrinsic methods: The key properties of each shall be fully specified at creation time.

All may be added or deleted using CreateInstance and DeleteInstance intrinsic methods. The key properties of each shall be fully specified at creation time. In addition to their keys, both Roles and Groups require that the CommonName property shall be specified at creation time.

There shall be exactly one Group instance with the same Name property for each instantiated OtherGroupInformation instance.

Figure 87: Groups and Roles



8.2.5.5.6 Client Considerations and Recipes

8.2.5.5.6.1 Create a new User instance with an associated Identity

```
// DESCRIPTION
// This recipe will create a UserContact and an associated Identity.
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// $User is a template supplied by the application for a new
//  instance of the class CIM_UserContact or one of its subclasses.
//  It is up to the incorporating profile to define exactly what subclass of
//  UserContact and any constraints on properties that must be filled in and what
values are permissible.

// Create a new Identity for the UserContact
//
$Identity = NewInstance("CIM_Identity")
$Identity-> = CreateInstance($Identity)

// Create the UserContact instance
//
$User-> = CreateInstance($User);

// Create AssignedIdentity between UserContact and Identity
//
$AssignedIdentity = NewInstance("CIM_AssignedIdentity")
$AssignedIdentity.IdentityInfo = $Identity->
$AssignedIdentity.ManagedElement= $User->
$AssignedIdentity-> = CreateInstance($AssignedIdentity)

// ON OUTPUT
//
// $User-> References the User
// $Identity-> References the Identity of the Account
// $AssignedIdentity-> References the AssignedIdentity association
```

8.2.5.5.6.2 Create an Account for an Identity

```
// DESCRIPTION
// This recipe creates an Account and attaches it to an existing Identity.

// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// $Identity-> points to an Identity.
//
// $Account contains a new Account.
//   Account.UserID MUST be set.  It is synonomous with User Name.
//          If the named Identity has an AssignedIdentity association to a
UserContact instance, then
//          theAccount.UserID MUST match that of UserContact.
//      Account.Password must be set to the encrypted value that it will compare
to.

// Create the Account
//
$Account-> = CreateInstance($Account);
```

```
// Create ConcreteIdentity between Account and Identity
//
$ConcreteIdentity = NewInstance(“CIM_ConcreteIdentity”)
$ConcreteIdentity.SameElement = $Identity->
$ConcreteIdentity.SystemElement = $Account->
$ConcreteIdentity-> = CreateInstance($ConcreteIdentity)

// ON OUTPUT
//
// $Account-> References the Account
// $Identity-> References the Identity of the Account
// $ConcreteIdentity-> References the ConcreteIdentity association
```

8.2.5.5.6.3 Create an Account and attach it to an existing User

```
// DESCRIPTION
// This recipe creates an Account and attach it to an existing User.

// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// $User-> points to an UserContact.
//      This recipe assumes that each UserContact instance has at least one
Identity
//      A user may have zero or more accounts.  Each Account/User pair has exactly
one Identity.
//      Account and Identity correlate on UserID
//
// $Account contains a new Account.
//   Account.UserID MUST be set.  It is synonomous with User Name.
//          If the named Identity has an AssignedIdentity association to a
UserContact instance, then
//          theAccount.UserID MUST match that of UserContact.
//      Account.Password must be set to the encrypted value that it will compare
to.



//  Get Identities currrently assigned to the User.
//
$Identity->[] = AssociatorNames ($User->, “CIM_AssignedIdentity”,null,null)

// Case 1: Account.UserID matches User.UserID
//
if ($Account.UserID = $User->UserID)
{
    // This is the principal Account.
    // To simplify, this recipe assumes this is the first Account added.
    //
    if ($Identity->[]size() != 1)
    {
     <ERROR! Expecting exactly one Identity when adding principal account>
    }

    $Account2->[] = AssociatorNames ($Identity[1]->,
“CIM_ConcreteIdentity”,”CIM_Account”,null,null)
    if ($Account2->[]size() != 0)
    {
     <ERROR! Principal account is already added.>
    }

    // Create the Account
    //
    $Account-> = CreateInstance($Account);

    // Create ConcreteIdentity between Account and Identity
    //
    $ConcreteIdentity = NewInstance(“CIM_ConcreteIdentity”)
    $ConcreteIdentity.SameElement = $Identity->[1]
    $ConcreteIdentity.SystemElement = $Account->
    $ConcreteIdentity-> = CreateInstance($ConcreteIdentity)
    <EXIT: “Principal Account Added”>
}

// If we are here, we are adding a secondary account.  We assume the account does
not already exist.
//  But don’t take it for granted.

for #i in $Identity->[]
{
    $Account2[] = AssociatorNames ($Identity->[#i],
“CIM_ConcreteIdentity”,”CIM_Account”,null,null)
    for #j in $Account2->[]
    {
        if (Account.UserID = Account2->[#j].UserID)
        {
         <ERROR! Specified secondary account is already added.>
        }
    }
}

// Create the Account and create a new Identity instance together with
associations.
//
$Account-> = CreateInstance($Account);
$Identity = NewInstance($Identity);
$Identity-> = CreateInstance($Identity);

// Create ConcreteIdentity between Account and Identity
```

```
//
$ConcreteIdentity = NewInstance(“CIM_ConcreteIdentity”)
$ConcreteIdentity.SameElement = $Identity->
$ConcreteIdentity.SystemElement = $Account->
$ConcreteIdentity-> = CreateInstance($ConcreteIdentity)

// Create AssignedIdentity between User and Identity
//
$AssignedIdentity = NewInstance(“CIM_AssignedIdentity”)
$AssignedIdentity.IdentityInfo = $Identity->
$AssignedIdentity.ManagedElement= $User->
$AssignedIdentity-> = CreateInstance($AssignedIdentity)

// Check that all these instances are created
//
try {
     $Account = GetInstance($Account->)
     $Identity = GetInstance($Identity->)
     $ConcreteIdentity = GetInstance($ConcreteIdentity->)
     $AssignedIdentity = GetInstance($AssignedIdentity->)
}
catch (CIM Exception $Exception) {
     throw $Exception
}




<EXIT: “Secondary Account Added”>
```

8.2.5.5.7 Registered Name and Version

Security Identity Management version 1.1.0

8.2.5.5.8 CIM Server Requirements

Table 495: CIM Server Requirements for Security Identity Management

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.5.5.9 CIM Elements

Table 496: CIM Elements for Security Identity Management

Element Name	Description
Mandatory Classes	
CIM_AccountMapsToAccount (8.2.5.5.9.3)	
CIM_AccountOnSystem (8.2.5.5.9.4)	
CIM_AssignedIdentity (8.2.5.5.9.5)	
CIM_ConcreteDependency (8.2.5.5.9.7)	
CIM_ConcreteIdentity (8.2.5.5.9.8)	
CIM_ElementConformsToProfile (8.2.5.5.9.9)	
CIM_HostedService (8.2.5.5.9.12)	
CIM_IdentityContext (8.2.5.5.9.15)	
CIM_ManagesAccount (8.2.5.5.9.17)	
CIM_RegisteredSubProfile (8.2.5.5.9.34)	
CIM_ServiceAvailableToElement (8.2.5.5.9.35)	
CIM_SubProfileRequiresProfile (8.2.5.5.9.37)	
CIM_System (8.2.5.5.9.38)	
Optional Classes	
CIM_Account (8.2.5.5.9.1)	
CIM_AccountManagementService (8.2.5.5.9.2)	
CIM_AuthenticationService (8.2.5.5.9.6)	
CIM_GatewayPathID (8.2.5.5.9.10)	
CIM_Group (8.2.5.5.9.11)	
CIM_IPNetworkIdentity (8.2.5.5.9.13)	
CIM_Identity (8.2.5.5.9.14)	
CIM_ManagedElement (8.2.5.5.9.16)	
CIM_MemberOfCollection (8.2.5.5.9.18)	
CIM_MoreGroupInfo (8.2.5.5.9.19)	
CIM_MoreOrgUnitInfo (8.2.5.5.9.20)	
CIM_MoreOrganizationInfo (8.2.5.5.9.21)	
CIM_MorePersonInfo (8.2.5.5.9.22)	
CIM_OrgStructure (8.2.5.5.9.23)	
CIM_OrgUnit (8.2.5.5.9.24)	
CIM_Organization (8.2.5.5.9.25)	
CIM_OrganizationalEntity (8.2.5.5.9.26)	
CIM_OtherGroupInformation (8.2.5.5.9.27)	
CIM_OtherOrgUnitInformation (8.2.5.5.9.28)	
CIM_OtherOrganizationInformation (8.2.5.5.9.29)	
CIM_OtherPersonInformation (8.2.5.5.9.30)	
CIM_OwningCollectionElement (8.2.5.5.9.31)	shall not be present if CollectionInOrganization is present.

Table 496: CIM Elements for Security Identity Management

Element Name	Description
CIM_Person (8.2.5.5.9.32)	
CIM_RegisteredProfile (8.2.5.5.9.33)	
CIM_StorageHardwareID (8.2.5.5.9.36)	
CIM_UserContact (8.2.5.5.9.39)	

8.2.5.5.9.1 CIM_Account

Class Mandatory: false

Table 497: SMI Referenced Properties/Methods for CIM_Account

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key
Userid		string	The name the user is known by in the System. Matches any UserContact or Person with the same value. Changing here changes corresponding values on matching UserContact or Person instances.
UserCertificate		string[]	The Public Key Certificate of this user. Changing here changes corresponding values on matching UserContact or Person instances.
UserPassword		string[]	The password used with the UserID. Changing here changes corresponding values on matching UserContact or Person instances.

8.2.5.5.9.2 CIM_AccountManagementService

Class Mandatory: false

Table 498: SMI Referenced Properties/Methods for CIM_AccountManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.5.9.3 CIM_AccountMapsToAccount

Class Mandatory: true

Table 499: SMI Referenced Properties/Methods for CIM_AccountMapsToAccount

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Account	Key
Dependent		CIM_Account	Key

8.2.5.5.9.4 CIM_AccountOnSystem

Class Mandatory: true

Table 500: SMI Referenced Properties/Methods for CIM_AccountOnSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	Key
PartComponent		CIM_Account	Key

8.2.5.5.9.5 CIM_AssignedIdentity

Class Mandatory: true

Table 501: SMI Referenced Properties/Methods for CIM_AssignedIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IdentityInfo		CIM_Identity	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.5.9.6 CIM_AuthenticationService

Class Mandatory: false

Table 502: SMI Referenced Properties/Methods for CIM_AuthenticationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.5.9.7 CIM_ConcreteDependency

Class Mandatory: true

Table 503: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Key

Table 503: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Dependent		CIM_ManagedElement	Key

8.2.5.5.9.8 CIM_ConcreteIdentity

Class Mandatory: true

Table 504: SMI Referenced Properties/Methods for CIM_ConcreteIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemElement		CIM_ManagedElement	Key
SameElement		CIM_ManagedElement	Key

8.2.5.5.9.9 CIM_ElementConformsToProfile

Class Mandatory: true

Table 505: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.5.9.10 CIM_GatewayPathID

Class Mandatory: false

Table 506: SMI Referenced Properties/Methods for CIM_GatewayPathID

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	True if currently authenticated

8.2.5.5.9.11 CIM_Group

Class Mandatory: false

Table 507: SMI Referenced Properties/Methods for CIM_Group

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key
CommonName		string	The Name by which the Group is known

8.2.5.5.9.12 CIM_HostedService

Class Mandatory: true

Table 508: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.5.9.13 CIM_IPNetworkIdentity

Class Mandatory: false

Table 509: SMI Referenced Properties/Methods for CIM_IPNetworkIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	True if currently authenticated

8.2.5.5.9.14 CIM_Identity

Class Mandatory: false

Table 510: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	True if currently authenticated

8.2.5.5.9.15 CIM_IdentityContext

Class Mandatory: true

Table 511: SMI Referenced Properties/Methods for CIM_IdentityContext

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementInContext		CIM_Identity	Key
ElementProvidingContext		CIM_ManagedElement	Key

8.2.5.5.9.16 CIM_ManagedElement

Class Mandatory: false

No specified properties or methods.

8.2.5.5.9.17 CIM_ManagesAccount

Class Mandatory: true

Table 512: SMI Referenced Properties/Methods for CIM_ManagesAccount

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_AccountManagementService	Key
Dependent		CIM_Account	Key

8.2.5.5.9.18 CIM_MemberOfCollection

Class Mandatory: false

Table 513: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	
Member		CIM_ManagedElement	

8.2.5.5.9.19 CIM_MoreGroupInfo

Class Mandatory: false

Table 514: SMI Referenced Properties/Methods for CIM_MoreGroupInfo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Group	
Dependent		CIM_OtherGroupInformation	

8.2.5.5.9.20 CIM_MoreOrgUnitInfo

Class Mandatory: false

Table 515: SMI Referenced Properties/Methods for CIM_MoreOrgUnitInfo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_OrgUnit	
Dependent		CIM_OtherOrgUnitInformation	

8.2.5.5.9.21 CIM_MoreOrganizationInfo

Class Mandatory: false

Table 516: SMI Referenced Properties/Methods for CIM_MoreOrganizationInfo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Organization	

Table 516: SMI Referenced Properties/Methods for CIM_MoreOrganizationInfo

Property	Flags	Type	Description & Notes
Dependent		CIM_OtherOrganizationInformation	

8.2.5.5.9.22 CIM_MorePersonInfo

Class Mandatory: false

Table 517: SMI Referenced Properties/Methods for CIM_MorePersonInfo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Person	
Dependent		CIM_OtherPersonInformation	

8.2.5.5.9.23 CIM_OrgStructure

Class Mandatory: false

Table 518: SMI Referenced Properties/Methods for CIM_OrgStructure

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Parent		CIM_OrganizationalEntity	
Child		CIM_OrganizationalEntity	

8.2.5.5.9.24 CIM_OrgUnit

Class Mandatory: false

Table 519: SMI Referenced Properties/Methods for CIM_OrgUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key
OU		string	The Name by which the Organizational Unit is known

8.2.5.5.9.25 CIM_Organization

Class Mandatory: false

Table 520: SMI Referenced Properties/Methods for CIM_Organization

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

Table 520: SMI Referenced Properties/Methods for CIM_Organization

Property	Flags	Type	Description & Notes
OrganizationName		string	The Name by which the Organization is known

8.2.5.5.9.26 CIM_OrganizationalEntity

Class Mandatory: false
No specified properties or methods.

8.2.5.5.9.27 CIM_OtherGroupInformation

Class Mandatory: false

Table 521: SMI Referenced Properties/Methods for CIM_OtherGroupInformation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key, Must match that of Group

8.2.5.5.9.28 CIM_OtherOrgUnitInformation

Class Mandatory: false

Table 522: SMI Referenced Properties/Methods for CIM_OtherOrgUnitInformation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key, Must match that of OrgUnit.

8.2.5.5.9.29 CIM_OtherOrganizationInformation

Class Mandatory: false

Table 523: SMI Referenced Properties/Methods for CIM_OtherOrganizationInformation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key, Must match that of Organization.

8.2.5.5.9.30 CIM_OtherPersonInformation

Class Mandatory: false

Table 524: SMI Referenced Properties/Methods for CIM_OtherPersonInformation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key, Must match that of Person.

Table 524: SMI Referenced Properties/Methods for CIM_OtherPersonInformation

Property	Flags	Type	Description & Notes
UserID		string[]	The Name by which the User is known to the System. Matches all Account or Person instances in the namespace with the same UserID. Changing here changes corresponding values on matching UserContact or Account instances.
UserCertificate		string[]	The Public Key Certificate of this user. Changing here changes corresponding values on matching UserContact or Account instances.
UserPassword		string[]	The password used with the UserID. Changing here changes the corresponding values on matching UserContact or Account instances.

8.2.5.5.9.31 CIM_OwningCollectionElement

shall not be present if CollectionInOrganization is present.
Class Mandatory: false

Table 525: SMI Referenced Properties/Methods for CIM_OwningCollectionElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
OwningElement		CIM_ManagedElement	
OwnedElement		CIM_Collection	

8.2.5.5.9.32 CIM_Person

Class Mandatory: false

Table 526: SMI Referenced Properties/Methods for CIM_Person

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key
Surname		string	The Name by which the User is known to other Persons
UserID		string	The Name by which the User is known to the System. Matches all Account or Person instances in the namespace with the same UserID. Changing here changes corresponding values on matching UserContact or Account instances.

8.2.5.5.9.33 CIM_RegisteredProfile

Class Mandatory: false

Table 527: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	Parent subprofile

8.2.5.5.9.34 CIM_RegisteredSubProfile

Class Mandatory: true

Table 528: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.5.9.35 CIM_ServiceAvailableToElement

Class Mandatory: true

Table 529: SMI Referenced Properties/Methods for CIM_ServiceAvailableToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ServiceProvided		CIM_Service	Key
UserOfService		CIM_ManagedElement	Key

8.2.5.5.9.36 CIM_StorageHardwareID

Class Mandatory: false

Table 530: SMI Referenced Properties/Methods for CIM_StorageHardwareID

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	True if currently authenticated

8.2.5.5.9.37 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 531: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.5.9.38 CIM_System

Class Mandatory: true

Table 532: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.5.9.39 CIM_UserContact

Class Mandatory: false

Table 533: SMI Referenced Properties/Methods for CIM_UserContact

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key
Surname		string	The Name by which the User is known to other users.
UserID		string	The Name by which the User is known to the System. Matches all Account or Person instances in the namespace with the same UserID. Changing here changes corresponding values on matching Person or Account instances.

8.2.5.5.10 Related Standards

Table 534: Related Standards for Security Identity Management

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.6 CredentialManagement Subprofile

8.2.5.6.1 Description

This subprofile provides for management of credentials used by a client to establish its identity to a serving system. An administrator of both the client and server systems establishes an Identity for the client on the server system and creates a credential for the client on the client system.

Note: SMI-S Servers are often clients of other services. For instance, a device that is managed by an SMI-S Server may require a login before it allows a client to discover or manage its components. Credentials used to access devices are known within this specification as “device credentials”.

As shown in Figure 88: "Credential Management", this subprofile applies to a System as a whole.

Credentials are created by a LocalCredentialManagementService. There shall be a one or more LocalCredentialManagementService instances on a System conforming to this subprofile.

The Credentials are intended to authenticate a client on this System to a service running on a remote system. There shall be one or more RemoteServiceAccessPoint instances for each of the Systems to which Credentials may be presented.

Credential setup

The administrator needs to have prior knowledge about the type of Credential required by the remote system. The LocalCredentialManagementService is subclassed into two types, a SharedSecretService and a PublicKeyManagementService. If the latter is present, then UnsignedPublicKey Credentials are supported. If the former, then SharedSecretService.Protocol = “SharedSecret” specifies that SharedSecret credentials are supported and SharedSecretService.Protocol = “IKE” specifies that NamedSharedIKESecret credentials are supported

The administrator uses CreateInstance and DeleteInstance to create or delete Credentials. The details of each are described below. In common to all is that the key properties: SystemCreationClassName, SystemName, ServiceCreationClassName, and ServiceName of each Credential shall be fully specified at creation time. This information is used by the system to locate the correct LocalCredentialManagementService instance and to snap the required IKESecretIsNamed, SharedSecretIsShared or LocallyManagedPublicKey associations. Additionally certain remaining properties of each credential shall be filled in as described below.

- Expires: Set to the datetime after which this credential will not be valid. Use a value of “99991231235959.999999+999” if this field is to be ignored.

SharedSecret Credential

- RemoteID: Set to the User ID or other value by which the client is known on the remote system. Typically this will correspond to Account.Userid or Person.UserID as stored on the remote system.
- Secret: Set to the password or other value by which the client is authenticated on the remote system. The value is provided in clear text. There is an underlying assumption that there is a secure communication path being used between the administrator and the CIM Service on the client system. This property is writable, but shall not be readable. Typically this will correspond to Account.Userid or Person.UserID as stored on the remote system.

NamedSharedIKE Credential

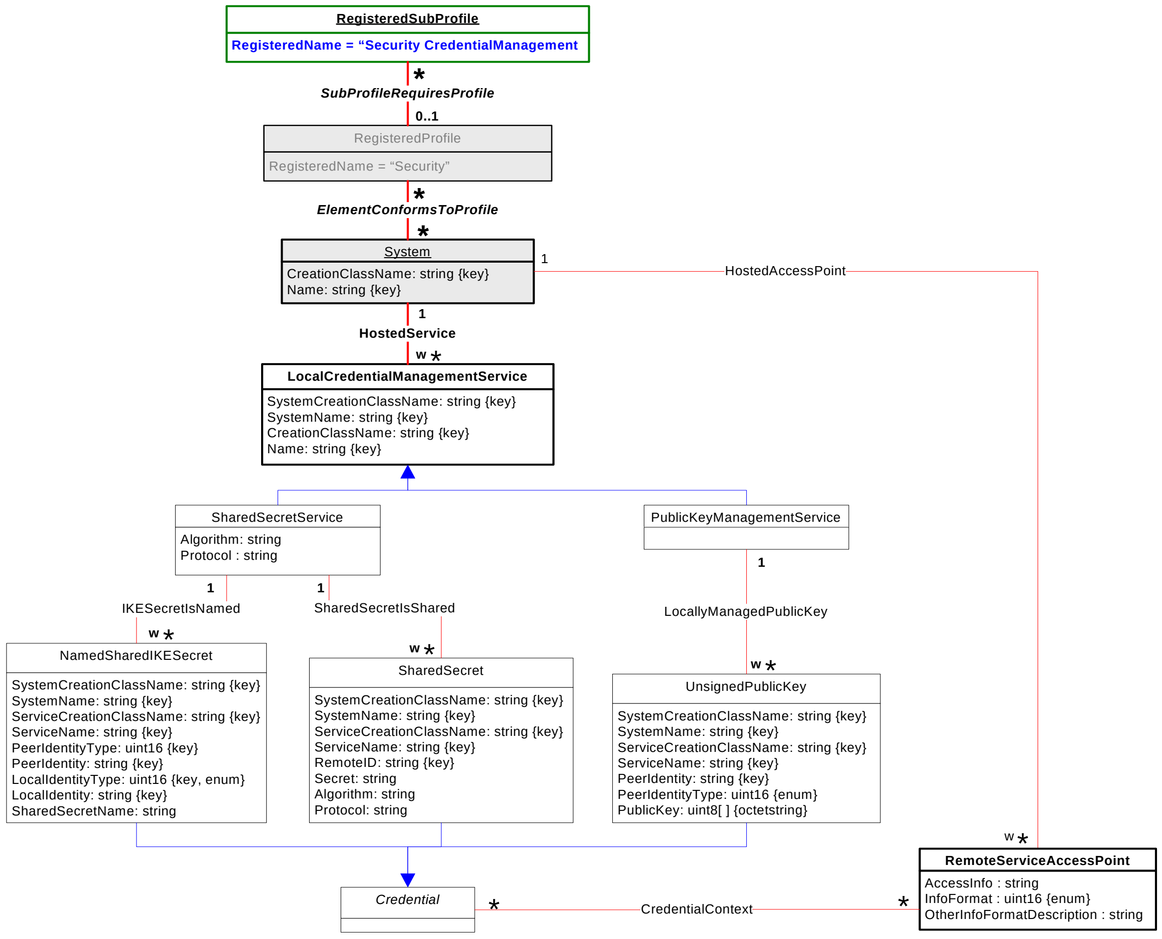
- PeerIdentityType: This describes the type of identity used to locate the remote peer. It is an enumerated type that shall correspond to one of the following values: “IPV4_ADDR”, “FQDN”, “USER_FQDN”, “IPV4_ADDR_SUBNET”,”IPV6_ADDR”, “IPV6_ADDR_SUBNET”, "IPV4_ADDR_RANGE", "IPV6_ADDR_RANGE", "DER_ASN1_DN", "DER_ASN1_GN", or "KEY_ID".

- PeerIdentity: An identity value conforming to the PeerIdentityType and naming the remote peer with whom a direct trust relation exists.
- LocalIdentityType: This describes the type of identity used to name the local peer. It is an enumerated type that shall correspond to one of the following values: "IPV4_ADDR", "FQDN", "USER_FQDN", "IPV4_ADDR_SUBNET","IPV6_ADDR", "IPV6_ADDR_SUBNET", "IPV4_ADDR_RANGE", "IPV6_ADDR_RANGE", "DER_ASN1_DN", "DER_ASN1_GN", or "KEY_ID".
- LocalIdentity: An identity value conforming to the LocalIdentityType and naming the local peer with whom a direct trust relation exists.
- SharedSecretName: On creation, this is set to the password or other shared value used to authenticate the client. When read, this is an indirect reference to a shared secret. The SecretService does not expose the actual secret.

UnsignedPublicKey Credential

- PeerIdentityType: This describes the type of identity used to locate the remote peer. It is an enumerated type that shall correspond to one of the following values: "IPV4_ADDR", "FQDN", "USER_FQDN", "IPV4_ADDR_SUBNET","IPV6_ADDR", "IPV6_ADDR_SUBNET", "IPV4_ADDR_RANGE", "IPV6_ADDR_RANGE", "DER_ASN1_DN", "DER_ASN1_GN", or "KEY_ID".
- PeerIdentity: An identity value conforming to the PeerIdentityType and naming the remote peer with whom a direct trust relation exists.
- PublicKey: The DER-encoded raw public key.

Figure 88: Credential Management


RegisteredSubProfile
RegisteredName = "Security CredentialManagement
SubProfileRequiresProfile
0..1
RegisteredProfile
RegisteredName = "Security"
ElementConformsToProfile
System
CreationClassName: string {key}
Name: string {key}
HostedAccessPoint
HostedService
LocalCredentialManagementService
SystemCreationClassName: string {key}
SystemName: string {key}
CreationClassName: string {key}
Name: string {key}
SharedSecretService
Algorithm: string
Protocol : string
PublicKeyManagementService
IKESecretIsNamed
SharedSecretIsShared
LocallyManagedPublicKey
NamedSharedIKESecret
SystemCreationClassName: string {key}
SystemName: string {key}
ServiceCreationClassName: string {key}
ServiceName: string {key}
PeerIdentityType: uint16 {key}
PeerIdentity: string {key}
LocalIdentityType: uint16 {key, enum}
LocalIdentity: string {key}
SharedSecretName: string
SharedSecret
SystemCreationClassName: string {key}
SystemName: string {key}
ServiceCreationClassName: string {key}
ServiceName: string {key}
RemoteID: string {key}
Secret: string
Algorithm: string
Protocol: string
UnsignedPublicKey
SystemCreationClassName: string {key}
SystemName: string {key}
ServiceCreationClassName: string {key}
ServiceName: string {key}
PeerIdentity: string {key}
PeerIdentityType: uint16 {enum}
PublicKey: uint8[ ] {octetstring}
RemoteServiceAccessPoint
AccessInfo : string
InfoFormat : uint16 {enum}
OtherInfoFormatDescription : string
Credential
CredentialContext

Credential Use

Once set up, a Credential may be enabled or disabled for use by using CreateInstance or DeleteInstance to add or remove CredentialContext associations between a Credential and the RemoteServiceAccessPoint used to access a remote system.

8.2.1.4, "Device Credentials Subprofile" for a complete discussion of the SMI-S requirements for modeling device credentials.

The SMI-S Server shall securely store the device credentials local to the SMI-S Server. A proxy SMI-S Server may need to store the credentials on disk so that they are available upon reboot. In this case the credentials shall be encrypted for confidentiality.

The device credentials shall be transmitted securely from the SMI-S Server to the device. The mechanism of communicating the credentials to the device is outside the scope of this specification, but it should be over a secure channel if possible.

A SMI-S Server may be configured with the device credentials necessary to talk to the device. If a SMI-S Server supports SSL 3.0 or TLS, the HTTP Client shall use SSL 3.0 or TLS to pass device credentials to the SMI-S Server. When new device credentials are passed to an SMI-S Server, the device credential information in the device shall be updated immediately.

Only the SMI-S Server responsible for communicating with the device has access to the properties of the SharedSecret object. No other SMI-S Client may read the Secret property of this object as it shall be implemented Write-Only.

8.2.5.6.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.5.6.3 Cascading Considerations

Not defined in this standard.

8.2.5.6.4 Supported Subprofiles and Packages

None.

8.2.5.6.5 Methods of the Profile

None.

8.2.5.6.6 Client Considerations and Recipes

None.

8.2.5.6.7 Registered Name and Version

Security Credential Management version 1.1.0

8.2.5.6.8 CIM Server Requirements

Table 535: CIM Server Requirements for Security Credential Management

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.5.6.9 CIM Elements

Table 536: CIM Elements for Security Credential Management

Element Name	Description
Mandatory Classes	
CIM_ElementConformsToProfile (8.2.5.6.9.2)	
CIM_HostedAccessPoint (8.2.5.6.9.3)	
CIM_HostedService (8.2.5.6.9.4)	
CIM_NamedSharedIKESecret (8.2.5.6.9.7)	
CIM_PublicKeyManagementService (8.2.5.6.9.8)	
CIM_RegisteredSubProfile (8.2.5.6.9.10)	
CIM_RemoteServiceAccessPoint (8.2.5.6.9.11)	
CIM_SharedSecret (8.2.5.6.9.12)	
CIM_SharedSecretService (8.2.5.6.9.14)	
CIM_SubProfileRequiresProfile (8.2.5.6.9.15)	
CIM_System (8.2.5.6.9.16)	
CIM_UnsignedPublicKey (8.2.5.6.9.17)	
Optional Classes	
CIM_CredentialContext (8.2.5.6.9.1)	
CIM_IKESecretIsNamed (8.2.5.6.9.5)	
CIM_LocallyManagedPublicKey (8.2.5.6.9.6)	
CIM_RegisteredProfile (8.2.5.6.9.9)	
CIM_SharedSecretIsShared (8.2.5.6.9.13)	

8.2.5.6.9.1 CIM_CredentialContext

Class Mandatory: false

Table 537: SMI Referenced Properties/Methods for CIM_CredentialContext

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementInContext		CIM_Credential	Key
ElementProvidingContext		CIM_ManagedElement	Key

8.2.5.6.9.2 CIM_ElementConformsToProfile

Class Mandatory: true

Table 538: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.6.9.3 CIM_HostedAccessPoint

Class Mandatory: true

Table 539: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_ServiceAccessPoint	Key

8.2.5.6.9.4 CIM_HostedService

Class Mandatory: true

Table 540: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.6.9.5 CIM_IKESecretIsNamed

Class Mandatory: false

Table 541: SMI Referenced Properties/Methods for CIM_IKESecretIsNamed

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SharedSecretService	Key
Dependent		CIM_NamedSharedIKESecret	Key

8.2.5.6.9.6 CIM_LocallyManagedPublicKey

Class Mandatory: false

Table 542: SMI Referenced Properties/Methods for CIM_LocallyManagedPublicKey

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PublicKeyManagementService	Key
Dependent		CIM_UnsignedPublicKey	Key

8.2.5.6.9.7 CIM_NamedSharedIKESecret

Class Mandatory: true

Table 543: SMI Referenced Properties/Methods for CIM_NamedSharedIKESecret

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
ServiceCreationClassName		string	Key
ServiceName		string	Key
PeerIdentity		string	Key, The identity of the remote peer trusted entity.
PeerIdentityType		uint16	The type of the remote PeerIdentity.
LocalIdentity		string	Key, The identity of the local peer trusted entity.
LocalIdentityType		uint16	The type of the LocalIdentity.
SharedSecretName	M	string	The name of the shared secret,

8.2.5.6.9.8 CIM_PublicKeyManagementService

Class Mandatory: true

Table 544: SMI Referenced Properties/Methods for CIM_PublicKeyManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.6.9.9 CIM_RegisteredProfile

Class Mandatory: false

Table 545: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	Parent subprofile

8.2.5.6.9.10 CIM_RegisteredSubProfile

Class Mandatory: true

Table 546: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.6.9.11 CIM_RemoteServiceAccessPoint

Class Mandatory: true

Table 547: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.6.9.12 CIM_SharedSecret

Class Mandatory: true

Table 548: SMI Referenced Properties/Methods for CIM_SharedSecret

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
ServiceCreationClassName		string	Key
ServiceName		string	Key
RemoteID		string	Key, The identity of the client as known on the remote system.
Secret		string	A secret

8.2.5.6.9.13 CIM_SharedSecretIsShared

Class Mandatory: false

Table 549: SMI Referenced Properties/Methods for CIM_SharedSecretIsShared

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SharedSecretService	Key
Dependent		CIM_SharedSecret	Key

8.2.5.6.9.14 CIM_SharedSecretService

Class Mandatory: true

Table 550: SMI Referenced Properties/Methods for CIM_SharedSecretService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key
Protocol	M	string	Select IKE'forSharedIKEsecret-sand'SharedSecret'forSharedsecrets.'

8.2.5.6.9.15 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 551: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.6.9.16 CIM_System

Class Mandatory: true

Table 552: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.6.9.17 CIM_UnsignedPublicKey

Class Mandatory: true

Table 553: SMI Referenced Properties/Methods for CIM_UnsignedPublicKey

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
ServiceCreationClassName		string	Key
ServiceName		string	Key
PeerIdentity		string	Key, The identity of the peer trusted entity.
PeerIdentityType		uint16	The type of the PeerIdentity.

Table 553: SMI Referenced Properties/Methods for CIM_UnsignedPublicKey

Property	Flags	Type	Description & Notes
PublicKey	M	uint8[]	Key, The identity of the peer trusted entity.

8.2.5.6.10 Related Standards

Table 554: Related Standards for Security Credential Management

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.5.7 3rd Party Authentication Subprofile

8.2.5.7.1 Description

This subprofile extends the Security IdentityManagement profile by specifying the necessary elements required to manage the relationships between a CIM Server and 3rd party Authentication Servers such as Radius.

The implementation shall use a HostedService association between the System and the AuthenticationService.

In this environment, the local AuthenticationService may delegate authentication requests to a 3rd-party authentication service which is accessed through a RemoteServiceAccessPoint as shown in Figure 89: "3rd Party Authentication for the CIM Service". The implementation shall instantiate a ServiceSAPDependency between the RemoteServiceAccessPoint and the AuthenticationService.

If the 3rd Party Authentication Service requires that the local system authenticate itself, then the required Credential is associated via CredentialContext to the RemoteServiceAccessPoint instance. (See the Security CredentialMangement subprofile.) This may be accomplished using intrinsic operations.

UserContact.Name, Group.Name, and Role.Name are used as a correlatable identifier for users, groups, and roles respectively. Note that the UserID property of UserContact is synonymous with a typical user Name. A user may have multiple Identities. This specification restricts a Group to having at most one Identity and does not assign Identities to Roles. An Identity for a UserContact is matched via an AssignedIdentity association and a match on both Name and UserID in the UserContact.

In the event that there is more than one 3rd Party Authentication Service, this profile does not specify the means used by which the local authentication service locates the correct 3rd Party Authentication Service, UserID, and if specified the Realm. See 8.2.4.1.1.4, "HTTP Security" in the Server Profile. A sufficient authentication strategy is to pass the requestor's UserID, Realm and credentials to each Authentication service.

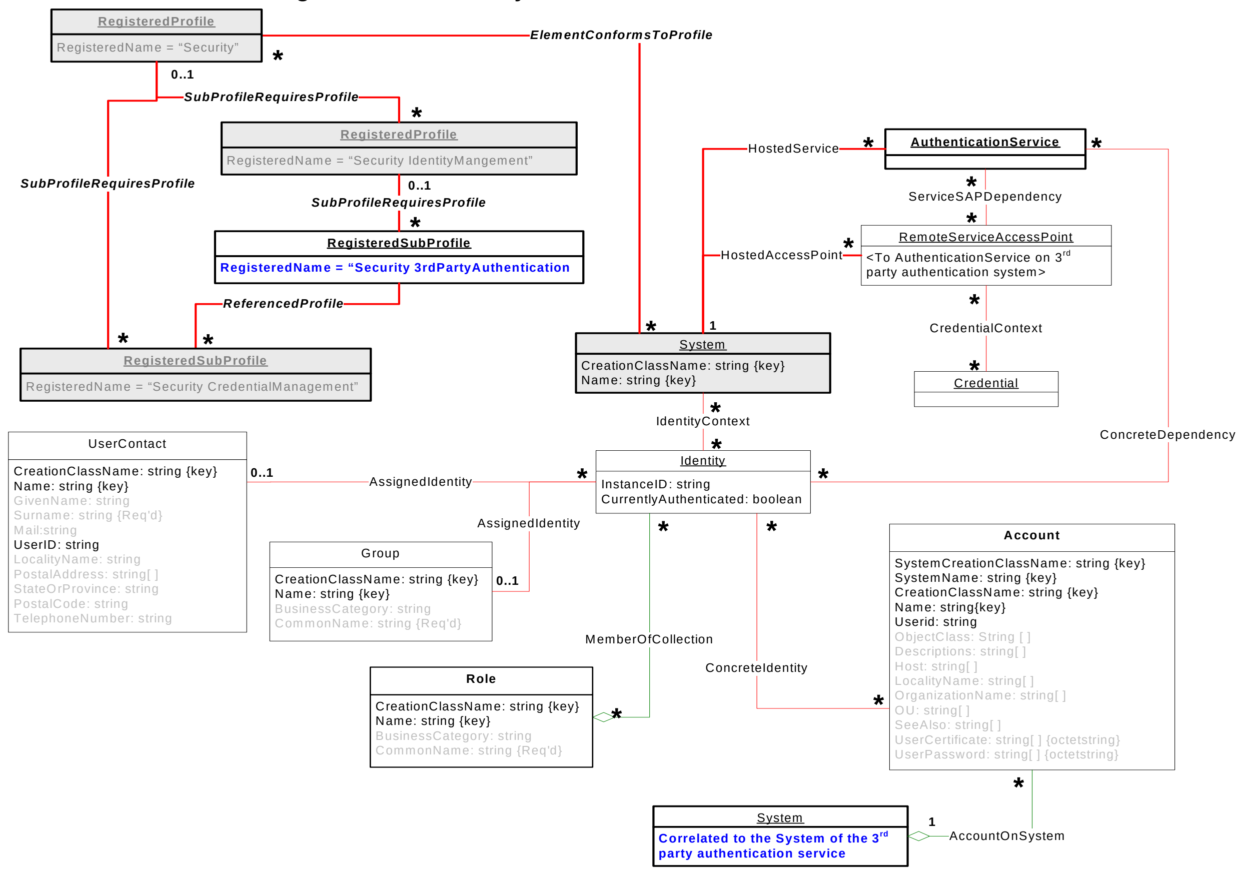
The 3rd Party Authentication Service should respond true or false, and if true should also respond with a list of discontinued names which represent at most one authenticated UserContact and a set of Group, and Role elements to which the authenticated user belongs. Each returned distinguished name matches the Name property of at most one such element.

If a UserContact is matched via Name, the UserID shall match that instance of UserContact or that of an associated Account instance. This specification allows a UserContact to be associated via AssignedIdentity to multiple Identities, which in turn may be associated to at most one Account via ConcreteIdentity. An Identity is selected which has matching Name and either a matching UserID or an associated Account with a matching UserID.   If no match is found, then this user is not known on this system. A profile that incorporates this subprofile may define an AuthenticationRule that designates some other Identity to authenticate in the case a matching Identity is not found by the above algorithm.

Additionally, the 3rd Party Authentication Service may return the distinguished names of groups or roles to which the user belongs. These names correlated to Group.Name or Role.Name. This specification restricts a Group to at most one Identity associated via AssignedIdentity. If a Group is matched, then the user belongs to the group and the Groups Identity is authenticated. If a Role is matched, then the user is authenticated for the Role. Profiles or subprofiles that rely on this profile may further qualify the types of Identity and AuthenticationRules that may be used.

.

Figure 89: 3rd Party Authentication for the CIM Service



8.2.5.7.2 Durable Names and Correlatable IDs of the Profile

When a UserID is passed from an SMI-S Client to an SMI-S Server and then to a 3rd Party Authentication service, there needs to be some means to assure that each is referring to the same entity. The process specified here is for the client to pass the server a UserID, together with Realm and Credential information. The server passes this through to the authentication service, which maps this to a particular user and zero or more groups and roles to which the User belongs. This subprofile specifies that users, groups, and roles need to have unique distinguished names, (see http://www.ietf.org/rfc/rfc1779.txt?number=1779.) These distinguished names are returned to the SMI-S Server by the 3rd Party Authentication service. The SMI-S Server correlates these distinguished names to the Name property of UserContact, Group, or Role instances.

The Identity of a user is determined by a match on both the UserID provided by the SMI-S Client and the distinguished name returned by the 3rd Party Authentication service. (See the algorithm described in the previous section.)

8.2.5.7.3 Client Considerations and Recipes

8.2.5.7.3.1 Create a new User instance with an associated Identity.

The client should use the "Create a new User instance with an associated Identity" recipe defined in the Security Identity Management subprofile. The UserContact (or subclass) instance supplied by the SMI-S Client shall have the Name property set to match the corresponding information on held on the system supporting the 3rd Party Authentication service. The UserID property shall be that of the principal account for that user.

8.2.5.7.3.2 Add an Account for a User.

If more than one Identity is maintained for a user on the SMI-S server, the client should use the “Create an Account and attach it to an existing User.” recipe defined in the Security Identity Management subprofile. The UserContact (or subclass) instance named by the SMI-S client shall correspond by NAME to the distinguished name of the user as known on the system of the 3rd Party Authentication service. If this is the principal account, the UserID property of the Account shall match that of the named UserContact instance. In all cases the UserID property of the supplied Account shall match the UserID used to authenticate the user. Since the Account is not directly authenticated, the Password property shall not be specified.

8.2.5.7.4 Registered Name and Version

Security 3rd Party Authentication version 1.1.0

8.2.5.7.5 CIM Server Requirements

Table 555: CIM Server Requirements for Security 3rd Party Authentication

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.5.7.6 CIM Elements

Table 556: CIM Elements for Security 3rd Party Authentication

Element Name	Description
Mandatory Classes	
CIM_ConcreteDependency (8.2.5.7.6.5)	
CIM_ConcreteIdentity (8.2.5.7.6.6)	
CIM_Credential (8.2.5.7.6.7)	
CIM_ElementConformsToProfile (8.2.5.7.6.9)	
CIM_HostedAccessPoint (8.2.5.7.6.11)	
CIM_HostedService (8.2.5.7.6.12)	
CIM_ReferencedProfile (8.2.5.7.6.16)	
CIM_RegisteredSubProfile (8.2.5.7.6.18)	Specifies additional requirements on an SMI-S Server when it is also a client of a 3rd party authentication service.
CIM_RemoteServiceAccessPoint (8.2.5.7.6.21)	
CIM_ServiceSAPDependency (8.2.5.7.6.23)	
CIM_SubProfileRequiresProfile (8.2.5.7.6.24)	
CIM_System (8.2.5.7.6.26)	
Optional Classes	
CIM_Account (8.2.5.7.6.1)	
CIM_AccountOnSystem (8.2.5.7.6.2)	
CIM_AssignedIdentity (8.2.5.7.6.3)	
CIM_AuthenticationService (8.2.5.7.6.4)	
CIM_CredentialContext (8.2.5.7.6.8)	
CIM_Group (8.2.5.7.6.10)	
CIM_Identity (8.2.5.7.6.13)	
CIM_IdentityContext (8.2.5.7.6.14)	
CIM_MemberOfCollection (8.2.5.7.6.15)	
CIM_RegisteredProfile (8.2.5.7.6.17)	

Table 556: CIM Elements for Security 3rd Party Authentication

Element Name	Description
CIM_RegisteredSubProfile (8.2.5.7.6.19)	Specifies additional requirements on an SMI-S Server that is also a client of some other service that enforces security.
CIM_RegisteredSubProfile (8.2.5.7.6.20)	Specifies additional requirements on an SMI-S Server that supports the management of Identities, including establishing Accounts, and defining User and Organizational entities and Groups of those entities.
CIM_Role (8.2.5.7.6.22)	
CIM_System (8.2.5.7.6.25)	
CIM_UserContact (8.2.5.7.6.27)	

8.2.5.7.6.1 CIM_Account

Class Mandatory: false

Table 557: SMI Referenced Properties/Methods for CIM_Account

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemName		string	Key
SystemCreationClassName		string	Key
Name		string	Key
CreationClassName		string	Key
UserID	CM	string	The users ID

8.2.5.7.6.2 CIM_AccountOnSystem

Class Mandatory: false

Table 558: SMI Referenced Properties/Methods for CIM_AccountOnSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	Key
PartComponent		CIM_Account	Key

8.2.5.7.6.3 CIM_AssignedIdentity

Class Mandatory: false

Table 559: SMI Referenced Properties/Methods for CIM_AssignedIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
IdentityInfo		CIM_Identity	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.7.6.4 CIM_AuthenticationService

Class Mandatory: false

Table 560: SMI Referenced Properties/Methods for CIM_AuthenticationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.7.6.5 CIM_ConcreteDependency

Class Mandatory: true

Table 561: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Key
Dependent		CIM_ManagedElement	Key

8.2.5.7.6.6 CIM_ConcreteIdentity

Class Mandatory: true

Table 562: SMI Referenced Properties/Methods for CIM_ConcreteIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemElement		CIM_ManagedElement	Key
SameElement		CIM_ManagedElement	Key

8.2.5.7.6.7 CIM_Credential

Class Mandatory: true

No specified properties or methods.

8.2.5.7.6.8 CIM_CredentialContext

Class Mandatory: false

Table 563: SMI Referenced Properties/Methods for CIM_CredentialContext

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementInContext		CIM_Credential	Key
ElementProvidingContext		CIM_ManagedElement	Key

8.2.5.7.6.9 CIM_ElementConformsToProfile

Class Mandatory: true

Table 564: SMI Referenced Properties/Methods for CIM_ElementConformsToProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ConformantStandard		CIM_RegisteredProfile	Key
ManagedElement		CIM_ManagedElement	Key

8.2.5.7.6.10 CIM_Group

Class Mandatory: false

Table 565: SMI Referenced Properties/Methods for CIM_Group

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name	C	string	Key
CommonName		string	The Name by which the Group is known

8.2.5.7.6.11 CIM_HostedAccessPoint

Class Mandatory: true

Table 566: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_ServiceAccessPoint	Key

8.2.5.7.6.12 CIM_HostedService

Class Mandatory: true

Table 567: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Key
Dependent		CIM_Service	Key

8.2.5.7.6.13 CIM_Identity

Class Mandatory: false

Table 568: SMI Referenced Properties/Methods for CIM_Identity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
CurrentlyAuthenticated		boolean	Currently trusted or not

8.2.5.7.6.14 CIM_IdentityContext

Class Mandatory: false

Table 569: SMI Referenced Properties/Methods for CIM_IdentityContext

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementProvidingContext		CIM_ManagedElement	Key
ElementInContext		CIM_Identity	Key

8.2.5.7.6.15 CIM_MemberOfCollection

Class Mandatory: false

Table 570: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	Key
Member		CIM_ManagedElement	Key

8.2.5.7.6.16 CIM_ReferencedProfile

Class Mandatory: true

Table 571: SMI Referenced Properties/Methods for CIM_ReferencedProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.7.6.17 CIM_RegisteredProfile

Class Mandatory: false

Table 572: SMI Referenced Properties/Methods for CIM_RegisteredProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	Parent subprofile

8.2.5.7.6.18 CIM_RegisteredSubProfile

Specifies additional requirements on an SMI-S Server when it is also a client of a 3rd party authentication service.
Class Mandatory: true

Table 573: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	This subprofile

8.2.5.7.6.19 CIM_RegisteredSubProfile

Specifies additional requirements on an SMI-S Server that is also a client of some other service that enforces security.
Class Mandatory: false

Table 574: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	SubProfile Name

8.2.5.7.6.20 CIM_RegisteredSubProfile

Specifies additional requirements on an SMI-S Server that supports the management of Identities, including establishing Accounts, and defining User and Organizational entities and Groups of those entities.
Class Mandatory: false

Table 575: SMI Referenced Properties/Methods for CIM_RegisteredSubProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Key
RegisteredOrganization	C	uint16	Indicate SNIA
RegisteredName	C	string	SubProfile Name

8.2.5.7.6.21 CIM_RemoteServiceAccessPoint

Class Mandatory: true

Table 576: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	Key
SystemName		string	Key
CreationClassName		string	Key
Name		string	Key

8.2.5.7.6.22 CIM_Role

Class Mandatory: false

Table 577: SMI Referenced Properties/Methods for CIM_Role

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name	C	string	Key

8.2.5.7.6.23 CIM_ServiceSAPDependency

Class Mandatory: true

Table 578: SMI Referenced Properties/Methods for CIM_ServiceSAPDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	Key
Dependent		CIM_Service	Key

8.2.5.7.6.24 CIM_SubProfileRequiresProfile

Class Mandatory: true

Table 579: SMI Referenced Properties/Methods for CIM_SubProfileRequiresProfile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_RegisteredSubProfile	Key
Antecedent		CIM_RegisteredProfile	Key

8.2.5.7.6.25 CIM_System

Class Mandatory: false

Table 580: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name		string	Key

8.2.5.7.6.26 CIM_System

Class Mandatory: true

Table 581: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key

Table 581: SMI Referenced Properties/Methods for CIM_System

Property	Flags	Type	Description & Notes
Name		string	Key

8.2.5.7.6.27 CIM_UserContact

Class Mandatory: false

Table 582: SMI Referenced Properties/Methods for CIM_UserContact

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Key
Name	C	string	Key
Surname		string	The Name by which the User is known to other users.
UserID	C	string	The Name by which the User is known to the System. Matches all Account or Person instances in the namespace with the same UserID. Changing here changes corresponding values on matching Person or Account instances.

8.2.5.7.7 Related Standards

Table 583: Related Standards for Security 3rd Party Authentication

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.6 Fabric Topology Profiles

8.2.6.1 Fabric Profile

8.2.6.1.1 Description

SANS and Fabrics as AdminDomains

A SAN and Fabric are represented in CIM by AdminDomain. A SAN contains one or more Fabrics, which are modeled as AdminDomains. The “containment” of Fabrics to SANs is through the association ContainedDomain. AdminDomain is sub-classed from System. This is significant because a SAN and a Fabric can be considered a group of components that operate together as a single system and should be/are managed as such. The relationship of the Fabrics in a SAN could be as redundant fabrics, interconnected (using the same or different transports/protocols), or not connected in any way. Even in the latter case where the Fabrics are disjoint, from an administrative perspective they may still be managed together applying common practices including naming across the Fabrics.

An AdminDomain in CIM is keyed by the property Name with an associated optional property NameFormat. Typically SANs are identified (“named”) administratively and precise naming conventions are left up to the implementation, which is then responsible for assuring that the names are unique within the discovery of known SANs that populate the same CIM Namespace.

For Fibre Channel Fabrics, the identifier (AdminDomain.Name) is the Fabric WWN that is the switch name of the principal switch. The AdminDomain for the Fibre Channel Fabric shall have a NameFormat of WWN.

Fabrics and Topology

A Fabric in CIM today minimally contains a ConnectivityCollection and its component systems. They are associated to the Fabric by the association Component. For the purposes of this discussion, it is assumed one models both.

ConnectivityCollection represents the foundation necessary for routing (and the reason it is defined in the Network model). A ConnectivityCollection groups a set of ProtocolEndpoints together that are able to communicate with each other directly. The ProtocolEndpoint is associated to the ConnectivityCollection by MemberOfCollection. A link is represented by the association ActiveCollection, which associates two ProtocolEndpoints, defined as a connection that is currently carrying traffic or is configured to carry traffic.

It is important at this point to clarify the relationship (or use) of the ProtocolEndpoint versus the use of FCPort (discussed later). A NetworkPort (from which FCPort is subclassed) is the device that is used to represent the logical aspects of the link and data layers. The ProtocolEndpoint is used to represent the higher network layers for routing. This is best understood when thinking about Ethernet and IP, but applies to fibre channel also. When two ProtocolEndpoints are capable of communicating, the association ActiveConnection is used to represent the capability to communicate and completes the picture of the topology.

One can ultimately represent multiple ConnectivityCollections (e.g., FC, IP (over FC), and IP (FC encapsulated in IP)) for the same fibre channel fabric.

Note that in modeling SANs, Fabrics, and ConnectivityCollections, a ConnectivityCollection does not require a Fabric, and a Fabric does not require a SAN. But a SAN requires a Fabric, and a Fabric (for the purposes of this profile) requires a ConnectivityCollection.

The minimum set of requirements for this profile is based on FC-GS.

Systems and NetworkPorts

As discussed in the previous section, a Port is associated to a device to represent the link layer. A NetworkPort is associated to the ProtocolEndpoint by DeviceSAPImplementation and “joins” the

System and Device model to the Network model. Instantiation of DeviceSAPImplementation, ProtocolEndpoint, and ActiveConnection is not necessary if the transceiver is not installed or the cable connecting the port to another port is not installed since the device is not capable of communicating.

Systems, or in this case ComputerSystem, represent the fabric elements that contain Ports. These are typically Hosts, Switches and Storage Systems. In Fibre Channel, these are called Platforms and Interconnect Elements. The property Dedicated in ComputerSystem allows these fabric elements to be identified. For a host, Dedicated is set to “Not Dedicated”, for a switch, Dedicated is set to “Switch”, and for a storage system, Dedicated is set to “Storage”. The Ports on a System are associated by SystemDevice.

Discovery from the viewpoint of the fabric includes the end device, but often times the information available is minimal or not available. In the case of Fibre Channel, this occurs if the platform database is not populated. If this is the case, then discovery cannot tell whether a Fibre Channel Node is contained within the same platform or not. When this occurs, ComputerSystem is not instantiated and the LogicalPortGroup representing the Node and the FCPort are associated to the AdminDomain representing the Fabric.

The instrumentation needs to respond to physical fabric changes by adding or removing Logical elements to the AdminDomain. Adding an element to the fabric is straightforward, however it is not always clear when an element has been removed. The device may have been reset, or temporarily shut down, in which case it would be an element in the fabric with an “unknown” status. The lifetime of objects that can no longer be discovered is implementation specific.

If the instrumentation is unable to determine the type of platform discovered (defined in FC-GS), then the agent shall set the ComputerSystem.Dedicated property to “Unknown”.

Additional identification information about ComputerSystem (e.g., DomainID) is placed in OtherIdentifyInfo property.

Figure 90: Fabric Instance Diagram

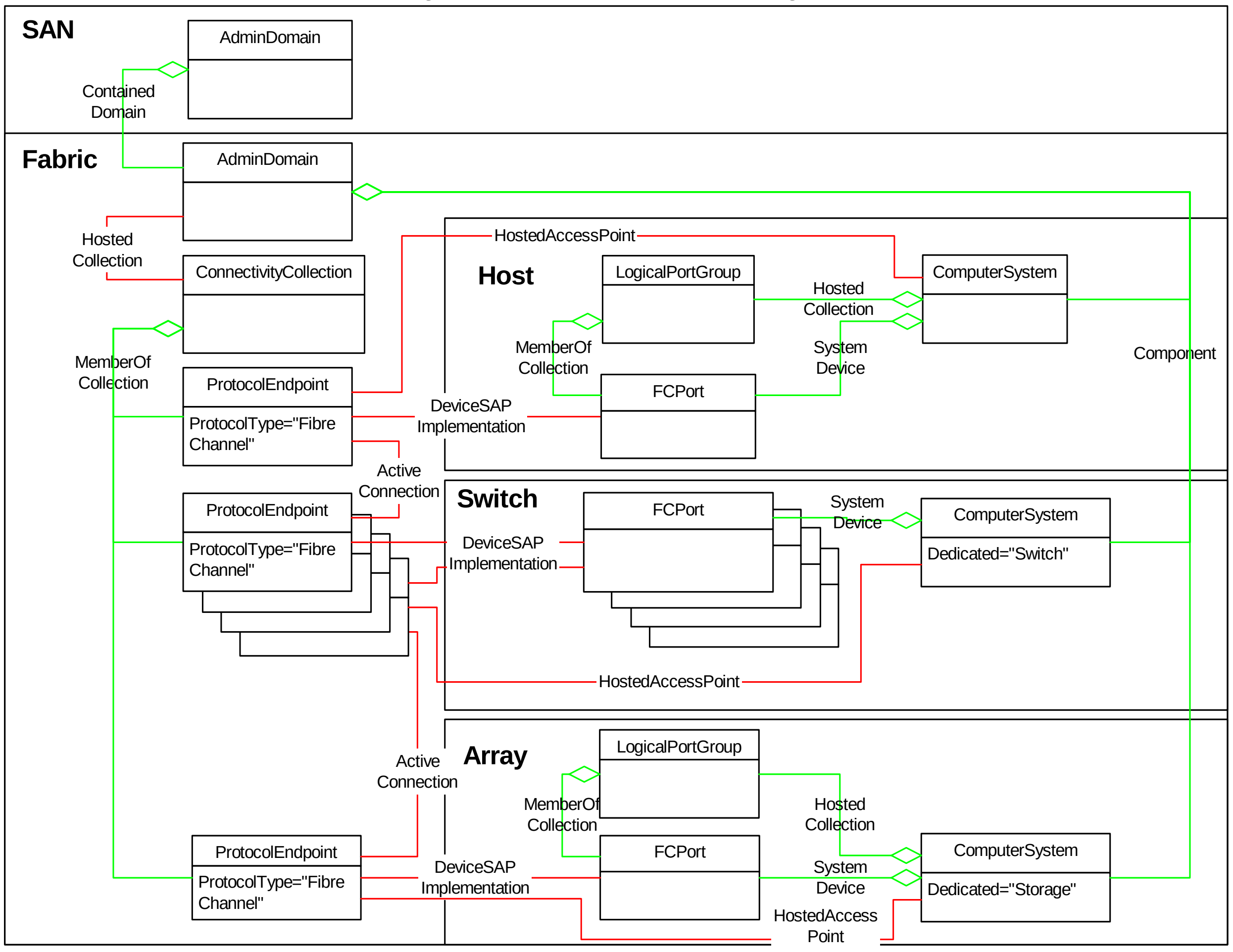


Figure 91: Zoning Instance Diagram (AdminDomain)

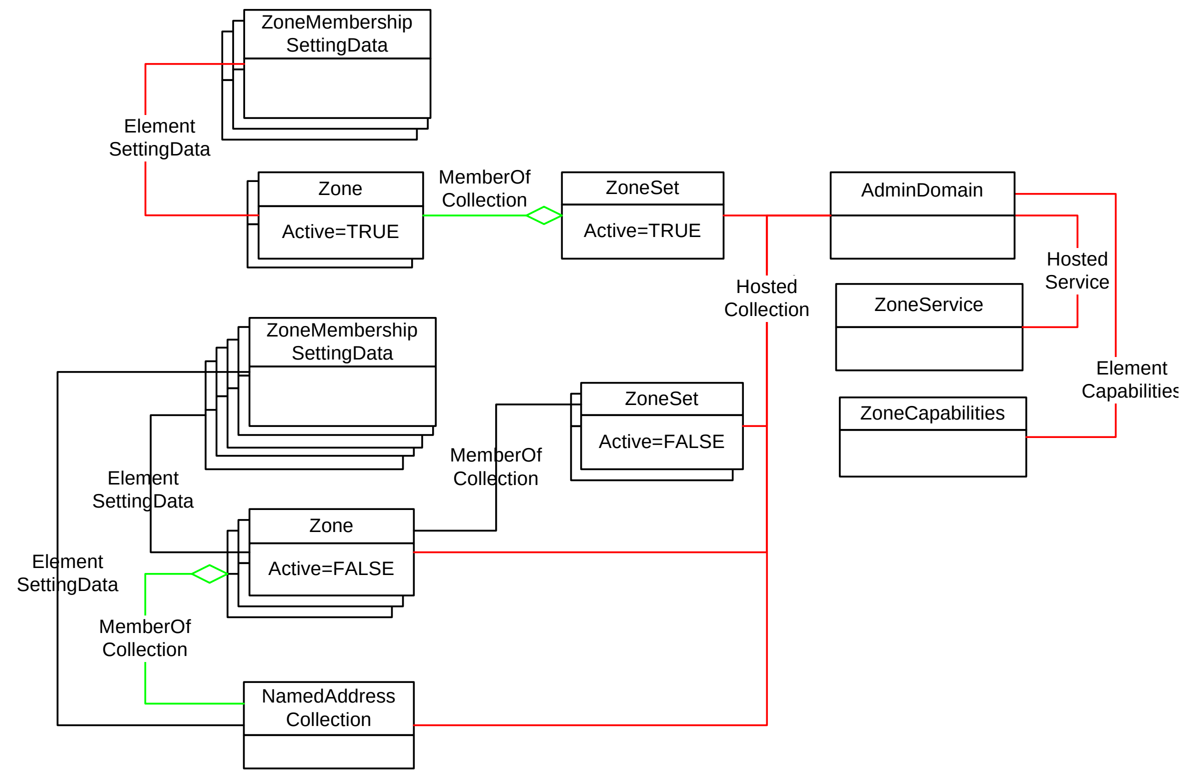
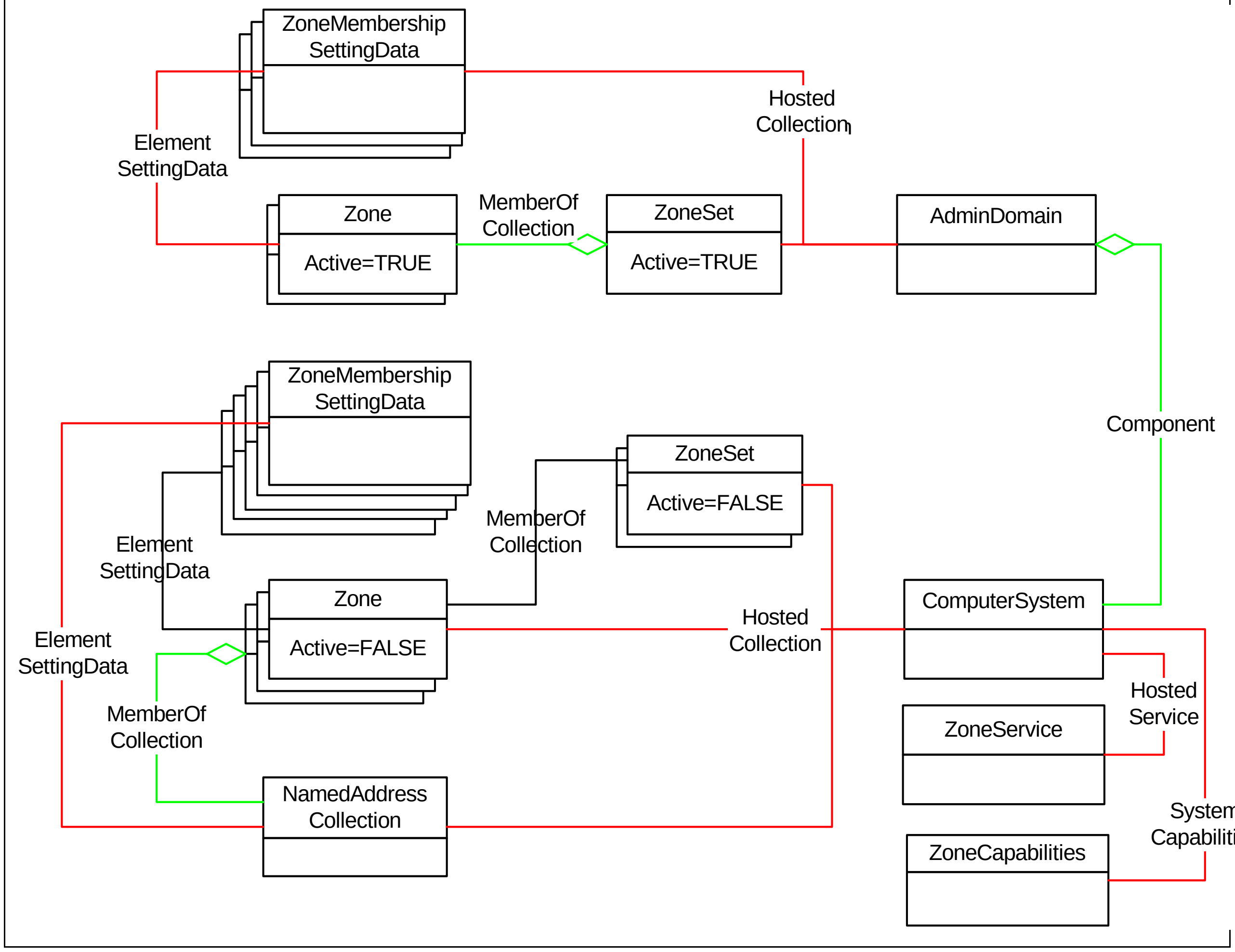


Figure 92: Zoning Instance Diagram (ComputerSystem)



Zoning

The zoning model is based on ANSI FC-GS-4. This model represents the management model for defining Zone Sets, Zones, and Zone Members and “activation” of a Zone Set for a fabric. In the following discussion it may be helpful to also define the following:

Active ZoneSet: the Zone Set currently enforced by the Fabric.

Zone Set Database: The database of the Zone Sets not enforced by the Fabric. Referred to in this document as the Inactive Zone Sets.

Zoning Definitions: a generic term used to indicate both the above concepts.

The zoning model refers to a Zone Set as ZoneSet, a Zone as Zone, ZoneAlias as a NamedAddressCollection, and Zone Member as ZoneMembershipSettingData. ZoneSets shall only contain Zones associated by MemberOfCollection. Zones shall only contain ZoneMembershipSettingData associated by ElementSettingData or NamedAddressCollections associated by MemberOfCollection. For more information with regards to NamedAddressCollection, see 8.2.6.2, "Enhanced Zoning Subprofile".

The class ZoneMembershipSettingData has two properties that indicate how the device was identified to be “zoned”. They are ConnectivityMemberType (e.g., PermanentAddress for WWN, NetworkAddress for FCID, etc.) and ConnectivityMemberID which contains the actual device identifier.

The Active Zone Set, defined by an instance of ZoneSet with the Active property set to TRUE, shall only be hosted on the AdminDomain representing the Fabric. The Inactive Zone Sets, defined by an instance of ZoneSet with the Active property set to FALSE, shall be hosted on either the AdminDomain representing the Fabric as shown in the Figure 91: "Zoning Instance Diagram (AdminDomain)" or the ComputerSystem representing the switch as shown in the Figure 92: "Zoning Instance Diagram (ComputerSystem)". It is allowed to have no ZoneSets (active or inactive), only an active ZoneSet, only an inactive ZoneSet(s), or both an inactive ZoneSet(s) and an active ZoneSet.

The ZoneService and ZoneCapabilities are also associated to the same System (AdminDomain or ComputerSystem) as the Inactive Zone Sets using the association HostedService or ElementCapabilities, respectively.

ZoneService provides the configuration methods to control create ZoneSets, Zones, Zone Aliases, and Zone Members, as well as activation of the Zone Set. This service and its methods are described in the 8.2.6.2, "Enhanced Zoning Subprofile".

8.2.6.1.2 Health and Fault Management

The following classes report possible Health and Fault information through LifeCycle indications:

- ComputerSystem,
- FCPort/

These LifeCycle indications are more fully described in 8.2.6.1.8, "CIM Server Requirements".

Also in Table 587, “CIM Server Requirements for Fabric” are a list of AlertIndications which may also be indicators for Health and Fault Management.

8.2.6.1.3 Cascading Considerations

None

8.2.6.1.4 Supported Subprofiles and Package

Table 584: Supported Subprofiles for Fabric

Registered Subprofile Names	Mandatory	Version
Zone Control	No	1.1.0
Enhanced Zoning and Enhanced Zoning Control	No	1.1.0
FDMI	No	1.1.0
Fabric Path Performance	No	1.1.0

8.2.6.1.5 Methods of this Profile

None

8.2.6.1.6 Client Considerations and Recipes

<u>**Fabric Identifier**</u>

The client needs to consider that the fabric identifier is not durable but is correlatable and may change over time. See 6.2.4, "Correlatable and Durable Names".

FCPort OperationalStatus

OperationalStatus is the property to indicate status and state for the FCPort. The FCPort instance has one of the following Operational Statuses.

Table 585: Port OperationalStatus

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

ComputerSystem OperationalStatus

OperationalStatus is the property to indicate status and state for the ComputerSystem. The ComputerSystem instance has one of the following Operational Statuses and possibly one of the Subsidiary statuses.

Table 586: OperationalStatus for ComputerSystem

Operational Status	Possible Subsidiary Operational Status	Description
OK		The system has a good status
OK	Stressed	The system is stressed, for example the temperature is over limit or there is too much IO in progress
OK	Predictive Failure	The system will probably will fail sometime soon
Degraded		The system is operational but not at 100% redundancy. A component has suffered a failure or something is running slow
Error		An error has occurred causing the system to stop. This error may be recoverable with operator intervention.
Error	Non-recoverable error	A severe error has occurred. Operator intervention is unlikely to fix it
Error	Supporting entity in error	A modeled element has failed
InService		Switch is in Self Test.
No contact		The provider knows about the array but has not talked to it since last reboot
Lost communication		The provider used to be able to communicate with the array, but has now lost contact.
Starting		The system is starting up
Stopping		The system is shutting down.
Stopped		The data path is OK but shut down, the management channel is still working.

8.2.6.1.6.1 Discover the Fabric Topology

```
// This recipe describes how to build a topology graph of a fabric.
```

```
//
// 1. Identifies all the Switches and adds their objects paths and the
// object paths of the FC Ports belonging to these Switches to the $nodes
// array
//
// 2. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Switch and a FC Port), setting its
// GroupComponent and PartComponent. Adds the object path of the
// Association to the $links array
//
// 3. Creates a map of all connected FC Ports (i.e., belonging to Switches
// that are ISL’d together and to Host HBAs and Storage System Front End
// Controllers)
//
// In this map, the FC Ports (i.e., the ones that are connected) are
// cross-connected.
//
// e.g., For a pair of FC Ports, one belonging to a Switch and the other
// belonging to a Host (HBA), the map indexed by the Switch Port WWN returns
// the Host (HBA) FC Port object path and the map indexed by the Host (HBA)
// FC Port WWN returns the Switch FC Port object path.
//
// Similar relationship exists between the pairs of FC Ports where one
// belongs to a Switch and the other belonging belongs to a Storage System
// Front End Controller and for FC Ports each of which belongs to a Switch.
//
// 4. Identifies all the Hosts and adds their objects paths to the $nodes
// array. Note that the object paths of the FC Ports (HBA Ports) belonging
// to these Hosts are already added to the $nodes array in step-3.
//
// 5. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Host and a FC Port), setting its
// GroupComponent and PartComponent. Adds the object path of the Association
// to the $links array.
//
// 6. Identifies all the Storage Systems and adds their objects paths to the
// $nodes array.
// Note that the object paths of the FC Ports (i.e., Front End Controller
// FC Ports) belonging to these Storage Systems are already added to the
// $nodes array in step-3.
//
// 7. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Storage System and a FC Port), setting
// its GroupComponent and PartComponent. Adds the object path of the
// Association to the $links array.

// DESCRIPTION
// Create a map of how elements in a SAN are connected together via
// Fibre-ChannelFC ports.
//
// The map is built in array $attachedFcPorts->[], where the index is a
// WWN of any device port on the SAN, and the value at that index is
// the object path of the connected Switch or HBA or Storage System FC port.
//
// First find all the switches in a SAN. Get all the FC Ports for each
// switch and get the Attached FC Ports for each Switch FC Port. Save these
// device FC ports in the map described above.

// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1. All agents/namespaces supporting Fabric Profile previously identified
// using SLP. Do this for each CIMOM supporting Fabric Profile

switches[] = enumerateInstances(“CIM_ComputerSystem”, true, false, true, true,
null)
for #i in $switches[]
{
    if (!contains(5, $switches[#i].Dedicated))
        continue

    // only process switches, not other computer systems

    // Add the switch to the $nodes array

    $nodes.addIfNotAlreadyAdded ($switches[#i].getObjectPath();

    // Get all the SystemDevice associations between this switch and its
    // FC Ports

    $sysDevAssoc[] = ReferenceNames($switches[#i],
                                “CIM_FCPort”,
                                “GroupComponent”);

    // Add these associations to the $links array

    for #a in $sysDevAssoc->[]
    $links.addIfNotAlreadyAdded ($sysDevAssoc->[#a];

    $fcPorts->[] = AssociatorNames(
        $switches[#i].getObjectPath(),
        “CIM_SystemDevice”,
        “CIM_FCPort”,
        “GroupComponent”,
        “PartComponent”)
    for #j in $fcPorts->[]
    {
```

```
// Add the FC Port in $nodes array

$nodes.addIfNotAlreadyAdded (fcPorts->[#j];

$protocolEndpoints->[] = AssociatorNames(
    fcPorts->[#j],
    “CIM_DeviceSAPImplementation”,
    “CIM_ProtocolEndpoint”,
    “Antecedent”,
“Dependent”);

// NOTE - It is possible for this collection to be empty (i.e., ports
// that are not connected). It is possible for this collection to
// have more than one element (loops attached to a switch port is the
// most common example).

if ($protocolEndpoints->[].length == 0)
    continue

// Add the Protocol End Point to the nodes array.
// Currently this recipe is designed to only save one
// ProtocolEndpoint.

$nodes.addIfNotAlreadyAdded (protocolEndpoints[0]);

// Add the associations between the fcPort and the Protocol end point
// to the links array

$devSAPImplassoc[]  = ReferenceNames($fcPorts->[#j],
                             “CIM_ProtocolEndpoint”,
                             null);
for #a in $devSAPImplassoc->[]
    $links.addIfNotAlreadyAdded ($devSAPImplassoc->[#a];

$attachedProtocolEndpoints->[] = AssociatorNames(
    $protocolEndpoints->[0],
    “CIM_ActiveConnection”,
    “CIM_ProtocolEndpoint”,
    null, null)

// Add the Attached Protocol End Point to the nodes array

$nodes.addIfNotAlreadyAdded (attachedProtocolEndpoints->[0]);

// Add the associations between the Protocol end point and the
// Attached protocol endpoint to the links array
```

```
        $actConnassoc[]  = ReferenceNames($protocolEndpoint->[#0],
                                 “CIM_ActiveConnection”,
                                  null);
        for #a in $actConnassoc->[]
            $links.addIfNotAlreadyAdded ($actConnassoc->[#a];

        // NOTE: role & resultRole are null as the direction of the
        // association is not dictated by the specification

        // $attachedFcPort is either a device FC port or an ISL’d switch FC
        // port from another switch. We store this result is stored (i.e.,
        // which device FC Port is connected // to which switch FC Port) in
        // a suitable data structure for subsequent correlation to ports
        // discovered on devices.

        for #k in $attachedProtocolEndpoints->[]
        {
            $attachedFcPorts->[] = Associators(
                $attachedProtocolEndpoints->[#k],
                “CIM_DeviceSAPImplementation”,
                “CIM_FCPort”,
                “Dependent”,
                “Antecedent”,
                false,
                false,
                [“PermanentAddress”])
           $attachedFcPort = $attachedFcPorts[0] // Exactly one member guaranteed
by model

            // Add the attached FC Port to the $nodes array
            if $attachedFcPort != null
                $nodes.addIfNotAlreadyAdded ($attachedFcPort);
        }
    }
}
```

```
Determine the active Zone Set in a SAN
// DESCRIPTION
// Traverse from the fabric to all zone sets, looking for
// the active zone set
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The fabric of interest (an AdminDomain) has been previously
//    identified and defined in the $Fabric-> variable

$ZoneSets[] = Associators($Fabric->, “CIM_HostedCollection”, “CIM_ZoneSet”, null,
null, false, false, null)

for #i in $ZoneSets[] {
    if ($ZoneSet[#i].Active) {
        // <found active ZoneSet>
        // NOTE - there can be only one active ZoneSet in a fabric, though there
may be none
        break
    }
}
```

8.2.6.1.7 Registered Name and Version

Fabric version 1.1.0

8.2.6.1.8 CIM Server Requirements

Table 587: CIM Server Requirements for Fabric

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.1.9 CIM Elements

Table 588: CIM Elements for Fabric

Element Name	Description
Mandatory Classes	
CIM_ActiveConnection (8.2.6.1.9.1)	The association between ProtocolEndpoints representing the links between devices.
CIM_AdminDomain (8.2.6.1.9.2)	AdminDomain representing the SAN
CIM_AdminDomain (8.2.6.1.9.3)	AdminDomain representing the Fabric.
CIM_Component (8.2.6.1.9.4)	Aggregates Hosts, Arrays and Switches in the AdminDomain that represents the Fabric
CIM_ComputerSystem (8.2.6.1.9.5)	The ComputerSystem representing the Interconnect Element (e.g. a switch) or Platform (e.g. Host and Array).
CIM_ComputerSystem (8.2.6.1.9.6)	The ComputerSystem representing the Platform (e.g. Host and Array).
CIM_ConnectivityCollection (8.2.6.1.9.7)	Collects the ProtocolEndpoints of the fabric.
CIM_ContainedDomain (8.2.6.1.9.8)	Associates a Fabric to a SAN
CIM_DeviceSAPImplementation (8.2.6.1.9.9)	Associates the FCPort to the ProtocolEndpoint
CIM_ElementCapabilities (8.2.6.1.9.10)	Associates ZoneCapabilities to a System
CIM_ElementSettingData (8.2.6.1.9.11)	Associates ZoneMembershipSettingData to the Zone or NamedAddressCollection representing the ZoneAlias.
CIM_FCPort (8.2.6.1.9.12)	Fibre Channel Port for Switch
CIM_FCPort (8.2.6.1.9.13)	Fibre Channel Port for Devices
CIM_HostedAccessPoint (8.2.6.1.9.14)	Associates the ProtocolEndpoint to the hosting System
CIM_HostedCollection (8.2.6.1.9.15)	Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System.
CIM_HostedCollection (8.2.6.1.9.16)	Associates the ConnectivityCollection to the AdminDomain representing the Fabric.
CIM_HostedCollection (8.2.6.1.9.17)	Associates the ZoneSets, Zones, and NamedAddressCollections representing the ZoneAliases to the hosting System (either the AdminDomain representing the Fabric or the ComputerSystem representing the switch).
CIM_LogicalPortGroup (8.2.6.1.9.18)	Fibre Channel Node
CIM_MemberOfCollection (8.2.6.1.9.19)	Associates FCPort to the LogicalPortGroup
CIM_MemberOfCollection (8.2.6.1.9.20)	Associates ProtocolEndpoints to the ConnectivityCollection
CIM_MemberOfCollection (8.2.6.1.9.21)	Associates ZoneMembershipSettingData and NamedAddressCollections to the Zone
CIM_MemberOfCollection (8.2.6.1.9.22)	Associates Zones to the ZoneSets
CIM_ProtocolEndpoint (8.2.6.1.9.23)	The endpoint of a link (ActiveConnection).
CIM_SystemDevice (8.2.6.1.9.24)	Associates the FCPort to the ComputerSystem
CIM_Zone (8.2.6.1.9.25)	The active Zones being enforced by the Fabric.
CIM_Zone (8.2.6.1.9.26)	The inactive Zones being enforced by the Fabric.

Table 588: CIM Elements for Fabric

Element Name	Description
CIM_ZoneCapabilities (8.2.6.1.9.27)	The Zoning Capabilities of the ZoneService of the Fabric (or Switch).
CIM_ZoneMembershipSettingData (8.2.6.1.9.28)	Defines the ZoneMember
CIM_ZoneSet (8.2.6.1.9.29)	The active ZoneSets being enforced by the Fabric.
CIM_ZoneSet (8.2.6.1.9.30)	The inactive ZoneSet
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Creation of a ComputerSystem instance.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Deletion of a ComputerSystem instance.
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort	Creation of a FC Port instance. Indications for FCPort creation should not be generated as a result of the ComputerSystem being created.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort	Deletion of a FC Port instance. Indications for FCPort deletion should not be generated as a result of the ComputerSystem being deleted.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Modification of OperationalStatus of a FC Port instance.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.CIM_FCPort::OperationalStatus <> PreviousInstance.CIM_FCPort::OperationalStatus	CQL - Modification of OperationalStatus of a FC Port instance.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.Operationalstatus <> PreviousInstance.Operationalstatus	Deprecated WQL - Modification of OperationalStatus of a ComputerSystem instance.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::Operationalstatus <> PreviousInstance.CIM_ComputerSystem::Operationalstatus	CQL - Modification of OperationalStatus of a ComputerSystem instance.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity='SNIA' and MessageID='FC1'	Modification of Zone Database.
SELECT * FROM CIM_AlertIndication WHERE OwningEntity='SNIA' and MessageID='FC2'	ZoneSet Activated.

8.2.6.1.9.1 CIM_ActiveConnection

The association between ProtocolEndpoints representing the links between devices (including ISLs). For loops, multiple ActiveConnections are instantiated as one to many relationships.

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

Class Mandatory: true

Table 589: SMI Referenced Properties/Methods for CIM_ActiveConnection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	The reference to the ProtocolEndpoint for one end of the link
Dependent		CIM_ServiceAccessPoint	The reference to the ProtocolEndpoint for the other end of the link

8.2.6.1.9.2 CIM_AdminDomain

AdminDomain representing the SAN

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 590: SMI Referenced Properties/Methods for CIM_AdminDomain (SAN)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name		string	An arbitrary name (implementation dependent)
NameFormat		string	Dependent on the arbitrary name chosen.

8.2.6.1.9.3 CIM_AdminDomain

AdminDomain representing the fabric. This is a logical entity and can represent virtual fabrics.

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The Fabric Name follows the requirements in 6.2.4.5.3
Class Mandatory: true

Table 591: SMI Referenced Properties/Methods for CIM_AdminDomain (Fabric)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name	C	string	WWN of Fabric
NameFormat		string	"WWN"

8.2.6.1.9.4 CIM_Component

Aggregates Hosts, Arrays and Switches in the AdminDomain that represents the Fabric

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

Class Mandatory: true

Table 592: SMI Referenced Properties/Methods for CIM_Component

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ManagedElement	The reference to the AdminDomain representing the Fabric
PartComponent		CIM_ManagedElement	The reference to the ComputerSystem representing the Host, Array, or Switch.

8.2.6.1.9.5 CIM_ComputerSystem

The ComputerSystem representing the Interconnect Element (e.g. a switch) or Platform (e.g. Host and Array).

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The Computer System Name follows the requirements in 6.2.4.5.3
Class Mandatory: true

Table 593: SMI Referenced Properties/Methods for CIM_ComputerSystem (Fibre Channel Switch)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name	C	string	The Switch WWN.
ElementName		string	The Switch Symbolic Name.
NameFormat		string	
OperationalStatus		uint16[]	One of the defined values shall be present in the array value.
OtherIdentifyingInfo		string[]	DomainID stored in decimal format
Dedicated		uint16[]	"Switch"
IdentifyingDescriptions		string[]	Identifying descriptor for OtherIdentifyingInfo. The value "DomainID" is in IdentifyingDescriptions and in the corresponding index for OtherIdentifyingInfo the DomainID is placed.

8.2.6.1.9.6 CIM_ComputerSystem

The ComputerSystem representing the Platform (e.g. Host and Array). This class is typically instantiated if the end device has populated the Fibre Channel Platform Database or FDMI.

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The Computer System Name follows the requirements in 6.2.4.5.3

Class Mandatory: true

Table 594: SMI Referenced Properties/Methods for CIM_ComputerSystem (Fibre Channel Platform)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name	C	string	The Platform Name or FDMI Host Name.
ElementName		string	The Platform Label.
NameFormat		string	
Dedicated		uint16[]	For a FC-GS Platform Type of Host, "Not Dedicated" (0); for storage subsystems, "Storage" (3); for Gateway, "Gateway" (20); for Router, "Router" (4); for Bridge, "Bridge/Extender" (19); for Platform Type of Other, "Other" (2).

8.2.6.1.9.7 CIM_ConnectivityCollection

Collects the ProtocolEndpoints of the fabric.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 595: SMI Referenced Properties/Methods for CIM_ConnectivityCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
Optional Properties/Methods			
ElementName		string	Not required, can be the Fabric WWN.

8.2.6.1.9.8 CIM_ContainedDomain

Associates one or more Fabrics to a SAN.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 596: SMI Referenced Properties/Methods for CIM_ContainedDomain

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_AdminDomain	The reference to the AdminDomain representing the SAN
PartComponent		CIM_AdminDomain	The reference to the AdminDomain representing the Fabric

8.2.6.1.9.9 CIM_DeviceSAPImplementation

Associates the FCPort to the ProtocolEndpoint

Created By : Extrinsic(s):
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 597: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalDevice	The reference to the FCPort
Dependent		CIM_ServiceAccessPoint	The reference to the ProtocolEndpoint

8.2.6.1.9.10 CIM_ElementCapabilities

Associates the ZoneCapabilities to a System. The system normally is the AdminDomain representing the Fabric, but in some cases where the Zone Database is not a fabric entity, it may be hosted on a ComputerSystem representing the Switch.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 598: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to a System representing either an AdminDomain or ComputerSystem
Capabilities		CIM_Capabilities	The reference to ZoneCapabilities

8.2.6.1.9.11 CIM_ElementSettingData

Associates ZoneMembershipSettingData to the Zone or NamedAddressCollection representing the ZoneAlias.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 599: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the Zone or ZoneAlias
SettingData		CIM_SettingData	The reference to ZoneMembershipSettingData

8.2.6.1.9.12 CIM_FCPort

Fibre Channel Port for Switch

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The PermanentAddress follows the requirements in 6.2.4.5.2
Class Mandatory: true

Table 600: SMI Referenced Properties/Methods for CIM_FCPort (Switch)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	Name of Class
DeviceID		string	Opaque
ElementName		string	Port Symbolic Name if available. Other-wise NULL. If the underlying implemen-tation includes characters that are illegal in CIM strings, then truncate before the first of those characters.
PermanentAddress	CD	string	Fibre Channel Port WWN
OperationalStatus		uint16[]	One of the defined values shall be present in the array value.
PortType		uint16	The specific port type currently enabled (from FC-GS Port.Type)
LinkTechnology		uint16	"FC"
Optional Properties/Methods			
Speed		uint64	Speed of zero represents a link not established. 1Gb is 1062500000 bps 2Gb is 2125000000 bps 4Gb is 4250000000 bps 10Gb single channel variants are 10518750000 bps 10Gb four channel variants are 12750000000 bps This is the raw bit rate.

8.2.6.1.9.13 CIM_FCPort

Fibre Channel Port for non-Switches (Non-Dedicated, Storage, Router, Bridge/Extender)

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The PermanentAddress follows the requirements in 6.2.4.5.2

Class Mandatory: true

Table 601: SMI Referenced Properties/Methods for CIM_FCPort (Devices)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	Name of Class
DeviceID		string	Opaque
OperationalStatus		uint16[]	One of the defined values shall be present in the array value.
PortType		uint16	The specific port type currently enabled (from FC-GS Port.Type)
LinkTechnology		uint16	"FC"
Optional Properties/Methods			
ElementName		string	Port Symbolic Name if available. Other-wise NULL. If the underlying implemen-tation includes characters that are illegal in CIM strings, then truncate before the first of those characters.
PermanentAddress	CD	string	Fibre Channel Port WWN. Expressed as 16 unseparated upper case hex dig-its (see Table 4 for more information about formats).
NetworkAddresses	C	string[]	Fibre Channel ID (FCID). Expressed as 8 unseparated upper case hex digits (see Table 4 for more information about formats).
SupportedFC4Types		uint16[]	An array of integers indicating the Fibre Channel FC-4 protocols supported
SupportedCOS		uint16[]	An array of integers indicating the Fibre Channel Classes of Service that are supported.

8.2.6.1.9.14 CIM_HostedAccessPoint

Associates the ProtocolEndpoint to the hosting System. The hosting System is either a ComputerSystem for the Switch or Platform or to the AdminDomain for those systems not registered in the Platform Database or discovered through FDMI.

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

Class Mandatory: true

Table 602: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Reference to the System
Dependent		CIM_ServiceAccessPoint	Reference to the ProtocolEndpoint

8.2.6.1.9.15 CIM_HostedCollection

Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System. The hosting System is either a ComputerSystem for the Platform or the AdminDomain for those systems not registered in the Platform Database or discovered through FDMI.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 603: SMI Referenced Properties/Methods for CIM_HostedCollection (LogicalPortGroup)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System
Dependent		CIM_SystemSpecificCollection	The reference to the LogicalPortGroup (Fibre Channel Node)

8.2.6.1.9.16 CIM_HostedCollection

Associates the ConnectivityCollection to the AdminDomain representing the Fabric.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 604: SMI Referenced Properties/Methods for CIM_HostedCollection (ConnectivityCollection)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the AdminDomain representing the Fabric
Dependent		CIM_SystemSpecificCollection	The reference to the ConnectivityCollection

8.2.6.1.9.17 CIM_HostedCollection

Associates the ZoneSets, Zones, and NamedAddressCollections representing the ZoneAliases to the hosting System (either the AdminDomain representing the Fabric or the ComputerSystem representing the switch).

Created By : External
Modified By : Static

Deleted By : External
Class Mandatory: true

Table 605: SMI Referenced Properties/Methods for CIM_HostedCollection (Zone, ZoneSet, and ZoneAlias)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System (AdminDomain representing the Fabric or the ComputerSystem representing the Switch)
Dependent		CIM_SystemSpecificCollection	The reference to the SystemSpecificCollection (ZoneSets, Zones, or NamedAddressCollection)

8.2.6.1.9.18 CIM_LogicalPortGroup

Represents the Fibre Channel Node. Associated to the host system by the HostedCollection Association. The hosting System is either a ComputerSystem representing the Platform or the AdminDomain representing the fabric in the case for those systems not registered in the Platform Database or discovered through FDMI (but available through the Name Server/Management Server).

Created By : External
Modified By : Static
Deleted By : External
Standard Names: The Name follows the requirements in 6.2.4.5.2
Class Mandatory: true

Table 606: SMI Referenced Properties/Methods for CIM_LogicalPortGroup (Fibre Channel Node)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
Name	CD	string	Fibre Channel Node WWN
NameFormat		string	"WWN"
ElementName	N	string	Node Symbolic Name if available. Otherwise NULL. If the underlying implementation includes characters that are illegal in CIM strings, then truncate before the first of those characters.

8.2.6.1.9.19 CIM_MemberOfCollection

Associates FCPort to the LogicalPortGroup

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

Class Mandatory: true

Table 607: SMI Referenced Properties/Methods for CIM_MemberOfCollection (FCPort to LogicalPortGroup)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the LogicalPortGroup representing the Fibre Channel Node
Member		CIM_ManagedElement	The reference to FCPort.

8.2.6.1.9.20 CIM_MemberOfCollection

Associates ProtocolEndpoints to the ConnectivityCollection

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 608: SMI Referenced Properties/Methods for CIM_MemberOfCollection (ProtocolEndpoint to ConnectivityCollection)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the ConnectivityCollection
Member		CIM_ManagedElement	The reference to ProtocolEndpoint

8.2.6.1.9.21 CIM_MemberOfCollection

Associates ZoneMembershipSettingData and NamedAddressCollections (Zone Alias) which are Zone Members to the Zone

Created By : CreateInstanceExtrinsic(s): AddZoneMembershipSettingData
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: true

Table 609: SMI Referenced Properties/Methods for CIM_MemberOfCollection (ZoneAlias and ZoneMember to Zone)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to Zone
Member		CIM_ManagedElement	The reference to either the ZoneMembershipSettingData or LogicalPortGroup

8.2.6.1.9.22 CIM_MemberOfCollection

Associates Zones to the ZoneSets

Created By : CreateInstanceExtrinsic(s): AddZone
Modified By : Static
Deleted By : DeleteInstance

Class Mandatory: true

Table 610: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Zone to ZoneSet)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the ZoneSet
Member		CIM_ManagedElement	The reference to the Zone

8.2.6.1.9.23 CIM_ProtocolEndpoint

The endpoint of a link (ActiveConnection). ProtocolEndpoint shall be implemented when an ActiveConnection exists. It may be implemented if no ActiveConnections exist.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 611: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	Name of Class
Name	CD	string	The Fibre Channel Port WWN.
NameFormat		string	"WWN"
ProtocolIFType		uint16	"Fibre Channel"

8.2.6.1.9.24 CIM_SystemDevice

Associates the FCPort to the ComputerSystem

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 612: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The reference to the ComputerSystem
PartComponent		CIM_LogicalDevice	The reference to the FCPort

8.2.6.1.9.25 CIM_Zone

The active Zones being enforced by the Fabric.

Created By : Extrinsic(s): ActivateZoneSet
Modified By : Static
Deleted By : External

Class Mandatory: true

Table 613: SMI Referenced Properties/Methods for CIM_Zone (Active)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	The Zone Name
ZoneType		uint16	The Zone Type
Active		boolean	Must be TRUE. Indicates that this ZoneSet is active.

8.2.6.1.9.26 CIM_Zone

The inactive Zones being enforced by the Fabric.

Created By : Extrinsic(s): CreateZone
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: true

Table 614: SMI Referenced Properties/Methods for CIM_Zone (Inactive)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	The Zone Name
ZoneType		uint16	The Zone Type
Active		boolean	Must be FALSE. Indicates that this ZoneSet is inactive.

8.2.6.1.9.27 CIM_ZoneCapabilities

The Zoning Capabilities of the ZoneService of the Fabric (or Switch).

ZoneCapabilities exposes the capabilities of the AdminDomain representing the Fabric for active zoning and the capabilities of the ComputerSystem representing the Switch or AdminDomain representing the Fabric for Zone Set Database.

If a ZoneCapability property is not applicable or does not explicitly exists (e.g. the capability is limited only by a memory size), the property is NULL.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 615: SMI Referenced Properties/Methods for CIM_ZoneCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque

Table 615: SMI Referenced Properties/Methods for CIM_ZoneCapabilities

Property	Flags	Type	Description & Notes
ZoneNameMaxLen		uint32	The maximum length for the name of a ZoneAlias (NamedAddressCollection.ElementName), Zone (Zone.ElementName) or ZoneSet (ZoneSet.ElementName) the Fabric (or Switch) are capable of supporting.
ZoneNameFormat		uint16	The name format of a ZoneAlias (NamedAddressCollection.ElementName), Zone (Zone.ElementName) or ZoneSet (ZoneSet.ElementName) supported by either the Fabric (or the Switch)
SupportedConnectivityMemberTypes		uint16[]	An array containing the supported connectivity member types supported which include Permanent Address (WWN), Switch Port ID (Domain:Port in base10),Network Address (FCID), Logical Port Group (Node WWN).
Optional Properties/Methods			
MaxNumZoneSets		uint32	The maximum number of ZoneSets in the Zone Set Database. NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.
MaxNumZone		uint32	The maximum number of Zones in the Zone Set Database. NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.
MaxNumZoneMembers		uint32	The maximum number of ZoneMembers in the Zone Set Database . All ZoneMembers included in both Zones and ZoneAliases are counted, while the same ZoneMember included in multiple Zones or ZoneAliases is counted only once. NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.
MaxNumZoneAliases		uint32	The maximum number of ZoneAliases in the Zone Set Database NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.
MaxNumZonesPerZoneSet		uint32	The maximum number of Zones per ZoneSet. NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.

Table 615: SMI Referenced Properties/Methods for CIM_ZoneCapabilities

Property	Flags	Type	Description & Notes
MaxNumZoneSets		uint32	The maximum number of ZoneSets in the Zone Set Database. NULL should be returned in such cases when the property is not applicable or the number is not limited explicitly.

8.2.6.1.9.28 CIM_ZoneMembershipSettingData

Defines the ZoneMember

Created By : Extrinsic(s): AddZoneMemberSettingData
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: true

Table 616: SMI Referenced Properties/Methods for CIM_ZoneMembershipSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ConnectivityMemberType		uint16	Permanent Address (WWN), Switch Port ID (Domain:Port in base10),Network Address (FCID).
ConnectivityMemberID	C	string	The value of the WWN, Domain/Port, or FCID.

8.2.6.1.9.29 CIM_ZoneSet

The active ZoneSet being enforced by the Fabric.

Created By : Extrinsic(s): ActivateZoneSet
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 617: SMI Referenced Properties/Methods for CIM_ZoneSet (Active)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	The ZoneSet name.
Active		boolean	shall be TRUE. Indicates that this ZoneSet is active and members cannot be changed.

8.2.6.1.9.30 CIM_ZoneSet

The inactive ZoneSets.

Created By : Extrinsic(s): CreateZoneSet
Modified By : Static
Deleted By : DeleteInstance

Class Mandatory: true

Table 618: SMI Referenced Properties/Methods for CIM_ZoneSet (Inactive)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	The ZoneSet name.
Active		boolean	Must be FALSE. Indicates that this ZoneSet is inactive.

8.2.6.1.10 Related Standards

Table 619: Related Standards for Fabric

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.11.0	DMTF

8.2.6.2 Enhanced Zoning Subprofile

8.2.6.2.1 Description

This profile describes the additional zoning functions for enhanced zoning. Note that Sessions are normally part of enhanced zoning, but are included in the base fabric profile to address the various types of zoning operations into a single object model. In this subprofile, then only Zone Alias is added.

8.2.6.2.2 Health and Fault Management

None

8.2.6.2.3 Cascading Considerations

None

8.2.6.2.4 Dependencies on Profiles, Subprofiles, and Packages

Support for the 8.2.6.3, "Zone Control Subprofile" is mandatory for the Enhanced Zoning and Enhanced Zoning Control subprofile.

8.2.6.2.5 Methods of this Profile

CreateZoneAlias

The method creates a ZoneAlias and the association HostedCollection. The newly created association, HostedCollection, associates the ZoneAlias to the same AdminDomain the ZoneService is hosted to. For the newly created ZoneAlias, the Active property is always set to false.

```
CreateZoneAlias(
     [IN] string CollectionAlias,
     [OUT] CIM_NamedAddressCollection ref ZoneAlias);
```

AddZoneAlias

Adds to the Zone the specified ZoneAlias.

```
AddZoneAlias(
     [IN] CIM_Zone ref Zone,
     [IN] CIM_NamedAddressCollection ref ZoneAlias);
```

8.2.6.2.6 Client Considerations and Recipes

8.2.6.2.6.1 Create a ZoneAlias

```
// DESCRIPTION
// Create zone alias and add new zone member based on
// the parameters collected by the  CIM Client.
// Before any operations can be imposed on the zoning
// service, a session is requested and obtained from the zone
// service. Create a new ZoneAlias. The session may not be ended if
// the ZoneAlias is empty, so add a zone member to the new ZoneAlias.
// The session is released when the operations are
// completed.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.The system of interest,either the fabric (AdminDomain)
//   or the switch (ComputerSystem), has been
//   previously identified and defined in the
//   $System-> variable
// 2.The name of the new zone alias is defined in the
//   #ZoneAliasName variable
// 3.   The zone member type is defined in the #ConnectivityMemberType
//   variable
// 4.   The zone member Id of the new zone member is defined in the
//   #ConnectiivityMemberID variable


// 1. Get the ZoneService and start a session
$ZoneServices->[] = AssociatorNames(
     $System->,
     “CIM_HostedService”,
     “CIM_ZoneService”, null, null)

// Assumption 1 above guarantees there is a zone service for this
// system. the fabric and switch profiles that there is no more than
// one ZoneService for this system
$ZoneService-> = $ZoneServices[0]

if(!&startSession($ZoneService->))
{
     return
}

// 2. Create the ZoneAlias
%InArguments[“CollectionAlias”] = #ZoneAliasName
#status = InvokeMethod(
     $ZoneService->,
     “CreateZoneAlias”,
     %InArguments[],
     %OutArguments[])

$ZoneAlias-> = %OutArguments[“ZoneAlias”]
if(#status != 0)
     // ERROR!

// 3. Create or locate a ZoneMembershipSettingData
%InArguments[“ConnectivityMemberType”] = #ConnectivityMemberType
%InArguments[“ConnectivityMemberID”] = #ConnectivityMemberID
%InArguments[“SystemSpecificCollection”] = $ZoneAlias->
#status = InvokeMethod($ZoneService->, “CreateZoneMembershipSettingData”,
                       %InArguments[], %OutArguments[])

// 4. Add to zone alias if not created as a member of the zone alias
//    Zone member reference is set accordingly in the output arguments.

$ZoneMember-> = %OutArguments[“ZoneMembershipSettingData”]
```

```
if (#status != 0)
    // ERROR!


// 5. End the session gracefully
&endSession($ZoneService->)
// 6. Verify that the ZoneAlias exists in the database
try{
     GetInstance($ZoneAlias->)
}catch(CIM_ERR_NOT_FOUND){
     // error
}
```

8.2.6.2.6.2 Delete a ZoneAlias

```
// DESCRIPTION
// Delete a zone alias.
// Before any operations can be imposed on the zoning service, a
// session is requested and obtained from the zone service.
// The session is released when the operations are completed.
//
// if the deletion fails, it may be because the Zone Alias is not empty.
// In this case, remove all members from the alias by deleting the
// ElementSettingData associations, and try the deletion again.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.The system of interest,either the fabric (AdminDomain)
//   or the switch (ComputerSystem), has been
//   previously identified and defined in the
//   $System-> variable
// 2.The object name of the zone alias to be deleted is
//   defined in the $ZoneAlias-> variable

// 1. Get the zone service and start a session
$ZoneServices->[] = AssociatorNames(
     $System->,
     “CIM_HostedService”,
     “CIM_ZoneService”,
     null,
     null)

// Assumption 1 above guarantees there is a zone service for this
// system. the fabric and switch profiles that there is no more than
// one ZoneService for this system
$ZoneService-> = $ZoneServices[0]
```

```
if(!&startSession($ZoneService->))
{
     return
}

// 2. Attempt to delete the alias
try{
     DeleteInstance($ZoneAlias->)
}catch(CIM_ERR_FAILED){
     // Try to remove any zone members in the alias
     // via the ElementSettingData association
     $ZoneMembers->[] = referenceNames($ZoneAlias->,
            “CIM_ElementSettingData”,
            null)
     for #j in $ZoneMembers->[] {
        DeleteInstance(ZoneMembers[#j])
     }
     // Try again
     DeleteInstance($ZoneAlias->)

}
// 3. End Session
&endSession($ZoneService->)
// verify that the deletion occurred
try{
     GetInstance($ZoneAlias->)
}catch(CIM_ERR_NOT_FOUND){
     //expect exception
     return
}
// error!!
```

8.2.6.2.7 Registered Name and Version

Enhanced Zoning and Enhanced Zoning Control version 1.1.0

8.2.6.2.8 CIM Server Requirements

Table 620: CIM Server Requirements for Enhanced Zoning and Enhanced Zoning Control

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.2.9 CIM Elements

Table 621: CIM Elements for Enhanced Zoning and Enhanced Zoning Control

Element Name	Description
Mandatory Classes	
CIM_HostedCollection (8.2.6.2.9.1)	Associates the NameAddressCollection representing the Zone Alias to the System
CIM_MemberOfCollection (8.2.6.2.9.2)	Associates the ZoneMembershipSettingData to the NamedAddressCollection
CIM_NamedAddressCollection (8.2.6.2.9.3)	The Zone Alias.
CIM_ZoneService (8.2.6.2.9.4)	The service that allows for all of the zoning configuration changes.

8.2.6.2.9.1 CIM_HostedCollection

Associates the NamedAddressCollection representing the Zone Alias to the System (AdminDomain representing the Fabric or the ComputerSystem representing the switch)

Class Mandatory: true

Table 622: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System
Dependent		CIM_SystemSpecificCollection	The reference to the NamedAddressCollection representing the Zone Alias.

8.2.6.2.9.2 CIM_MemberOfCollection

Associates the ZoneMembershipSettingData to the NamedAddressCollection representing the Zone Alias.

Class Mandatory: true

Table 623: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the NamedAddress-Collection
Member		CIM_ManagedElement	The reference to the ZoneMembership-SettingData

8.2.6.2.9.3 CIM_NamedAddressCollection

The Zone Alias.

Class Mandatory: true

Table 624: SMI Referenced Properties/Methods for CIM_NamedAddressCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
CollectionAlias		string	The Zone Alias Name

8.2.6.2.9.4 CIM_ZoneService

The service that allows for all of the zoning configuration changes.

Class Mandatory: true

Table 625: SMI Referenced Properties/Methods for CIM_ZoneService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping SystemsCreationClass-Name.'
SystemName		string	The scoping SystemsName.'
CreationClassName		string	The Class Name
Name		string	Opaque
CreateZoneAlias()			
AddZoneAlias()			

8.2.6.2.10 Related Standards

Table 626: Related Standards for Enhanced Zoning and Enhanced Zoning Control

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.6.3 Zone Control Subprofile

8.2.6.3.1 Description

The zoning model includes extrinsic methods for creating Zone Sets, Zones, and Zone Members and adding Zones to Zone Sets and Zone Members to Zones. Additionally SMI-S defines intrinsic methods for the removing of Zone Members from Zones and Zone Aliases, Zones from Zone Sets, and deleting Zone Members, Zones, and Zone Sets.

When an Inactive ZoneSet is “Activated”, new instances representing the Active Zone Set and Active Zones are generated from the Inactive Zone Set definition (where a switch may prune the referenced Zone Set collapsing aliases, removes empty zones, etc.).

When a new Zone Set is “Activated”, the instances representing the previous active Zone Set no longer exists.

In the case where the Inactive Zone Sets are hosted on a switch, the client cannot know which Inactive Zone Set was used to define the current Active Zone Set. Also if two Inactive Zone Sets with the same name are hosted on two different switches, the definitions maybe completely different.

8.2.6.3.2 Durable Names and Correlatable IDs of the Profile

None

8.2.6.3.3 Instrumentation Requirements

The agent shall support the use case defined in the 8.2.6.3.8, "Client Considerations and Recipes".

8.2.6.3.4 Health and Fault Management

None

8.2.6.3.5 Cascading Considerations

None

8.2.6.3.6 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.6.3.7 Methods of this Profile

The CIM Server shall support extrinsic methods for the Fabric Discovery Profile.

CreateZoneSet

The method creates a ZoneSet and associates it to the System (AdminDomain representing the Fabric or the ComputerSystem representing the Switch) that the ZoneService is hosted on.

```
CreateZoneSet (
            [IN] string ZoneSetName,
            [OUT] CIM_ZoneSet ref ZoneSet);
```

CreateZone

The method creates a Zone and associates it to System (AdminDomain representing the Fabric or the ComputerSystem representing the Switch) that the ZoneService is hosted on.

```
CreateZone (
            [IN] string ZoneName,
            [IN] uint16 ZoneType,
            [IN] uint16 ZoneSubType,
            [OUT] CIM_Zone ref Zone);
```

CreateZoneMembershipSettingData

The method creates a ZoneMembershipSettingData (a zone member) and adds it to the specified Zone or NamedAddressCollection representing a Zone Alias. The ConnectivityMemberID is dependent upon the ConnectivityMemberType.

For Fibre Channel, the ConnectivityMemberType of “PermanentAddress”, the ConnectivityMemberID is the NxPort WWN; for ConnectivityMemberType of “NetworkAddress”, the ConnectivityMemberID is the NXPort Address ID; for ConnectivityMemberType of “SwitchPortID”, the ConnectivityMemberID is “Domain:PortNumber”.

```
CreateZoneMembershipSettingData (
            [IN] uint16 ConnectivityMemberType,
            [IN] string ConnectivityMemberID,
            [IN] CIM_SystemSpecificCollection ref SystemSpecificCollection,
            [OUT] CIM_ZoneMembershipSettingData ref ZoneMembershipSettingData);
```

AddZone

The method adds to the specified ZoneSet the specified Zone. Adding a Zone to a ZoneSet, extends the zone enforcement definition of the ZoneSet to include the members of that Zone. If adding the Zone is, successful, the Zone should be associated to the ZoneSet by MemberOfCollection.

```
AddZone (
            [IN] CIM_ZoneSet ref ZoneSet,
            [IN] CIM_Zone ref Zone);
```

AddZoneMembershipSettingData

The method adds to the specified Zone or NamedAddessCollection representing the Zone Alias the specified ZoneMembershipSettingData (a zone member).

```
AddZoneMembershipSettingData (
            [IN] CIM_SystemSpecificCollection ref SystemSpecificCollection,
            [IN] CIM_ZoneMembershipSettingData ref ZoneMembershipSettingData);
```

ActivateZoneSet

The method activates the specified ZoneSet. Once a ZoneSet is activated, a ZoneSet with the property Active set to true, its associated Zones with the property Active set to true, and the Zone’s associated ZoneMembershipSettingData are instantiated.

ActivateZoneSet shall be supported outside of a session. ActivateZoneSet being called within a session is implementation specific.

Calling ActivateZoneSet outside of a session while a session is open is implementation specific.

```
Uint32 ActivateZoneSet (
            [IN] CIM_ZoneSet ref ZoneSet,
            [IN] boolean Activate )
```

SessionControl

The method enables a client to request a lock of the fabric to begin zoning configuration changes.

This method allows a client to request or release a lock on the fabric for zoning configuration changes. As described in FC-GS, in the context of Enhanced Zoning Management, management actions to a Zone Server (e.g., write access to the Zoning Database) shall occur only inside a GS session. Clients executing zoning management operations shall use fabric sessions cooperatively if the SMI-S agent supports it. (If the value of SessionState is 4 (“Not Applicable”) then no cooperative session usage is possible).

Before a client executes zoning management operations (intrinsic or extrinsic methods), the client shall request a new session and wait for the request to be granted. To request a new session, first wait until the property “SessionState” of the fabric’s ZoneService is 3 (“Ended”) and the property “RequestedSessionState” is 5 “No Change”. Then call SessionControl with RequestedSessionState = 2 (“Started”). Once zoning management operations are completed, the client shall release the session to enable the provider to propagate changes to the fabric, and to allow other clients to perform management operations. To end a session and commit the changes, call SessionControl with RequestedSessionState = 3 (“Ended”). To abort a sequence of zoning management operations without updating the fabric, call SessionControl with RequestedSessionState = 4 (“Terminated”).

SMIS agents shall block on calls to SessionControl until the request is fulfilled. For example, an error may occur while committing changes to a fabric, i.e., after a call to SessionControl with RequestedSessionState = 3 (“Ended”). The method cannot return until the session has ended, so that a CIM error can be returned if a problem occurs. While the method is in progress, another client may read the value of the RequestedSessionState property and see the value set by the method currently in progress. Once the request is fulfilled, the RequestedSessionState property is set to value 5 “No Change”, regardless of the value in the setInstance operation.

Sessions can timeout. The session timeout behavior and settings are defined by FC-SW in the section discussing mapping GS sessions for Enhanced Zoning Management.

A SMIS agent may raise an error if these client cooperation rules are not followed. For the purposes of a SMIS agent, a series of requests from the same authenticated entity are considered to be from a single client. An agent may verify that such a series corresponds to the sequence described above and raise the error CIM_ERR_FAILED at any time if the sequence is violated.

```
Uint32 SessionControl (
            [IN,
             ValueMap {"2", "3", "4"},
             Values {"Started", "Ended", "Terminated"}]
            uint16 RequestedSessionState;};
```

<u>Intrinsics for removing a zone from a zone set</u>

As seen in the instance diagram, a zone is a member of a zone set if there is a “CIM_MemberOfCollection” association from the zone set to the zone. To remove a zone from a zone set, delete the instance of the association “CIM_MemberOfCollection” using the intrinsic operation deleteInstance.

<u>Intrinsics for removing a zone alias from a zone</u>

A zone alias is a member of a zone if there is a “CIM_MemberOfCollection” association from the zone to the zone alias. To remove a zone alias from a zone set, delete the instance of the association “CIM_MemberOfCollection” using the intrinsic operation deleteInstance.

<u>Intrinsics for removing a zone member from a zone or zone alias</u>

Zone members are represented by CIM_ZoneMembershipSettingData instances. No instance of CIM_ZoneMembershipSettingData exists unless it is associated to a zone or zone alias by a CIM_ElementSettingData association. However, an instance of CIM_ZoneMembershipSettingData may be associated to more than one zone or zone alias.

Removing a zone member from a zone or zone alias is equivalent to deleting the instance of the CIM_ElementSettingData association. Delete the instance using the intrinsic operation deleteInstance.

If this is the last instance of a CIM_ElementSettingData association for a particular CIM_ZoneMembershipSettingData, do not delete the instance of CIM_ZoneMembershipSettingData; it is the provider's responsibility to clean up these structures.

Intrinsic for deleting a zone member

Zone members are represented by CIM_ZoneMembershipSettingData instances associated to zones or zone aliases via CIM_ElementSettingData associations. To delete a zone member (and remove it from any zones or zone aliases from which it is a member) use the CIM operation deleteInstance to delete the instance of CIM_ZoneMembershipSettingData.

Do not delete the corresponding instances of the CIM_ElementSettingData; it is the provider's responsibility to clean up these structures.

Intrinsic for deleting a zone, zone alias, or zone set

Use the intrinsic operation deleteInstance to delete a zone, zone alias or zone set. Client are allowed to delete zones or zone aliases that are members of collections (zones or zone sets). Clients are allowed to delete the last member of a zone or zone set, leaving the collection empty.

A zone set or zone cannot be deleted if it is currently active (the error would be CIM_ERR_FAILED). Some implementations may prohibit deleting zonesets, zones or zone aliases that still have members (the error would be CIM_ERR_FAILED). When a zone, zone alias or zone set is deleted, the client does not have to delete the corresponding instances of CIM_MemberOfCollection or CIM_HostedCollection; it is the provider's responsibility to clean up these structures.

8.2.6.3.8 Client Considerations and Recipes

Many agent implementations do not allow Zone, a ZoneAlias or a Zone Set to be defined empty. Since the methods defined in SMI-S do not support creating a Zone Set with a Zone and a Zone with a Zone Member, the SessionControl method should be used to build a Zone Definition that is interoperable. This is done by calling ZoneSession() to “Start” defining or updating the Zone Definition. The client then calls the appropriate methods as necessary to build the desired Zone Definition. For example, calling CreateZoneSet() to create a new Zone Set, CreateZone() to create a new Zone, AddZoneToZoneSet() to add the newly created Zone to the newly created Zone Set, and CreateZoneMembershipSettingData() to create and add a new Zone Member to the newly created Zone. Upon completion of the new zoning definition, ZoneControl is called again to “End” the session. The changes to the Zone Definition would then be applied to the Zone Set Database. This set of calls would create a Zone Definition where the Zone and ZoneSet are not empty and would be interoperable across all agent implementations.

8.2.6.3.8.1 Create or delete zones Common Functions

```
// DESCRIPTION
//
// Common functions used by the recipes below.
//
// startSession: attempt to start fabric session if required;
//   returns false if attempt fails; returns true if attempt succeeds
//   or if session control is unnecessary
//
// endSession: finalize fabric session if required; returns false
//   if attempt fails; returns true if attempt succeeds or if session
//   control is unnecessary
//
```

```
//
// findActiveZoneSet: routine to find the active zoneset
// on a fabric, and return the reference to it
//


// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// None

sub boolean startSession ($ZoneService->)
{
    $ZoneService = GetInstance($ZoneService->, false, false, false, null)

    // session statuses
    #Ended = 3
    #NotApplicable = 4

    // requested session statuses
    #Started = 2
    #NoChange = 5

    if ($ZoneService.SessionState == #NotApplicable)
         return true // no session control implemented by this agent

    if ($ZoneService.SessionState != #Ended)
         return false // fabric session is in use by another client or agent

    if ($ZoneService.RequestedSessionState != #NoChange)
         return false // another client has already requested session

    %InArguments[“RequestedSessionState”] = #Started

    #status = InvokeMethod($ZoneService->, “SessionControl”, %InArguments,
%OutArguments)
    if (#status != 0) // e.g. “Failed”
        return false

    $ZoneService = GetInstance($ZoneService->, false, false, false, null)
    if ($ZoneService.SessionState != #Started)
        return false

    return true
}


// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
```

```
// None

sub boolean endSession ($ZoneService->) {
    $ZoneService = GetInstance($ZoneService->, false, false, false, null)

    // session statuses
    #Started = 2
    #NotApplicable = 4

    // requested session statuses
    #End = 3

    if ($ZoneService.SessionStatus == #NotApplicable){
        return true      // no need for session control

    if ($ZoneService.SessionStatus != #Started)
        return false     // no session started by this client

    %InArguments["RequestedSessionState"] = #End
    #status = InvokeMethod($ZoneService, "SessionControl", %InArguments,
%OutArguments)
    if (#status != 0)   // e.g. "Failed"
        return false

    // Do not wait, or even check, for SessionState to have value "Ended" as
    // a) InvokeMethod will block till done (or failed) anyway
    // b) Before the check can be made, session may already be started
    //    by another client

    return true
}


// PREEXISTING CONDITIONS AND ASSUMPTIONS
// The reference to the fabric on which the active
// zoneset it to be sought is already known in
// the input variable $Fabric. Calling code
// should verify that the returned reference is non-null
//
sub Ref findActiveZoneSet($Fabric->){
     $ActiveZoneSet->=null
     $ZoneSets[] = Associators(
            $Fabric->,
            "CIM_HostedCollection",
            "CIM_ZoneSet",
            null,
            null,
            false,
            false,
            {“Active”} )
    // there may be no active zoneset
    if(0 < ZoneSets[].size()){
       for(#i in $ZoneSets[]){
          if(true==$ZoneSets[#i].Active){
             $ActiveZoneSet->=nameof($ZoneSets[#i])
             break
          }
       }
    }
    return $ActiveZoneSet->
}
```

8.2.6.3.8.2 Add new Zone Member to Existing Zone

```
// DESCRIPTION
// Add new Zone Member to Existing Zone
//
// Assume the client has already invoked some logic to determine which
// System (fabric or switch) will host the zone database and zone
// service to be used.  Request and obtain a fabric session from the
// zone service.  Use an extrinsic method to attempt to create a new
// instance of ZoneMembershipSettingData, associated to a zone.  If
// the creation fails because an instance already exists for the
// desired zone member id, simply create an association between the
// pre-existing ZoneMembershipSettingData instance and the zone
// instance.  Then close the fabric session.
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The System hosting the zone database (ComputerSystem or
//    AdminDomain) has been previously identified and defined in the
//    $System-> variable
//
// 2. The zone member type is defined in the #ConnectivityMemberType variable
//
// 3. The zone member id of the new zone member is defined in the
//    #ConnectivityMemberID variable
//
// 4. An existing zone is defined in the $Zone-> variable
//
// FUNCTIONS

// 1. Get the Zone Service and start the session

$ZoneServices->[] = AssociatorNames($System->, “CIM_HostedService”,
                                    “CIM_ZoneService”, null, null)

// Assumption 1 (above) guarantees there is a zone service for this
// System, Fabric Profile mandates there is no more than one zone
// service for this System
$ZoneService-> = $ZoneService->[0]

// Start the session
if (!&startSession($ZoneService->)) {
    <ERROR! Failed to start zone session>
}

// 2. Create a ZoneMembershipSettingData
%InArguments[“ConnectivityMemberType”] = #ConnectivityMemberType
%InArguments[“ConnectivityMemberID”] = #ConnectivityMemberID
%InArguments[“SystemSpecificCollection”] = $Zone->
#status = InvokeMethod($ZoneService->, “CreateZoneMembershipSettingData”,
                       %InArguments[], %OutArguments[])

if (#status != 0){
     <ERROR! call to method CreateZoneMembershipSettingData failed #status>
}
// 3. Store the returned object path for verification
$ZoneMember-> = %OutArguments[“ZoneMembershipSettingData”]

// 4. End session successfully
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end session, changes may not have been committed>
}

// 5. Verify that the zonemember exist within the specified zone

$ZoneMembers->[]=associatorNames(
        $Zone->,
        “CIM_ElementSettingData”,
        “CIM_ZoneMembershipSettingData”,
        “ManagedElement”,
        “SettingData” )
if(!contains($ZoneMember->,$ZoneMembers[])){
     <ERROR! Failed to verify zone member created>
}
```

8.2.6.3.8.3 Create new Zone, add new Zone Member, and add to existing ZoneSet

```
// DESCRIPTION
// Create new Zone, add new Zone Member, and add to existing ZoneSet
//
```

```
// Assume the client has already invoked some logic to determine which
// System (fabric or switch) will host the zone database and zone
// service to be used.  Request and obtain a fabric session from the
// zone service.  Create a new Zone using an extrinsic method.  The
// session may not be ended if any zone is empty, so add a zone member
// to the new zone.  The session also may not be ended unless every
// zone is a member of at least one zone set, so add the new zone to
// an existing zone set.  Then close the fabric session.
//
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The System hosting the zone database (ComputerSystem or
//    AdminDomain) has been previously identified and defined in the
//    $System-> variable
//
// 2. The name for a new zone is defined in the #ZoneName variable
//
// 3. The type for the new zone is defined in the #ZoneType variable
//
// 4. The sub type for the new zone is defined in the #ZoneSubType
//    variable
//
// 5. The zone member type is defined in the #ConnectivityMemberType variable
//
// 6. The zone member id of the new zone member is defined in the
//    #ConnectivityMemberID variable
//
// 7. An existing zoneSet is defined in the $ZoneSet-> variable
//
// FUNCTIONS

// 1. Get the Zone Service and start the session
$ZoneServices->[] = AssociatorNames($System->, “CIM_HostedService”,
                                    “CIM_ZoneService”, null, null)

// Assumption 1 (above) guarantees there is a zone service for this
// System, Fabric Profile mandates there is no more than one zone
// service for this System
$ZoneService-> = $ZoneServices->[0]

    if (!&startSession($ZoneService->)) {
    <ERROR! Failed to start zone session>
}

// 2. Create a zone
%InArguments[“ZoneName”] = #ZoneName
```

```
%InArguments["ZoneType"] = #ZoneType
%InArguments["ZoneSubType"] = #ZoneSubType
InvokeMethod($ZoneService->, "CreateZone", %InArguments[], %OutArguments[])
$Zone-> = $OutArguments["Zone"]

// 3. Create  a ZoneMembershipSettingData
%InArguments["ConnectivityMemberType"] = #ConnectivityMemberType
%InArguments["ConnectivityMemberID"] = #ConnectivityMemberID
%InArguments["SystemSpecificCollection"] = $Zone->
#status = InvokeMethod($ZoneService->, "CreateZoneMembershipSettingData",
                       %InArguments[], %OutArguments[])

if (#status != 0){
     <ERROR! Call to method CreateZoneMembershipSettingData failed #status>
}
// 4. Save the returned member objectpath for verification
$ZoneMember-> = %OutArguments["ZoneMembershipSettingData"]


// 5. Add the new zone to the existing zone set
%InArguments["ZoneSet"] = $ZoneSet->
%InArguments["Zone"] = $Zone->
#status = InvokeMethod($ZoneService->, "AddZone", %InArguments[], %OutArguments[])
if (#status != 0){
     <ERROR Call to method AddZone failed>
}
// 6. End Session
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end session, changes may not have been committed>
}
// 7. Verify that the zone exists in the zone set
$Zones->[]=associatorNames(
        $ZoneSet->,
        "CIM_MemberOfCollection",
        "CIM_Zone",
        "Collection",
        "Member"
        )
// see if the zone is in the returned array
if(!contains($Zone->,$Zones->[])){
     <ERROR! Failed to verify that Zone was added to ZoneSet>
}
```

8.2.6.3.8.4 Create new ZoneSet and add existing Zone

```
// DESCRIPTION
// Create new ZoneSet and add existing Zone
//
// Assume the client has already invoked some logic to determine which
// System (fabric or switch) will host the zone database and zone
// service to be used.  Request and obtain a fabric session from the
// zone service.  Create a new ZoneSet with a given name, using an
// extrinsic method.  The session may not be ended if any ZoneSet is
// empty, so add an existing zone to the ZoneSet. Then close the
// fabric session.
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. The System hosting the zone database (ComputerSystem or
//    AdminDomain) has been previously identified and defined in the
//    $System-> variable
//
// 2. The name for the new zone set is defined in the #ZoneSetName
//    variable
//
// 3. An existing zone is defined in the $Zone-> variable
//
// FUNCTIONS

// 1. Get the Zone Service and start the session

$ZoneServices->[] = AssociatorNames($System->, “CIM_HostedService”,
                                    “CIM_ZoneService”, null, null)
// Assumption 1 (above) guarantees there is a zone service for this
// System, Fabric Profile mandates there is no more than one zone
// service for this System
$ZoneService-> = $ZoneServices->[0]

if (!&startSession($ZoneService->)){
     <ERROR! Failed to start zone session>
}

// 2. Create a zone set
%InArguments[“ZoneSetName”] = #ZoneSetName
#status = InvokeMethod($ZoneService->, “CreateZoneSet”, %InArguments[],
%OutArguments[])
if (#status != 0){
     <ERROR! Call to method CreateZoneSet failed>
}

$ZoneSet-> = %OutArguments[“ZoneSet”]

// 3. Add the existing zone to the new zone set
%InArguments[“ZoneSet”] = $ZoneSet->
```

```
%InArguments[“Zone”] = $Zone->
#status = InvokeMethod($ZoneService->, “AddZone”, %InArguments[], %OutArguments[])
if (#status != 0){
     <ERROR! Call to method AddZone failed #status>
}

// 4. End Session
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end zone session, changes may not be committed>
}

// 5. Verify that the new zone set exists in the zone database
try{
     GetInstance($ZoneSet->);
}catch(CIM_ERR_NOT_FOUND){
     <ERROR! Failed to verify ZoneSet created>
}
```

8.2.6.3.8.5 Delete zone

```
// DESCRIPTION
// Delete Zone
//
// Try to use intrinsic delete operation to delete a Zone instance.
// Before any operations can be imposed on the zoning service, a
// session is requested and obtained from the zone service.  If the
// deletion fails, this may be because the zone is active, or because
// it is not empty.  In the latter case, remove all members from the
// zone by deleting the ElementSettingData association instances, and
// try the deletion again.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.  The object name of the zone to be deleted is defined in the
//     $Zone-> variable
// 2.  The object name of the zone service object for the System
//     hosting the zone database is defined in the $ZoneService->
//     variable


if(!&startSession($ZoneService->)){
     <ERROR! Failed to start session>
}

try {
    DeleteInstance($Zone->)
}
catch(CIM_ERR_FAILED) {
    // Verify that Zone is not active
    $Zone = GetInstance($Zone->, false, false, false, null)
    if ($Zone.Active) {
        // tell client of its logic problem
        <ERROR! May not delete Zone from active ZoneSet>
    }

    // Failure may be caused because zone has members
    // Try to delete all zone memberships (not zone members themselves)
    $ZoneElements->[] = ReferenceNames($Zone->, “CIM_ElementSettingData”, null)
    for #i in $ZoneElements->[] {
        DeleteInstance($ZoneElements[#i])
    }

    // Try again
    DeleteInstance($Zone->)
}
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end session, changes may not be committed>
}
// Verify that the zone no longer exists in the zone database
try{
     GetInstance($Zone->)
}catch(CIM_ERR_NOT_FOUND){
     // expect failure
     return
}
// error if no exception thrown
<ERROR! Found Zone that should have been deleted>
```

8.2.6.3.8.6 Delete ZoneSet

```
// DESCRIPTION
// Delete Zone Set
//
// Try to use intrinsic delete operation to delete a ZoneSet
// instance. Before any operations can be imposed on the zoning
// service, a session is requested and obtained from the zone service.
// The session is released when the operations are complete.  If the
// deletion fails, this may be because the zone set is active, or
// because it is not empty.  In the latter case, remove all zones from
// the zone set by deleting the MemberOfCollection association
// instances, and try the deletion again.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.  The object name of the zone set to be deleted is defined in the
//     $ZoneSet-> variable
// 2.  The object name of the zone service object for the system
//     hosting the zone database is defined in the $ZoneService->
//     variable

if (!&startSession($ZoneService->))
     <ERROR! Failed to start session>
}

try {
    DeleteInstance($ZoneSet->)
}
catch(CIM_ERR_FAILED) {
    $ZoneSet = GetInstance($ZoneSet->, false, false, false, null)
    if ($ZoneSet.Active) {
        // tell client of logic problem
        <ERROR! May not delete an active ZoneSet>
    }

    // Failure may be because zoneset is not empty
    $ZoneMemberships->[] = ReferenceNames($ZoneSet->, “CIM_MemberOfCollection”,
null)
    for #i in $ZoneMemberships->[] {
        DeleteInstance($ZoneMemberships->[$i])
    }

    // Try again
    DeleteInstance($ZoneSet->)
}
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end session, changes may not have been committed>
}
// Verify that the deletion did indeed occur
try{
     GetInstance($ZoneSet->)
}catch(CIM_ERR_NOT_FOUND){
     // expected, not a recipe error
     return
}
// error if no exception caught
<ERROR! Found ZoneSet that should have been deleted>
```

8.2.6.3.8.7 Delete ZoneMember

```
// DESCRIPTION
// Delete a zone member, removing it from any zones and aliases of
// which it is a member.
//
```

```
// Use the intrinsic delete operation to delete a
// ZoneMembershipSettingData instance. Before any operations can be
// imposed on the zoning service, a session is requested and obtained
// from the zone service.  The session is released when the operations
// are complete.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.  The object name of the ZoneMembershipSettingData to be deleted is defined in
the
//     $ZoneMember-> variable
// 2.  The object name of the zone service object for the system
//     hosting the zone database is defined in the $ZoneService->
//     variable

if(!&startSession($ZoneService->)){
     <ERROR! Failed to start session>
}

DeleteInstance($ZoneMember->)
if(!&endSession($ZoneService->)){
     <ERROR! Failed to end session, changes may not have been committed>
}
// verify that it is indeed deleted
try{
     GetInstance($ZoneMember->)
}catch(CIM_ERR_NOT_FOUND){
     // expect an exception,
     // not a recipe error
     return
}
// error if no exception caught
<ERROR! Found ZoneMember that should have been deleted>
```

8.2.6.3.9 Registered Name and Version

Zone Control version 1.1.0

8.2.6.3.10 CIM Server Requirements

Table 627: CIM Server Requirements for Zone Control

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.3.11 CIM Elements

Table 628: CIM Elements for Zone Control

Element Name	Description
Mandatory Classes	
CIM_HostedService (8.2.6.3.11.1)	Associates the ZoneService to the AdminDomain representing the fabric or the ComputerSystem representing the switch.
CIM_ZoneService (8.2.6.3.11.2)	The service that allows for all of the zoning configuration changes.

8.2.6.3.11.1 CIM_HostedService

Associates the ZoneService to the AdminDomain representing the fabric or the ComputerSystem representing the switch.

Class Mandatory: true

Table 629: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System representing the fabric or the switch.
Dependent		CIM_Service	The reference to the ZoneService.

8.2.6.3.11.2 CIM_ZoneService

The service that allows for all of the zoning configuration changes.

Class Mandatory: true

Table 630: SMI Referenced Properties/Methods for CIM_ZoneService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping SystemsCreationClassName.

Table 630: SMI Referenced Properties/Methods for CIM_ZoneService

Property	Flags	Type	Description & Notes
SystemName		string	The scoping SystemsName.
CreationClassName		string	The Class Name
Name		string	Opaque
OperationalStatus		uint16[]	Status of Zoning Service.
SessionState		uint16	State of session. Valid values are "Starting", "Ended".
RequestedSessionState		uint16	The requested session state from the client. The valid values that can be set are "Start", "End", and "Terminate".
DefaultZoningState		uint16	
CreateZoneSet()			The method creates a ZoneSet and associates it to the System (AdminDomain representing the Fabric or the ComputerSystem representing the Switch) that the ZoneService is hosted on.
CreateZone()			The method creates a Zone and associates it to System (AdminDomain representing the Fabric or the ComputerSystem representing the Switch) that the ZoneService is hosted on.
CreateZoneMembershipSettingData()			The method creates a ZoneMembershipSettingData (a zone member) and adds it to the specified Zone or NamedAddressCollection representing a Fibre Channel Node.
AddZone()			The method adds to the specified ZoneSet the specified Zone.
AddZoneMembershipSettingData()			The method adds to the specified Zone or NamedAddessCollection representing the Fibre Channel Node the specified ZoneMembershipSettingData (a zone member).
ActivateZoneSet()			The method activates the specified ZoneSet.
SessionControl()			The method enables a client to request a lock of the fabric to begin zoning configuration changes.

8.2.6.3.12 Related Standards

Table 631: Related Standards for Zone Control

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

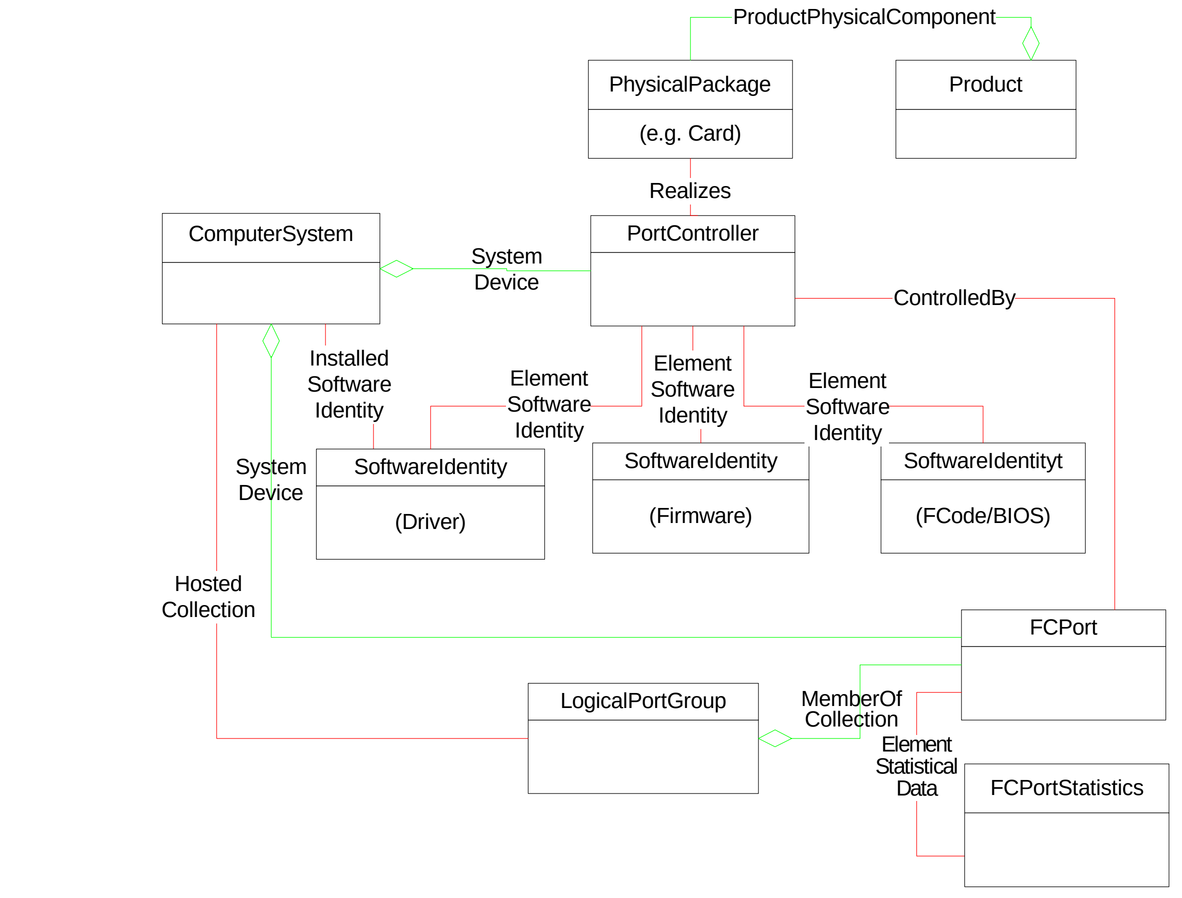
8.2.6.4 FDMI Subprofile

8.2.6.4.1 Description

The Fabric-Device Management Interface (FDMI) enables the management of devices such as HBAs through the Fabric. The FDMI complements data in the Fabric Profile. It allows for any entity in the Fabric to expose through SMI the HBA information without having an agent resident on the Host containing the HBA.

This profile only addresses HBA type devices. The HBA Management Interface defined by FDMI is a subset of interface defined by the Fibre Channel HBA API specification, as exposed by the 8.2.7.1, "FC HBA Profile".

Figure 93: FDMI Instance Diagram



8.2.6.4.2 Health and Fault Management

None

8.2.6.4.3 Cascading Considerations

None

8.2.6.4.4 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.6.4.5 Methods of this Profile

None

8.2.6.4.6 Client Considerations and Recipes

None

8.2.6.4.7 Registered Name and Version

FDMI version 1.1.0

8.2.6.4.8 CIM Server Requirements

Table 632: CIM Server Requirements for FDMI

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.4.9 CIM Elements

Table 633: CIM Elements for FDMI

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.6.4.9.1)	The System the HBA is within.
CIM_ControlledBy (8.2.6.4.9.2)	Associates the ComputerSystem with the PortController
CIM_ElementSoftwareIdentity (8.2.6.4.9.3)	Associates the SoftwareIdentity to the HBA
CIM_FCPort (8.2.6.4.9.4)	The HBA Fibre Channel Port
CIM_HostedCollection (8.2.6.4.9.5)	Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System.
CIM_InstalledSoftwareIdentity (8.2.6.4.9.6)	Associates the SoftwareIdentity representing the driver to the System it is installed on.
CIM_LogicalPortGroup (8.2.6.4.9.7)	The Fibre Channel Node
CIM_MemberOfCollection (8.2.6.4.9.8)	Associates FCPort to the LogicalPortGroup
CIM_PhysicalPackage (8.2.6.4.9.9)	The physical package that the HBA is contained in
CIM_PortController (8.2.6.4.9.10)	The HBA
CIM_Product (8.2.6.4.9.11)	The product information for the HBA
CIM_ProductPhysicalComponent (8.2.6.4.9.12)	Associates the Product to the PhysicalPackage
CIM_Realizes (8.2.6.4.9.13)	Associates the PhysicalPackage to the PortController
CIM_SoftwareIdentity (8.2.6.4.9.14)	The software for the firmware
CIM_SoftwareIdentity (8.2.6.4.9.15)	The software for the driver
CIM_SoftwareIdentity (8.2.6.4.9.16)	The software for the Option ROM
CIM_SystemDevice (8.2.6.4.9.17)	Associates the ComputerSystem with the FCPort
CIM_SystemDevice (8.2.6.4.9.18)	Associates the ComputerSystem with the PortController

8.2.6.4.9.1 CIM_ComputerSystem

The system the HBA is within. It is identified using Host Name from the FDMI interface.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 634: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	The name of the host containing the Device. The key identifier helping in discovery to determine which HBAs are in the same host.
NameFormat		string	

8.2.6.4.9.2 CIM_ControlledBy

Associates the ComputerSystem with the PortController.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 635: SMI Referenced Properties/Methods for CIM_ControlledBy

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Controller	PortController
Dependent		CIM_LogicalDevice	FCPort

8.2.6.4.9.3 CIM_ElementSoftwareIdentity

Associates the SoftwareIdentities representing the various software for the HBA to the PortController representing the HBA.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 636: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SoftwareIdentity	
Dependent		CIM_ManagedElement	

8.2.6.4.9.4 CIM_FCPort

The HBA Fibre Channel Port.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 637: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	Name of Class
DeviceID		string	Opaque
ElementName		string	Port Symbolic Name if available. Other-wise NULL. If the underlying implemen-tation includes characters that are illegal in CIM strings, then truncate before the first of those characters.
LinkTechnology		uint16	"FC"

Table 637: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
PermanentAddress		string	Fibre Channel Port WWN
NetworkAddresses	C	string[]	Fibre Channel ID (FCID). Expressed as 8 unseparated upper case hex digits (see Table 4 for more information about formats).
ActiveFC4Types		uint16[]	The active Fibre Channel FC-4 protocol
PortType		uint16	The specific port type currently enabled (from FC-GS Port.Type)
Optional Properties/Methods			
SupportedFC4Types		uint16[]	An array of integers indicating the Fibre Channel FC-4 protocols supported
SupportedCOS		uint16[]	An array of integers indicating the Fibre Channel Classes of Service that are supported.
Speed		uint64	Speed of zero represents a link not established. 1Gb is 1062500000 bps 2Gb is 2125000000 bps 4Gb is 4250000000 bps 10Gb single channel variants are 10518750000 bps 10Gb four channel variants are 12750000000 bps This is the raw bit rate.

8.2.6.4.9.5 CIM_HostedCollection

Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 638: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System
Dependent		CIM_SystemSpecificCollection	The reference to the LogicalPortGroup (Fibre Channel Node)

8.2.6.4.9.6 CIM_InstalledSoftwareIdentity

Associates the SoftwareIdentity representing the driver to the System it is installed on.

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

Class Mandatory: true

Table 639: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.6.4.9.7 CIM_LogicalPortGroup

The Fibre Channel Node

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 640: SMI Referenced Properties/Methods for CIM_LogicalPortGroup

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
Name	D	string	Fibre Channel Node WWN
NameFormat		string	"WWN"
ElementName	N	string	Node Symbolic Name if available. Otherwise NULL. If the underlying implementation includes characters that are illegal in CIM strings, then truncate before the first of those characters.

8.2.6.4.9.8 CIM_MemberOfCollection

Associates FCPort to the LogicalPortGroup

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 641: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the LogicalPortGroup representing the Fibre Channel Node
Member		CIM_ManagedElement	The reference to FCPort.

8.2.6.4.9.9 CIM_PhysicalPackage

The physical package that the HBA is contained by. It can be simply a PhysicalPackage that the system and HBA is contained within. If it is known that the HBA is on a separate board, Card (a subclass of PhysicalPackage) can be used.

Created By : External

Modified By : Static
Deleted By : External
Class Mandatory: true

Table 642: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Tag		string	An arbitrary string that uniquely identifies the PhysicalPackage.
Manufacturer		string	
Model		string	
Optional Properties/Methods			
ElementName		string	User-friendly name. This property is OPTIONAL.
Name		string	
SerialNumber		string	
Version		string	
PartNumber		string	

8.2.6.4.9.10 CIM_PortController

The HBA. The HBA may have logical operations that can apply to it (e.g., OperationalStatus).

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 643: SMI Referenced Properties/Methods for CIM_PortController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ControllerType		uint16	

8.2.6.4.9.11 CIM_Product

The product information for the HBA

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

Class Mandatory: true

Table 644: SMI Referenced Properties/Methods for CIM_Product

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	Commonly used Product name.
IdentifyingNumber		string	Product identification such as a serial number.
Vendor		string	The manufacturer or the OEM.
Version		string	Product version information.
ElementName		string	User-friendly name. Suggested use is Vendor, Version and product name.

8.2.6.4.9.12 CIM_ProductPhysicalComponent

Associates the Product to the PhysicalPackage. This is necessary to link the Product information to the HBA.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 645: SMI Referenced Properties/Methods for CIM_ProductPhysicalComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_Product	
PartComponent		CIM_PhysicalElement	

8.2.6.4.9.13 CIM_Realizes

Associates the PhysicalPackage to the PortController.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 646: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	
Dependent		CIM_LogicalDevice	

8.2.6.4.9.14 CIM_SoftwareIdentity

The software for the firmware

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

Class Mandatory: true

Table 647: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	
Manufacturer		string	
Classifications		uint16[]	
Optional Properties/Methods			
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.6.4.9.15 CIM_SoftwareIdentity

The software for the driver

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 648: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	
Manufacturer		string	
Classifications		uint16[]	
Optional Properties/Methods			
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.6.4.9.16 CIM_SoftwareIdentity

The software for the Option ROM

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

Class Mandatory: true

Table 649: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	
Manufacturer		string	
Classifications		uint16[]	
Optional Properties/Methods			
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.6.4.9.17 CIM_SystemDevice

Associates the ComputerSystem with the FCPort

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 650: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	ComputerSystem
PartComponent		CIM_LogicalDevice	FCPort

8.2.6.4.9.18 CIM_SystemDevice

Associates the ComputerSystem with the PortController

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 651: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	ComputerSystem
PartComponent		CIM_LogicalDevice	PortController

8.2.6.4.10 Related Standards

Table 652: Related Standards for FDMI

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

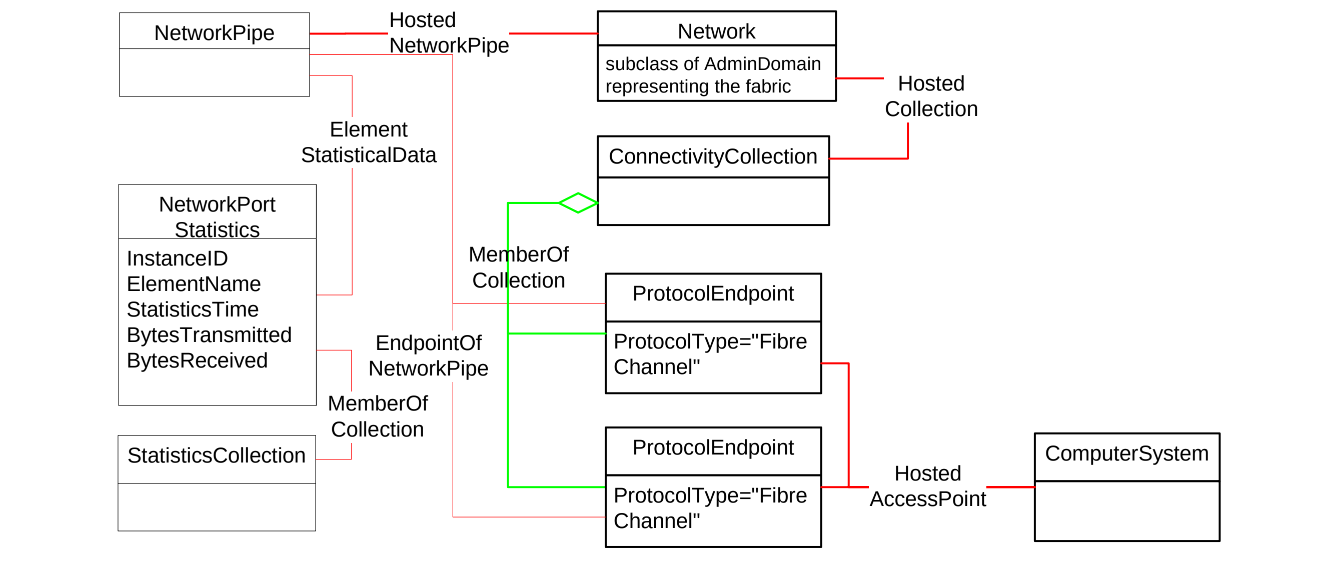
8.2.6.5 Fabric Path Performance Subprofile

8.2.6.5.1 Description

The fabric path performance subprofile extends the standard capabilities of obtaining performance associated to a port to identify the performance in the path defined by an initiator and target ProtocolEndpoint. In the current networking model, the path through the “cloud” is defined by NetworkPipe which is a class that is associated to a ProtocolEndpoint by EndpointOfNetworkPipe. Since the statistics model is defined to allow an association to any LogicalElement, the statistics collected for an NetworkPort, NetworkPortStatistics, can be associated to the NetworkPipe also. When a device supports the Fabric Path Performance Subprofile, it will instantiate the NetworkPipe and as it collects statistics will instantiate the StatisticalData.

The class, StatisticsCollection, provides a mechanism to “collect” all the statistics associated to the NetworkPipes.

Figure 94: Instance Diagram



8.2.6.5.2 Health and Fault Management

None

8.2.6.5.3 Dependencies on Profiles, Subprofiles, and Packages

None

8.2.6.5.4 Methods of this Profile

None

8.2.6.5.5 Client Considerations and Recipes

None

8.2.6.5.6 Registered Name and Version

FabricPathPerformance version 1.1.0

8.2.6.5.7 CIM Server Requirements

Table 653: CIM Server Requirements for FabricPathPerformance

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.5.8 CIM Elements

Table 654: CIM Elements for FabricPathPerformance

Element Name	Description
Mandatory Classes	
CIM_ElementStatisticalData (8.2.6.5.8.1)	Associates FCPortStatistics to the FCPort
CIM_EndpointOfNetworkPipe (8.2.6.5.8.2)	Associates FCPortRateStatistics to the FCPort
CIM_HostedCollection (8.2.6.5.8.3)	Associates the Statistics Collection to the Network representing the fabric.
CIM_HostedNetworkPipe (8.2.6.5.8.4)	Associates NetworkPipe to the Network
CIM_MemberOfCollection (8.2.6.5.8.5)	Associates the NetworkPortStatistics to the StatisticsCollection.
CIM_Network (8.2.6.5.8.6)	Subclass of AdminDomain representing the fabric
CIM_NetworkPipe (8.2.6.5.8.7)	Pipe through the cloud from an initiator to the target.
CIM_NetworkPortStatistics (8.2.6.5.8.8)	NetworkPort Statistics of the NetworkPipe
CIM_ProtocolEndpoint (8.2.6.5.8.9)	The initiator or target (ends of the NetworkPipe).
CIM_StatisticsCollection (8.2.6.5.8.10)	Collection to aggregate the NetworkPipe statistics

8.2.6.5.8.1 CIM_ElementStatisticalData

Associates FCPortStatistics to the FCPort
Class Mandatory: true

Table 655: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	FCPort
Stats		CIM_StatisticalData	FCPortStatistics

8.2.6.5.8.2 CIM_EndpointOfNetworkPipe

Associates FCPortRateStatistics to the FCPort

Class Mandatory: true

Table 656: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	ProtocolEndpoint
Dependent		CIM_NetworkPipe	NetworkPipe

8.2.6.5.8.3 CIM_HostedCollection

Associates the Statistics Collection to the Network representing the fabric.
Class Mandatory: true

Table 657: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_SystemSpecificCollection	

8.2.6.5.8.4 CIM_HostedNetworkPipe

Associates NetworkPipe to the Network
Class Mandatory: true

Table 658: SMI Referenced Properties/Methods for CIM_HostedNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Network	NetworkPipe
Dependent		CIM_NetworkPipe	Network

8.2.6.5.8.5 CIM_MemberOfCollection

Associates the NetworkPortStatistics to the StatisticsCollection.
Class Mandatory: true

Table 659: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	StatisticsCollection
Member		CIM_ManagedElement	FCPortStatistics

8.2.6.5.8.6 CIM_Network

Subclass of AdminDomain representing the fabric

Class Mandatory: true

Table 660: SMI Referenced Properties/Methods for CIM_Network

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	
NameFormat		string	

8.2.6.5.8.7 CIM_NetworkPipe

The NetworkPipe for this profile is instantiated to provide a mechanism to indicate monitors are in place in the network to collect statistical information. NetworkPortStatistics are associated to the pipe via the association ElementStatisticalData to NetworkPortStatistics and subclasses of NetworkPortStatistics (e.g. FCPortStatistics).

Class Mandatory: true

Table 661: SMI Referenced Properties/Methods for CIM_NetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Optional Properties/Methods			
ElementName		string	

8.2.6.5.8.8 CIM_NetworkPortStatistics

Network Port Statistics represent a snapshots of counters for the NetworkPipe. An instance of this class can represent the statistics for the current statistics, archived and consolidated statistics, or both.

Class Mandatory: true

Table 662: SMI Referenced Properties/Methods for CIM_NetworkPortStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
BytesTransmitted		uint64	
BytesReceived		uint64	
Optional Properties/Methods			
ElementName		string	
StatisticTime		datetime	The time the statistics were collected. If historical data is instantiated (present), this property shall be set with the time representing the time the statistic was collected.

8.2.6.5.8.9 CIM_ProtocolEndpoint

The initiator or target (ends of the NetworkPipe).

Class Mandatory: true

Table 663: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
NameFormat		string	
ProtocolIFType		uint16	

8.2.6.5.8.10 CIM_StatisticsCollection

Collection to aggregate the NetworkPipe statistics
Class Mandatory: true

Table 664: SMI Referenced Properties/Methods for CIM_StatisticsCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceId		string	
ElementName		string	
SampleInterval		datetime	
TimeLastSampled		dateTime	

8.2.6.5.9 Related Standards

Table 665: Related Standards for FabricPathPerformance

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.6.6 Switch Profile

8.2.6.6.1 Description

The switch profile models logical and physical aspects of a Fibre Channel fabric switch. The ComputerSystem class constitutes the center of the switch model (and is the top level object which the profile registration points to). An instance of a ComputerSystem is identified as a switch by the property Dedicated set to “switch”.

This profile includes discovery components including ports, port statistics, product information, software, and chassis information. It also includes configuration of the switch including switch and port state change, port speed, switch and port symbolic naming, and DomainID.

Both the Switch and Port have a capabilities class, FCSwitchCapabilities and FCPortCapabilities, respectively, defining which configuration options are supported by the switch. The capabilities define what components are configurable and any restrictions that apply. Except for state change, an associated settings class is defined for both the switch and port, FCSwitchSettings and FCPortSettings, which the client uses to request configuration changes to the Switch or Port, respectively. A setting does not necessarily result in a change to the underlying Switch or Port. The client can determine whether the setting was applied by looking at the associated property in the Switch or Port class.

The model for configuration is made up of three components, capabilities, settings, and the ManagedElements ComputerSystem and FCPort. The capabilities define what components are configurable and any restrictions that apply, the settings define what the client requests, and the ManagedElement expose the actual changes that were applied.

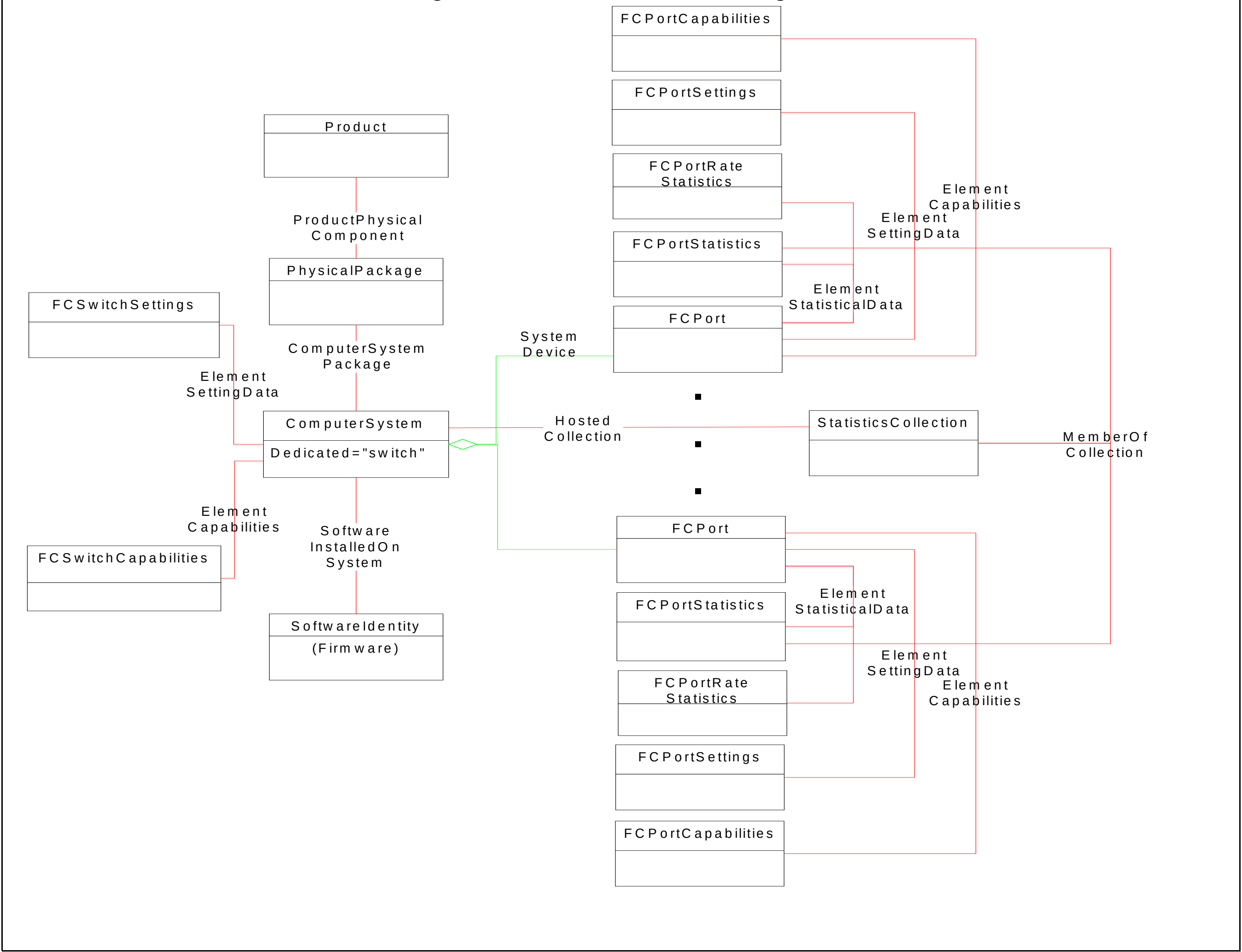
The ComputerSystem (Dedicated as Switch) and FCPort classes have the method RequestStateChange() for requesting the state be changed and an associated property RequestedState on the classes which indicates the current state that has been requested. To determine whether the state change has completed, the property EnabledState can be examined to determine whether the device has completed the state change.

If a switch is modular, for instance if the switch is comprised of multiple blades on a backplane, LogicalModule can optionally be used to model each sub-module, and as an aggregation point for the switch ports. This is described in the Blade Subprofile.

FCPort describes the logical aspects of the port link and the data layers. PhysicalConnector models the physical aspects of a port. An instance of the FCPortStatistics class is expected for each instance of the FCPort class. FCPortStatistics expose real time port health and traffic information.

If the instrumentation is embedded in a switch, it shall provide a switch profile implementation for the hosting switch, and it may proxy a switch profile implementation for other switches reported in the Fabric Profile.

Figure 95: Switch Instance Diagram



8.2.6.6.1.1 FC Port Settings and Capabilities

Capabilities describe the possible features that a ManagedElement supports. Settings are used to describe the requested configuration. The ManagedElement itself describes what settings have been applied and operating.

So lets look at FC Port Type. Here we have are settings that are not in the actual ManagedElement, FCPort.Types. These are settings that allow a subrange of possible port types. They are:

- A G_Port is a Switch Port that is capable of either operating as an E_Port or F_Port. A G_Port determines through Port Initialization whether it operates as an E_Port or as an F_Port.
- A GL_Port is a G_Port that is also capable of operating as an FL_Port.
- A Fx_Port is a switch port capable of operating as an F_Port or FL_Port.

The actual FCPort when operating can only run one of the port types as per FC-GS. In most cases a switch has a default setting to autonegotiate, which in most cases equates to GL or G being set in FCPortSetting.RequestedType. It is required that this setting, FCPortSetting.RequestedType, be shown regardless of whether it was set administratively or is the default behavior of the switch. FCPortSetting.RequestedType represents a setting that the administrator can understand and clearly identify why a switch port ends up running a particular port type. If the switch does not support setting the port type, the RequestedTypesSupported array will be empty. It is valid to have a port type of “Unknown” until the link has been established.

Table 666: FC Port Settings and Capabilities

	FCPort Type	FCPortSetting RequestedType	FCPortCapabilities RequestedTypesSupported
N	X	X	X
NL	X	X	X
F/NL	X		
Nx	X	X	X
E	X	X	X
F	X	X	X
fx		X	X
FL	X		
B	X	X	X
G		X	X
GL		X	X
Unknown	X		
Other	X		

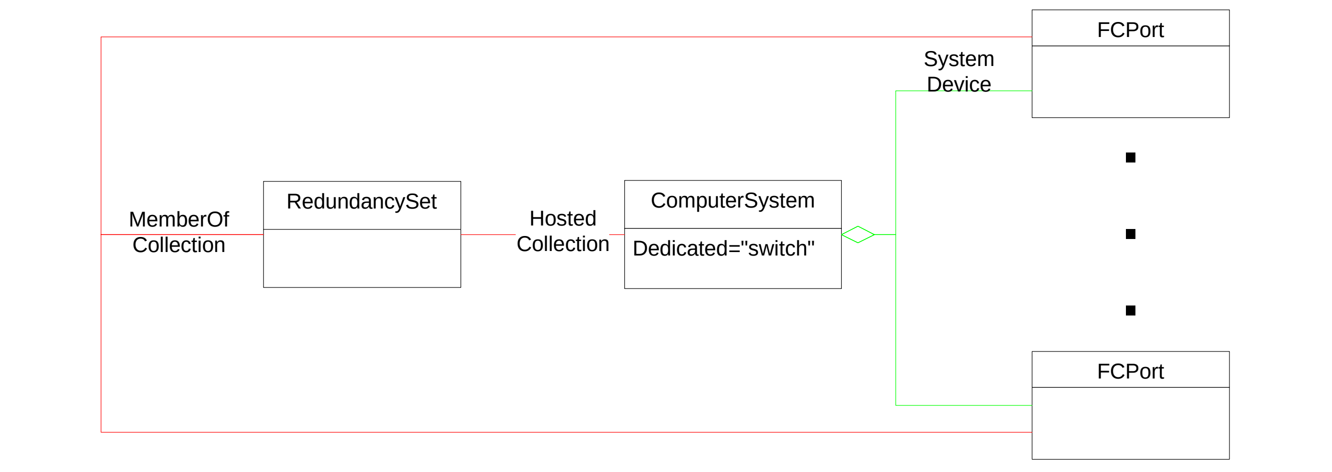
The same concept applies for FCPort settings for speed except there is a separate property indicating auto negotiate, FCPortSettings.AutoSenseSpeed (LogicalPortSettings.AutoSenseSpeed). Note that this setting may have been previously set through some other administrative interface (e.g., CLI), but should still be reported in FCPortSettings.RequestedSpeed. If FCPortSetting.AutoSenseSpeed is true, then the value of FCPortSettings.RequestedSpeed is ignored and the speed will be negotiated by the hardware. If it is disabled, the port will operate at the speed configured in FCPortSettings.RequestedSpeed.

FCPortSettings.RequestedSpeed allows the port speed to be administratively set (WRITE qualifier). It also indicates to the client that the port has been administratively set (now or at a previous time). This property can only be set administratively if FCPortCapabilities.RequestedSpeedsSupported[] is not empty, and may only be set to one of the values in FCPortCapabilities.RequestedSpeedsSupported[].

FCPortCapabilities.RequestedSpeedsSupported indicates whether the device allows the speed to be administratively set. For instance a 4Gb port may allow 1, 2, and 4 Gb. FCPort.Speed (LogicalPort.Speed) represents the actual speed the port is running and a speed of zero represents that the link has not been established.

8.2.6.6.1.2 Trunking

Figure 96: Trunking Instance Diagram



Trunking describes from a switch perspective which ports are working together passing frames using the class RedundancySet. The RedundancySet has a property TypeOfSet which is used to identify what type of redundancy or trunking is occurring among the switch ports associated to the RedundancySet using MemberOfCollection.

8.2.6.6.2 Health and Fault Management

The following classes report possible Health and Fault information through LifeCycle indications:

- ComputerSystem
- FCPort

These LifeCycle indications are more fully described in Table 141, “OperationalStatus Details”.

Also in Table 669, “CIM Server Requirements for Switch” are a list of AlertIndications which may also be indicators for Health and Fault Management.

8.2.6.6.3 Cascading Considerations

None

Dependencies on Profiles, Subprofiles, and Packages

Table 667: Supported Subprofiles for Switch

Registered Subprofile Names	Mandatory	Version
Blades	No	1.1.0
Access Points	No	1.1.0
Software Installation	No	1.1.0
Multiple Computer System	No	1.1.0
Switch Configuration Data	No	1.1.0

Table 668: Supported Packages for Switch

Registered Package Names	Version
Physical Package	1.1.0
Software	1.1.0

8.2.6.6.4 Methods of this Profile

Not defined in this standard.

8.2.6.6.5 Client Considerations and Recipes

8.2.6.6.5.1 Enable FCPort

```
// DESCRIPTION
// This recipe describes how to enable a port on a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance of the port to be enabled is known as $Port.

// MAIN
// Step 1. Retrieve the capabilities of the port.
$PortCapabilities[] = Associators($Port.getObjectPath(),
     “CIM_ElementCapabilities”,
     “CIM_FCPortCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“RequestedStatesSupported”})
if ($PortCapabilities[] == null || PortCapabilities[].length != 1) {
    <ERROR! The required port capabilities are not available>
}

// Step 2. Verify that the port can be enabled.
if (!contains(2, $PortCapabilities[0].RequestedStatesSupported)) {
    <EXIT! Enabling the specified port is not supported>
}

// Step 3. Verify that the port is in a state in which enabling is appropriate.
if ($Port.EnabledState != 2 && $Port.RequestedState == 5) {

    // Step 4. Enable the port.
    %InArguments[“RequestedState”] = 2// “Enabled”
    // Timeout request after 90 seconds
    %InArguments[“TimeoutPeriod”] = 00000000000130.000000:000
    #ReturnValue = InvokeMethod($Port.getObjectPath(),
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
    if (#ReturnValue == 0) {// “Completed with No Error”
     <EXIT! Port successfully enabled>
    } else if (#ReturnValue == 4098) {// “Timeout Parameter Not Supported”
     %InArguments[“RequestedState”] = 2// “Enabled”
     %InArguments[“TimeoutPeriod”] = 0// No timeout
     #ReturnValue = InvokeMethod($Port.getObjectPath(),
        “RequestStateChange”,
        %InArguments,
        %OutArguments)
     if (#ReturnValue == 0) {// “Completed with No Error”
         <EXIT! Port successfully enabled>
     } else {
         <ERROR! Port state transition failed>
     }
    }
} else {
    <ERROR! The specified port is already enabled or currently in a
         state transition>
}
```

8.2.6.6.5.2 Disable Port

```
// DESCRIPTION
// This recipe describes how to disable a port on a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance of the port to be disabled is known as $Port.

// MAIN
// Step 1. Retrieve the capabilities of the port.
$PortCapabilities[] = Associators($Port.getObjectPath(),
     “CIM_ElementCapabilities”,
     “CIM_FCPortCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“RequestedStatesSupported”})
if ($PortCapabilities[] == null || PortCapabilities[].length != 1) {
    <ERROR! The required port capabilities are not available>
}

// Step 2. Verify that the port can be disabled.
if (!contains(3, $Capabilities.RequestedStatesSupported)) {
    <EXIT! Disabling the specified port is not supported>
}
```

```
// Step 3. Verify that the port is in a state in which disabling is appropriate.
if ($Port.EnabledState != 3 && $Port.RequestedState == 5) {

    // Step 4. Disable the port.
    %InArguments[“RequestedState”] = 3// “Disabled”
    // Timeout request after 90 seconds
    %InArguments[“TimeoutPeriod”] = 00000000000130.000000:000
    #ReturnValue = InvokeMethod($Port.getObjectPath(),
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
    if (#ReturnValue == 0) {// “Completed with No Error”
     <EXIT! Port successfully disabled>
    } else if (#ReturnValue == 4098) {// “Timeout Parameter Not Supported”
     %InArguments[“RequestedState”] = 3 // “Disabled”
     %InArguments[“TimeoutPeriod”] = 0 // No timeout
     #ReturnValue = InvokeMethod($Port.getObjectPath(),
        “RequestStateChange”,
        %InArguments,
        %OutArguments)
     if (#ReturnValue == 0) {// “Completed with No Error”
         <EXIT! Port successfully disabled>
     } else {
         <ERROR! Port state transition failed>
     }
    }
} else {
    <ERROR! The specified port is already disabled or currently in a
         state transition>
}
```

8.2.6.6.5.3 Enable Switch

```
// DESCRIPTION
// This recipe describes how to enable a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch to enable is known and defined in the
//    variable $Switch->.

// MAIN
// Step 1. Retrieve the relevant Switch instance information.
$Switch = GetInstance($Switch->,
     false,
     false,
     false,
     {“EnabledState”, “RequestedState”})
```

```
// Step 2. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“RequestedStatesSupported”})
if ($SwitchCapabilities[] == null || SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 3. Verify that the Switch can be enabled.
if (!contains(2, $SwitchCapabilities[0].RequestedStatesSupported)) {
    <EXIT! Enabling the specified Switch is not supported>
}

// Step 4. Verify that the Switch is in a state in which enabling is
// appropriate.
if ($Switch.EnabledState != 2 && $Switch.RequestedState == 5) {

    // Step 5. Enable the Switch.
    %InArguments[“RequestedState”] = 2// “Enabled”
    // Timeout request after 90 seconds
    %InArguments[“TimeoutPeriod”] = 00000000000130.000000:000
    #ReturnValue = InvokeMethod($Switch->,
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
    if (#ReturnValue == 0) {// “Completed with No Error”
     <EXIT! Switch successfully enabled>
    } else if (#ReturnValue == 4098) {// “Timeout Parameter Not Supported”
     %InArguments[“RequestedState”] = 2// “Enabled”
     %InArguments[“TimeoutPeriod”] = 0// No timeout
     #ReturnValue = InvokeMethod($Switch->,
        “RequestStateChange”,
        %InArguments,
        %OutArguments)
     if (#ReturnValue == 0) {// “Completed with No Error”
        <EXIT! Switch successfully enabled>
     } else {
         <ERROR! Switch state transition failed>
     }
    }
} else {
    <ERROR! The specified Switch is already enabled or currently in
         a state transition>
}
```

8.2.6.6.5.4 Disable Switch

```
// DESCRIPTION
// This recipe describes how to disable a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch to disable is known and defined in the
//    variable $Switch->.

// MAIN
// Step 1. Retrieve the relevant Switch instance information.
$Switch = GetInstance($Switch->,
     false,
     false,
     false,
     {“EnabledState”, “RequestedState”})

// Step 2. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“RequestedStatesSupported”})
if ($SwitchCapabilities[] == null || SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 3. Verify that the Switch can be disabled.
if (contains(3, $SwitchCapabilities[0].RequestedStatesSupported)) {
    <EXIT! Disabling the specified Switch is not supported>
}

// Step 4. Verify that the Switch is in a state in which disabling is
// appropriate.
if ($Switch.EnabledState != 3 && $Switch.RequestedState == 5) {

    // Step 5. Disable the Switch.
    %InArguments[“RequestedState”] = 3// “Disabled”
    // Timeout request after 90 seconds
    %InArguments[“TimeoutPeriod”] = 00000000000130.000000:000
    #ReturnValue = InvokeMethod($Switch->,
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
    if (#ReturnValue == 0) {// “Completed with No Error”
     <EXIT! Switch successfully disabled>
    } else if (#ReturnValue == 4098) {// “Timeout Parameter Not Supported”
     %InArguments[“RequestedState”] = 3// “Disabled”
     %InArguments[“TimeoutPeriod”] = 0// No timeout
     #ReturnValue = InvokeMethod($Switch->,
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
     if (#ReturnValue == 0) {// “Completed with No Error”
         <EXIT! Switch successfully disabled>
     } else {
         <ERROR! Switch state transition failed>
     }
    } else {
     <ERROR! Switch state transition failed>
    }
} else {
    <ERROR! The specified Switch is already disabled or currently in
         a state transition>
}
```

8.2.6.6.5.5 Reset Switch

```
// DESCRIPTION
// This recipe describes how to reset a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch to reset is known and defined in the
//    variable $Switch->.

// MAIN
// Step 1. Retrieve the relevant Switch instance information.
$Switch = GetInstance($Switch->,
     false,
     false,
     false,
     {“EnabledState”, “RequestedState”})

// Step 2. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“RequestedStatesSupported”})
if ($SwitchCapabilities[] == null || SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 3. Verify that the Switch can be reset.
if (contains(11, $SwitchCapabilities[0].RequestedStatesSupported)) {
    <EXIT! Resetting the specified Switch is not supported>
}

// Step 4. Verify that the Switch is in a state in which reseting is
// appropriate.
if ($Switch.EnabledState == 2 && $Switch.RequestedState == 5) {

    // Step 5. Reset the Switch.
    %InArguments[“RequestedState”] = 11// “Reset”
    // Timeout request after 90 seconds
    %InArguments[“TimeoutPeriod”] = 00000000000130.000000:000
    #ReturnValue = InvokeMethod($Switch->,
         “RequestStateChange”,
         %InArguments,
         %OutArguments)
    if (#ReturnValue == 0) {// “Completed with No Error”
     <EXIT! Switch successfully reset>
    } else if (#ReturnValue == 4098) {// “Timeout Parameter Not Supported”
     %InArguments[“RequestedState”] = 11// “Reset”
     %InArguments[“TimeoutPeriod”] = 0// No timeout
     #ReturnValue = InvokeMethod($Switch->,
        “RequestStateChange”,
        %InArguments,
        %OutArguments)
     if (#ReturnValue == 0) {// “Completed with No Error”
         <EXIT! Switch successfully reset>
     } else {
         <ERROR! Switch state transition failed>
     }
    } else {
     <ERROR! Switch state transition failed>
    }
} else {
    <ERROR! The specified Switch is already reset or currently in
         a state transition>
}
```

8.2.6.6.5.6 Set Port Speed

```
// DESCRIPTION
// This recipe describes how to modify the speed of a port on a Fibre Channel
// Switch.
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance of the port to whose speed to modify is known as $Port.
// 2. The desired port speed is known and defined in the variable #Speed.

// MAIN
// Step 1. Retrieve the capabilities of the port.
$PortCapabilities[] = Associators($Port.getObjectPath(),
     “CIM_ElementCapabilities”,
     “CIM_FCPortCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“AutoSenseSpeedConfigurable”, “RequestedSpeedsSupported”})
if ($PortCapabilities[] == null || PortCapabilities[].length != 1) {
    <ERROR! The required port capabilities are not available>
}
$Capabilities = $PortCapabilities[0]

// Step 2. Verify that the port speed can be set to the specified speed.
if (contains(#Speed, $Capabilities.RequestedSpeedsSupported)) {

     // Step 3. Retrieve the port settings.
     $Settings[] = Associators($Port.getObjectPath(),
        “CIM_ElementSettingData”,
        “CIM_FCPortSettings”,
        “ManagedSetting”,
        “SettingData”,
        false,
        false,
        {“InstanceID”, “AutoSenseSpeed”, “RequestedSpeed”})
     if ($Settings[] == null || Settings[].length != 1) {
        <ERROR! The required port settings are not available>
     }
     $PortSetting = $Settings[0]

     // Step 4. Port speed is ignored unless AutoSenseSpeed is disabled,
     if ($PortSetting.AutoSenseSpeed) {
        if ($Capabilities.AutoSenseSpeedConfigurable) {
           $PortSetting.AutoSenseSpeed = false
        } else {
           //Unlikely, but not an error
```

```
                }
            }

            // Step 5. Modify the port speed to the specified speed.
            $PortSetting.RequestedSpeed = #Speed
            ModifyInstance($PortSetting.getObjectPath(),
                $PortSetting,
                false,
                {“AutoSenseSpeed”, “RequestedSpeed”})

            // Step 6. Verify that the port speed modification was applied.
            $Port = GetInstance($Port.getObjectPath(),
                false,
                false,
                false,
                {“Speed”})
            if ($Port.Speed == #Speed) {
                <EXIT! Port speed modified successfully>
            } else {
                <ERROR! Port speed was not modified as specified>
            }
      } else {
            <EXIT! Specified port speed is not supported>
      }
```

8.2.6.6.5.7 Set Port Type

```
// DESCRIPTION
// This recipe describes how to modify the port type on a Fibre Channel Switch.
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance of the port to whose type to modify is known as $Port.
// 2. The desired port type is known and defined in the variable #Type.

// MAIN
// Step 1. Retrieve the capabilities of the port.
$PortCapabilities[] = Associators($Port.getObjectPath(),
      “CIM_ElementCapabilities”,
      “CIM_FCPortCapabilities”,
      “ManagedElement”,
      “Capabilities”,
      false,
      false,
      {“RequestedTypesSupported”})
if ($PortCapabilities[] == null || PortCapabilities[].length != 1) {
    <ERROR! The required port capabilities are not available>
}
```

```
// Step 2. Verify that the port type can be modified as specified.
$Capabilities = $PortCapabilities[0]
if (contains(#Type, $Capabilities.RequestedTypesSupported)) {

    // Step 3. Retrieve the port settings.
    $Settings[] = Associators($Port.getObjectPath(),
         “CIM_ElementSettingData”,
         “CIM_FCPortSettings”,
         “ManagedSetting”,
         “SettingData”,
         false,
         false,
         {“RequestedType”})
    if ($Settings[] == null || Settings[].length != 1) {
     <ERROR! The required port settings are not available>
    }
    $PortSetting = $Settings[0]

    // Step 4. Modify the port type to the specified type.
    $PortSetting.RequestedType = #Type
    ModifyInstance($PortSetting.getObjectPath(),
         $PortSetting,
         false,
         {“RequestedType”})

    // Step 5. Verify that the port type modification was applied.
    $Port = GetInstance($PortSetting.getObjectPath(),
         false,
         false,
         false,
         {“RequestedType”})
    if ($PortSetting.RequestedType == #Type) {
     <EXIT! Port type request successfully>
    }
    <ERROR! Port type request was not modified as specified>
} else {
    <ERROR! Port type request cannot be set to specified type>
}
```

8.2.6.6.5.8 Set Fibre Channel Switch Principal Priority

```
// DESCRIPTION
// This recipe describes how to modify the Principal Priority of a Fibre Channel
// Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
```

```
// 1. A reference to the Switch whose Principal Priority to modify is known and
//    defined in the variable $Switch->
// 2. The desired Principal Priority of the Switch is known as #Priority.

// MAIN
// Step 1. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“PrincipalPrioritiesSupported”})
if ($SwitchCapabilities[] == null || SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 2. Verify that the Switch Principal Priority can be modified.
$Capabilities = $SwitchCapabilities[0]
if (!contains(5, $Capabilities.PrincipalPrioritiesSupported[])) {

    $SwitchSettings[] = Associators($Switch->,
     “CIM_ElementSettingData”,
     “CIM_FCSwitchSettings”,
     “ManagedElement”,
     “SettingData”,
     false,
     false,
     {“PrincipalPriority”})
    if ($SwitchSettings[] == null || SwitchSettings[].length != 1) {
     <ERROR! Required Switch settings are not available>
    }
    $Settings = $SwitchSettings[0]

    // Step 3. Ensure a new Principal Priority is being set.
    if (#Priority != $Settings.PrincipalPriority)) {

     // Step 4. Modify the Principal Priority of the Switch.
     $Settings.PrincipalPriority = #Priority
     ModifyInstance($Settings.getObjectPath(),
        $Settings,
        false,
        {“PrincipalPriority”})
     // Step 5. Verify that the Switch priority modification was applied.
     $Settings = GetInstance($Settings.getObjectPath(),
        false,
            false,
            false,
            {“PrincipalPriority”})
        if ($Settings.PrincipalPriority == #Priority) {
            <EXIT! Switch Principal Priority was modified successfully>
        }
        <EXIT! Switch Principal Priority was not modified successfully>
    } else {
        <ERROR! Principal Priority specified is already set>
    }
} else {
    // “Not Applicable”
    <EXIT! The Switch does not support Principal Priority modification>
}
```

8.2.6.6.5.9 Set Switch Name

```
// DESCRIPTION
// This recipe describes how to modify the name of a Fibre Channel Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
//
// 1. A reference to the Switch whose name to modify is known and defined in
//    the variable $Switch->
// 2. The desired name of the Switch is known as #Name.

// MAIN
// Step 1. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“ElementNameEditSupported”, “MaxElementNameLen”})
if ($SwitchCapabilities[] == null || $SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 2. Verify that the Switch name can be modified.
$Capabilities = $SwitchCapabilities[0]
if ($Capabilities.ElementNameEditSupported) {

    // Step 3. Verify that the new name to be specified is within the
    // constraints of the name length supported by the Switch.
    if (#Name.length() < $Capabilities.MaxElementNameLen) {
     // Step 4. Retrieve the instance representing the Switch.
     $Switch = GetInstance($Switch->,
        false,
        false,
        false,
        {"ElementName"})

     // Step 5. Modify the name of the Switch.
     $Switch.ElementName = #Name
     ModifyInstance($Switch->,
        $Switch,
        false,
        {"ElementName"})

     // Step 6. Verify that the Switch name change was applied.
     $Switch = GetInstance($Switch->,
        false,
        false,
        false,
        {"ElementName"})
     if (compare(#Name, $Switch.ElementName)) {
         <EXIT! Switch name was modified successfully>
     }
     <ERROR! Switch name was not modified successfully>
    }
    <ERROR! Specified Switch name exceeds length limit>
} else {
    <EXIT! The Switch does not support name modification>
}
```

8.2.6.6.5.10 Set Port Name

```
// DESCRIPTION
// This recipe describes how to modify the name of a Port on a Fibre Channel
// Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. The instance of the port to whose type to modify is known as $Port.
// 3. The desired name of the port is known as #Name.

// MAIN
// Step 1. Retrieve the capabilities of the port.
$PortCapabilities[] = Associators($Port.getObjectPath(),
     "CIM_ElementCapabilities",
     "CIM_FCPortCapabilities",
     "ManagedElement",
     "Capabilities",
     false,
      false,
      {“ElementNameEditSupported”, “MaxElementNameLen”})
if ($PortCapabilities[] == null || $PortCapabilities[].length != 1) {
    <ERROR! The required Port capabilities are not available>
}

// Step 2. Verify that the port name can be modified.
$Capabilities = $PortCapabilities[0]
if ($Capabilities.ElementNameEditSupported) {

    // Step 3. Verify that the new name to be specified is within the
    // constraints of the name length supported by the port.
    if (#Name.length() < $Capabilities.MaxElementNameLen) {

     // Step 4. Modify the name of the port.
     $Port.ElementName = #Name
     ModifyInstance($Port.getObjectPath(),
        $Port,
        false,
        {“ElementName”})

     // Step 5. Verify that the port name change was applied.
     $Port = GetInstance($Port.getObjectPath(),
        false,
        false,
        false,
        {“ElementName”})
     if (compare(#Name, $Port.ElementName)) {
         <EXIT! Port name was modified successfully>
     }
     <ERROR! Port name was not modified successfully>
    }
    <ERROR! Specified Port name exceeds length limit>
} else {
    <EXIT! The Port does not support name modification>
}
```

8.2.6.6.5.11 Set Fibre Channel Switch Preferred Domain ID

```
// DESCRIPTION
//
// This recipe describes how to modify the preferred Domain ID of a Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch to reset is known and defined in the
//    variable $Switch->.
// 2. The new preferred Domain ID to be set on the Switch is known as #DomainID.
// MAIN
// Step 1. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     “CIM_ElementCapabilities”,
     “CIM_FCSwitchCapabilities”,
     “ManagedElement”,
     “Capabilities”,
     false,
     false,
     {“DomainIDConfigureable”, “MinDomainID”, “MaxDomainID”})
if ($SwitchCapabilities[] == null || SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}

// Step 2. Verify that the Switch’s preferred Domain ID can be modified.
$Capabilities = $SwitchCapabilities[0]
if ($Capabilities.DomainIDConfigureable) {

    // Step 3. Verify that the desired Domain ID is within the permissible
    // range.
    if (#DomainID >= $Capabilities.MinDomainID
         && #DomainID <= $Capabilities.MaxDomainID) {

     // Step 4. Retrieve the Switch settings.
     $Settings[] = Associators($Switch->,
        “CIM_ElementSettingData”,
        “CIM_FCSwitchSettings”,
        “ManagedSetting”,
        “SettingData”,
        false,
        false,
        {“PreferredDomainID”})
     if ($Settings[] == null || Settings[].length != 1) {
         <ERROR! The required Switch settings are not available>
     }
     $SwitchSetting = $Settings[0]

     // Step 5. Modify the Switch Domain ID to the specified preferred value.
     $SwitchSetting.PreferredDomainID = #DomainID
     ModifyInstance($SwitchSetting.getObjectPath(),
        $SwitchSetting,
        false,
        {“PreferredDomainID”})

     // Step 6. Verify that the Switch Domain ID modification was applied.
     $Switch = GetInstance($Switch->,
        false,
        false,
        false,
        {"IdentifyingDescriptions", "OtherIdentifyingInfo"})
    // NOTE: The Domain ID value is contained in the OtherIdentifyingInfo
    // property at the same index as the "DomainID" element index in the
    // IdentifyingDescriptions property.
    #index = -1
    while (#i < $Switch.IdentifyingDescriptions[].length
        && #index < 0) {
         if ($Switch.IdentifyingDescriptions[#i] == "DomainID") {
        #index = #i
         }
    }
    if (#index >= 0 && $Switch.OtherIdentifyingInfo[#index] == #DomainID) {
        <EXIT! Switch Domain ID successfully modified>
    }
    <ERROR! Switch Domain ID was not modified as specified>
   } else {
    <ERROR! Domain ID specified is not within permitted range>
   }
} else {
    <EXIT! Domain ID configuration on the specified Switch is not supported>
}
```

8.2.6.6.5.12 Lock Fibre Channel Switch Domain ID

```
// DESCRIPTION
//
// This recipe describes how to set the Domain ID Lock of a Switch.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch whose Domain ID to lock is known and defined
// in the variable $Switch->.

// MAIN
// Step 1. Retrieve the capabilities of the Switch.
$SwitchCapabilities[] = Associators($Switch->,
     "CIM_ElementCapabilities",
     "CIM_FCSwitchCapabilities",
     "ManagedElement",
     "Capabilities",
     false,
     false,
     {"DomainIDLockedSupported"})
if ($SwitchCapabilities[] == null || $SwitchCapabilities[].length != 1) {
    <ERROR! The required Switch capabilities are not available>
}
```

```
// Step 2. Verify that the Switch's Domain ID Lock can be set.
$Capabilities = $SwitchCapabilities[0]
if ($Capabilities.DomainIDLockedSupported) {

    // Step 3. Retrieve the Switch settings.
    $Settings[] = Associators($Switch->,
         "CIM_ElementSettingData",
         "CIM_FCSwitchSettings",
         "ManagedSetting",
         "SettingData",
         false,
         false,
         {"DomainIDLocked", "PreferredDomainID"})
    if ($Settings[] == null || Settings[].length != 1) {
     <ERROR! The required Switch settings are not available>
    }
    $SwitchSetting = $Settings[0]
    #PreferredDomainID = $SwitchSetting.PreferredDomainID

    // Step 4. Verify that the Domain ID is not already locked.
    if ($SwitchSetting.DomainIDLocked) {
     <EXIT! The Domain ID Lock is already set>
    }

    // Step 5. Lock the Switch Domain ID.
    $SwitchSetting.DomainIDLocked = true
    ModifyInstance($SwitchSetting.getObjectPath(),
         $SwitchSetting,
         false,
         {"DomainIDLocked"})

    // Step 6. Verify that the Switch Domain ID specifies the preferred
    // Domain ID.
    $Switch = GetInstance($Switch->,
         false,
         false,
         false,
         {"IdentifyingDescriptions", "OtherIdentifyingInfo"})
    // NOTE: The Domain ID value is contained in the OtherIdentifyingInfo
    // property at the same index as the "DomainID" element index in the
    // IdentifyingDescriptions property.
    #index = -1
    while (#i < $Switch.IdentifyingDescriptions[].length && #index < 0) {
     if ($Switch.IdentifyingDescriptions[#i] == "DomainID") {
         #index = #i
     }
    }
```

```
        if (#index >= 0 &&
             $Switch.OtherIdentifyingInfo[#index] == #PreferredDomainID) {
         <EXIT! Switch Domain ID successfully locked>
        }
        <ERROR! Switch Domain ID does not reflect the preferred Domain ID>
    } else {
        <EXIT! Domain ID configuration on the specified Switch is not supported>
    }
```

8.2.6.6.6 Registered Name and Version

Switch version 1.1.0

8.2.6.6.7 CIM Server Requirements

Table 669: CIM Server Requirements for Switch

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.6.8 CIM Elements

Table 670: CIM Elements for Switch

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.6.6.8.1)	Represents the Switch
CIM_ComputerSystemPackage (8.2.6.6.8.2)	Associated PhysicalPackage to the ComputerSystem (Switch)
CIM_ElementCapabilities (8.2.6.6.8.3)	Associates FCSwitchCapabilities to the ComputerSystem (Switch)
CIM_ElementCapabilities (8.2.6.6.8.4)	Associates FCPortCapabilities to the FCPort
CIM_ElementSettingData (8.2.6.6.8.5)	Associates FCSwitchSettings to ComputerSystem (Switch)
CIM_ElementSettingData (8.2.6.6.8.6)	Associates FCPortSettings to FCPort
CIM_ElementStatisticalData (8.2.6.6.8.7)	Associates FCPortRateStatistics to the FCPort
CIM_ElementStatisticalData (8.2.6.6.8.8)	Associates FCPortStatistics to the FCPort
CIM_FCPort (8.2.6.6.8.9)	Fibre Channel Switch Port
CIM_FCPortCapabilities (8.2.6.6.8.10)	Switch Port Capabilities
CIM_FCPortStatistics (8.2.6.6.8.13)	Fibre Channel Switch Port Statistics.
CIM_FCSwitchCapabilities (8.2.6.6.8.14)	Fibre Channel Switch Capabilities
CIM_FCSwitchSettings (8.2.6.6.8.15)	Fibre Channel Switch Settings
CIM_HostedCollection (8.2.6.6.8.16)	Associates the Statistics Collection to the Network representing the fabric.
CIM_MemberOfCollection (8.2.6.6.8.17)	Associates the NetworkPortStatistics to the StatisticsCollection.
CIM_StatisticsCollection (8.2.6.6.8.21)	Collection to aggregate the FCPortStatistics for each switch
CIM_SystemDevice (8.2.6.6.8.22)	Associated FCPort to the ComputerSystem (Switch)
Optional Classes	
CIM_FCPortRateStatistics (8.2.6.6.8.11)	Fibre Channel Switch Port Rate Statistics
CIM_FCPortSettings (8.2.6.6.8.12)	Switch Port Settings
CIM_MemberOfCollection (8.2.6.6.8.18)	Associates the FCPort to the RedundancySet.
CIM_ProtocolEndpoint (8.2.6.6.8.19)	The endpoint of a link (ActiveConnection).
CIM_RedundancySet (8.2.6.6.8.20)	The class RedundancySet along with the association MemberOfCollection in this profile is used to show port aggregation for Fibre Channel trunking.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	New Switch Instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Deletion of Switch Instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.Operationalstatus <> PreviousInstance.Operationalstatus	Deprecated WQL - Modification of OperationalStatus in Switch Instance

Table 670: CIM Elements for Switch

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Modification of OperationalStatus in FC Port Instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::Operationalstatus <> PreviousInstance.CIM_ComputerSystem::Operationalstatus	CQL - Modification of OperationalStatus in Switch Instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FCPort AND SourceInstance.CIM_FCPort::OperationalStatus <> PreviousInstance.CIM_FCPort::OperationalStatus	CQL - Modification of OperationalStatus in FC Port Instance

8.2.6.6.8.1 CIM_ComputerSystem

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 671: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	Name of Class
Name	D	string	Switch Name (WWN)
NameFormat		string	"WWN"
OperationalStatus		uint16[]	See Table 586, “OperationalStatus for ComputerSystem”.
Dedicated		uint16[]	"Switch"
RequestedState		uint16	The Switch state requested via RequestStateChange(). Shall be of the range specified in FCSwitchCapabilities.RequestedStatesSupported if a state change has been requested. Otherwise shall be "Not Applicable".
Optional Properties/Methods			
ElementName		string	User-friendly name. Can be set if FCSwitchCapabilities.ElementNameEditSupported for the switch is True.
OtherIdentifyingInfo	C	string[]	DomainID stored in decimal format
IdentifyingDescriptions		string[]	"DomainID" is placed into corresponding index of OtherIdentifyingInfo
EnabledState		uint16	See Table 586, “OperationalStatus for ComputerSystem”
EnabledDefault		uint16	Default startup for the Switch
RequestStateChange()			

8.2.6.6.8.2 CIM_ComputerSystemPackage

Associated PhysicalPackage to the ComputerSystem (Switch)
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 672: SMI Referenced Properties/Methods for CIM_ComputerSystemPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalPackage	The reference to the PhysicalPackage
Dependent		CIM_ComputerSystem	The reference to the ComputerSystem

8.2.6.6.8.3 CIM_ElementCapabilities

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 673: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the ComputerSystem
Capabilities		CIM_Capabilities	The reference to the FCSwitchCapabilities

8.2.6.6.8.4 CIM_ElementCapabilities

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 674: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the FCPort
Capabilities		CIM_Capabilities	The reference to the FCPortCapabilities

8.2.6.6.8.5 CIM_ElementSettingData

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

Class Mandatory: true

Table 675: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the ComputerSystem
SettingData		CIM_SettingData	The reference to the FCSwitchSettings

8.2.6.6.8.6 CIM_ElementSettingData

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 676: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the FCPort
SettingData		CIM_SettingData	The reference to the FCPortSettings

8.2.6.6.8.7 CIM_ElementStatisticalData

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 677: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the FCPort
Stats		CIM_StatisticalData	The reference to the FCPortRateStatistics

8.2.6.6.8.8 CIM_ElementStatisticalData

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 678: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the FCPort
Stats		CIM_StatisticalData	The reference to the FCPortStatistics

8.2.6.6.8.9 CIM_FCPort

The Fibre Channel Switch Port.

Created By : Static
Modified By : ModifyInstanceExtrinsic(s): RequestStateChange
Deleted By : Static
Class Mandatory: true

Table 679: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	The Class Name
DeviceID		string	Opaque
ElementName		string	User-friendly name. Can be set if FCPortCapabilities.ElementNameEdit-Supported is True.
OperationalStatus		uint16[]	See Table 586, “OperationalStatus for ComputerSystem”.
RequestedState		uint16	The port state requested via Request-StateChange(). Shall be of the range specified in FCPortCapabili-ties.RequestedStatesSupported if a state change has been requested. Oth-erwise Shall be "Not Applicable".
EnabledDefault		uint16	Default startup for the port. Used in conjunction with RequestedState can allow for persistent disabling of a port.
Speed		uint64	Speed of zero represents a link not established. 1Gb is 1062500000 bps 2Gb is 2125000000 bps 4Gb is 4250000000 bps 10Gb single channel variants are 10518750000 bps 10Gb four channel variants are 12750000000 bps This is the raw bit rate.
MaxSpeed		uint64	The max speed of the Port in Bits per Second using the same algorithm as Speed.

Table 679: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
PortType		uint16	FC-GS Port.Type The specific mode currently enabled for the Port. The values: "N" = Node Port "NL" = Node Port supporting FC arbitrated loop "E" = Expansion Port connecting fabric elements (for example, FC switches) "F" = Fabric (element) Port "FL" = Fabric (element) Port supporting FC arbitrated loop "B" = Bridge Port. PortTypes are defined in the ANSI INCITS FC-GS standards. Can be set using FCPortSettings.RequestedType.
PortNumber		uint16	NetworkPorts are often numbered relative to either a logical modules or a network element.
PermanentAddress		string	Fibre Channel Port WWN.
LinkTechnology		uint16	"FC"
Optional Properties/Methods			
EnabledState		uint16	See Table 586, “OperationalStatus for ComputerSystem”
RequestStateChange()			Method to change the port state. FCPortCapabilities.RequestedStatesSupported indicates what states can be set.

8.2.6.6.8.10 CIM_FCPortCapabilities

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 680: SMI Referenced Properties/Methods for CIM_FCPortCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	Shall be set to "FC Port Capabilities"
ElementNameEditSupported		boolean	Indicates whether FCPort.ElementName is settable
MaxElementNameLen		uint16	Indicates the maximum string length of FCPort.ElementName

Table 680: SMI Referenced Properties/Methods for CIM_FCPortCapabilities

Property	Flags	Type	Description & Notes
RequestedStatesSupported		uint16[]	Indicates the supported states for calling FCPort.RequestStateChange().
RequestedSpeedsSupported		uint64[]	Indicates the supported speeds that can be set in FCPortSettings.RequestedSpeed
AutoSenseSpeedConfigurable		boolean	Indicates whether FCPortSettings.AutoSenseSpeed can be set to auto-negotiate speed.
RequestedTypesSupported		uint16[]	Indicates the list of supported port types that can be set in FCPortSettings.RequestedType.

8.2.6.6.8.11 CIM_FCPortRateStatistics

Fibre Channel Switch Port Rate Statistics represent the rate per second over the SampleInterval. An instance of this class can represent the statistics for the current statistics, archived and consolidated statistics, or both.

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 681: SMI Referenced Properties/Methods for CIM_FCPortRateStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
StatisticTime		datetime	The time the statistic was collected.
SampleInterval		datetime	The interval at which the rates are calculated.
TxRate		uint64	
RxRate		uint64	
Optional Properties/Methods			
TxFrameRate		uint64	
RxFrameRate		uint64	
MaxTxFrameRate		uint64	
MaxRxFrameRate		uint64	
PeakTxRate		uint64	
PeakRxRate		uint64	

8.2.6.6.8.12 CIM_FCPortSettings

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

Class Mandatory: false

Table 682: SMI Referenced Properties/Methods for CIM_FCPortSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	Shall be set to "FC Port Settings"
RequestedSpeed	M	uint64	The requested value to which FCPort.Speed should be set.
AutoSenseSpeed	M	boolean	The request for the FCPort to auto sense the speed (FCPort.Speed).
RequestedType	M	uint16	The requested setting for the FCPort.PortType.

8.2.6.6.8.13 CIM_FCPortStatistics

Snapshot of performance and error counters for the Fibre Channel Switch.

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 683: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
BytesTransmitted		uint64	
BytesReceived		uint64	
PacketsTransmitted		uint64	
PacketsReceived		uint64	
CRCErrors		uint64	
LinkFailures		uint64	
PrimitiveSeqProtocolErrCount		uint64	
Optional Properties/Methods			
StatisticTime		datetime	The time the statistics were collected. If historical data is instantiated (present), this property shall be set with the time representing the time the statistic was collected.
ElementName		string	
LIPCount		uint64	
NOSCount		uint64	
ErrorFrames		uint64	
DumpedFrames		uint64	
LossOfSignalCounter		uint64	
LossOfSyncCounter		uint64	

Table 683: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
InvalidTransmissionWords		uint64	
FramesTooShort		uint64	
FramesTooLong		uint64	
AddressErrors		uint64	
BufferCreditNotProvided		uint64	
BufferCreditNotReceived		uint64	
DelimiterErrors		uint64	
EncodingDisparityErrors		uint64	
LinkResetsReceived		uint64	
LinkResetsTransmitted		uint64	
MulticastFramesReceived		uint64	
MulticastFramesTransmitted		uint64	
FBSYFrames		uint64	
PBSYFrames		uint64	
FRJTFrames		uint64	
PRJTFrames		uint64	
RXClass1Frames		uint64	
TXClass1Frames		uint64	
Class1FBSY		uint64	
Class1PBSY		uint64	
Class1FRJT		uint64	
Class1PRJT		uint64	
RXClass2Frames		uint64	
TXClass2Frames		uint64	
Class2FBSY		uint64	
Class2PBSY		uint64	
Class2FRJT		uint64	
Class2PRJT		uint64	
RXClass3Frames		uint64	
TXClass3Frames		uint64	
Class3FramesDiscarded		uint64	
RXBroadcastFrames		uint64	
TXBroadcastFrames		uint64	

8.2.6.6.8.14 CIM_FCSwitchCapabilities

The Fibre Channel Switch Capabilities.

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

Class Mandatory: true

Table 684: SMI Referenced Properties/Methods for CIM_FCSwitchCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	Shall be set to "FC Switch Capabilities"
ElementNameEditSupported		boolean	Capability indicating whether ComputerSystem.ElementName for the switch can be set.
MaxElementNameLen		uint16	Capability specifying the maximum name of ComputerSystem.ElementName for the switch
RequestedStatesSupported		uint16[]	The states the switch can support via ComputerSystem.RequestedState.
DomainIDLockedSupported		boolean	
PrincipalPrioritiesSupported		uint16[]	
Optional Properties/Methods			
MinDomainID		uint8	Shall be set if DomainIDConfigurable is TRUE.
MaxDomainID		uint8	Shall be set if DomainIDConfigurable is TRUE.

8.2.6.6.8.15 CIM_FCSwitchSettings

Created By : External
Modified By : ModifyInstance
Deleted By : External
Class Mandatory: true

Table 685: SMI Referenced Properties/Methods for CIM_FCSwitchSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	Shall be set to "FC Switch Settings"
Optional Properties/Methods			
PreferredDomainID	M	uint8	Required if FCSwitchCapabilities.DomainIDConfigurable is TRUE.
DomainIDLocked	M	boolean	Required if FCSwitchCapabilities.DomainIDLockSupported is TRUE.
PrincipalPriority	M	uint16	Required if FCSwitchCapabilities.PrincipalPrioritiesSupported is not set to "Not Applicable".

8.2.6.6.8.16 CIM_HostedCollection

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

Class Mandatory: true

Table 686: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_SystemSpecificCollection	

8.2.6.6.8.17 CIM_MemberOfCollection

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 687: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the StatisticsCollection
Member		CIM_ManagedElement	The reference to the FCPortStatistics

8.2.6.6.8.18 CIM_MemberOfCollection

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 688: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the RedundancySet
Member		CIM_ManagedElement	The reference to the FCPortPort

8.2.6.6.8.19 CIM_ProtocolEndpoint

The endpoint of a link (ActiveConnection). ProtocolEndpoint shall be implemented when BroadcastReset() is supported (Force LIP). It is expected that the Fabric Profile is also implemented which defines the necessary information for determining who will receive the Force LIP on the loop.

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

Class Mandatory: false

Table 689: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System's CreationClass-Name.
SystemName		string	The scoping System's Name.
CreationClassName		string	Name of Class
Name	CD	string	The Fibre Channel Port WWN.
NameFormat		string	"WWN"
ProtocolIFType		uint16	"Fibre Channel"
BroadcastResetSupported		boolean	
Optional Properties/Methods			
BroadcastReset()			Sends a Force LIP to all attached Ports. Required if BroadcastResetSup-ported is TRUE.

8.2.6.6.8.20 CIM_RedundancySet

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 690: SMI Referenced Properties/Methods for CIM_RedundancySet

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
TypeOfSet		uint16[]	

8.2.6.6.8.21 CIM_StatisticsCollection

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 691: SMI Referenced Properties/Methods for CIM_StatisticsCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceId		string	Opaque
ElementName		string	
SampleInterval		datetime	
TimeLastSampled		dateTime	

8.2.6.6.8.22 CIM_SystemDevice

Created By : External

Modified By : Static
Deleted By : External
Class Mandatory: true

Table 692: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The reference to the System
PartComponent		CIM_LogicalDevice	The reference to the FCPort

8.2.6.6.9 Related Standards

Table 693: Related Standards for Switch

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.6.7 Switch Configuration Data Subprofile

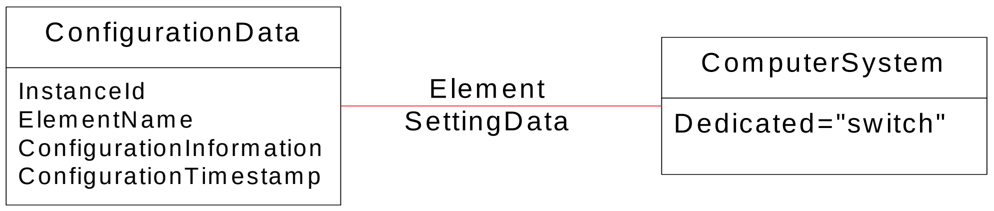
8.2.6.7.1 Description

This subprofile describes the ability to retrieve a configuration from a switch and latter apply that configuration back on the switch (similar to an image backup and restoration of a computer system).

The profile only has three classes providing all the functionality. When a client needs to obtain a snapshot of the switch configuration, he enumerates ConfigurationData which will return the current configuration with the timestamp set appropriately.

When the client wants to apply a configuration, he creates and instance of ConfigurationData and calls the method ApplyConfiguration() on the instance containing the property ConfigurationInformation which is to be applied to the switch.

Figure 97: Switch Configuration Data Instance Diagram



8.2.6.7.2 Durable Names and Correlatable IDs of the Profile

Not defined in this standard.

8.2.6.7.3 Instrumentation Requirements

Not defined in this standard.

8.2.6.7.4 Health and Fault Management

None

8.2.6.7.5 Cascading Considerations

None

8.2.6.7.6 Methods of this Profile

8.2.6.7.6.1 ApplyConfiguration

This method applies the configuration data to the switch. The data in the instance's ConfigurationInformation property is used as the configuration to apply. Note that it is not necessary for the element to be associated with the ConfigurationData instance at the time that this method is called.

```
uint32 ApplyConfiguration (
     boolean ValidateOnly,
     uint16 TypeOfConfiguration
     CIM_ManagedElement REF ManagedElement);
```

8.2.6.7.7 Client Considerations and Recipes

8.2.6.7.7.1 Get Switch Configuration

```
// DESCRIPTION
//
```

```
// This recipe describes how to retrieve Switch configuration data.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS
// 1. A reference to the Switch whose configuration data to retrieve is known
// and defined in the variable $Switch->.

// MAIN
// Step 1. Retrieve the configuration of the Switch.
$ConfigData[] = Associators($Switch->,
     “CIM_ElementSettingData”,
     “CIM_ConfigurationData”,
     “ManagedElement”,
     “SettingData”,
     false,
     false,
     {“ConfigurationInformation”, “ConfigurationTimestamp”})
if ($ConfigData[] == null || $ConfigData[].length != 1) {
    <ERROR! The required Switch configuration data is not available>
}
$SwitchConfig = $ConfigData[0]
```

8.2.6.7.7.2 Set Switch Configuration

```
// DESCRIPTION
//
// Set Switch Configuration
//
// PREEXISTING CONDITIONS AND ASSUMPTIONS
//
// None

Placeholder File
```

8.2.6.7.8 Registered Name and Version

Switch Configuration Data version 1.1.0

8.2.6.7.9 CIM Server Requirements

Table 694: CIM Server Requirements for Switch Configuration Data

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.7.10 CIM Elements

Table 695: CIM Elements for Switch Configuration Data

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.6.7.10.1)	Represents the Switch
CIM_ConfigurationData (8.2.6.7.10.2)	Switch Configuration Data
CIM_ElementSettingData (8.2.6.7.10.3)	Associates ConfigurationData to the switch

8.2.6.7.10.1 CIM_ComputerSystem

Represents the Switch
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 696: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	The class name
Name		string	Switch Name (WWN)
NameFormat		string	"WWN"
Dedicated		uint16[]	"Switch"

8.2.6.7.10.2 CIM_ConfigurationData

Switch Configuration Data
Created By : Extrinsic(s):
Modified By : Static
Deleted By : DeleteInstance

Class Mandatory: true

Table 697: SMI Referenced Properties/Methods for CIM_ConfigurationData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	User-friendly for configuration file.
ConfigurationInformation		uint8[]	The configuration data of the switch.
ConfigurationTimestamp		datetime	Time the configuration data was obtained
ApplyConfiguration()			Method that processes the configuration in the same instance and applies it to the switch

8.2.6.7.10.3 CIM_ElementSettingData

Associates ConfigurationData to the switch
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 698: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The reference to the ComputerSystem
SettingData		CIM_SettingData	The reference to the ConfigurationData

8.2.6.7.11 Related Standards

Table 699: Related Standards for Switch Configuration Data

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

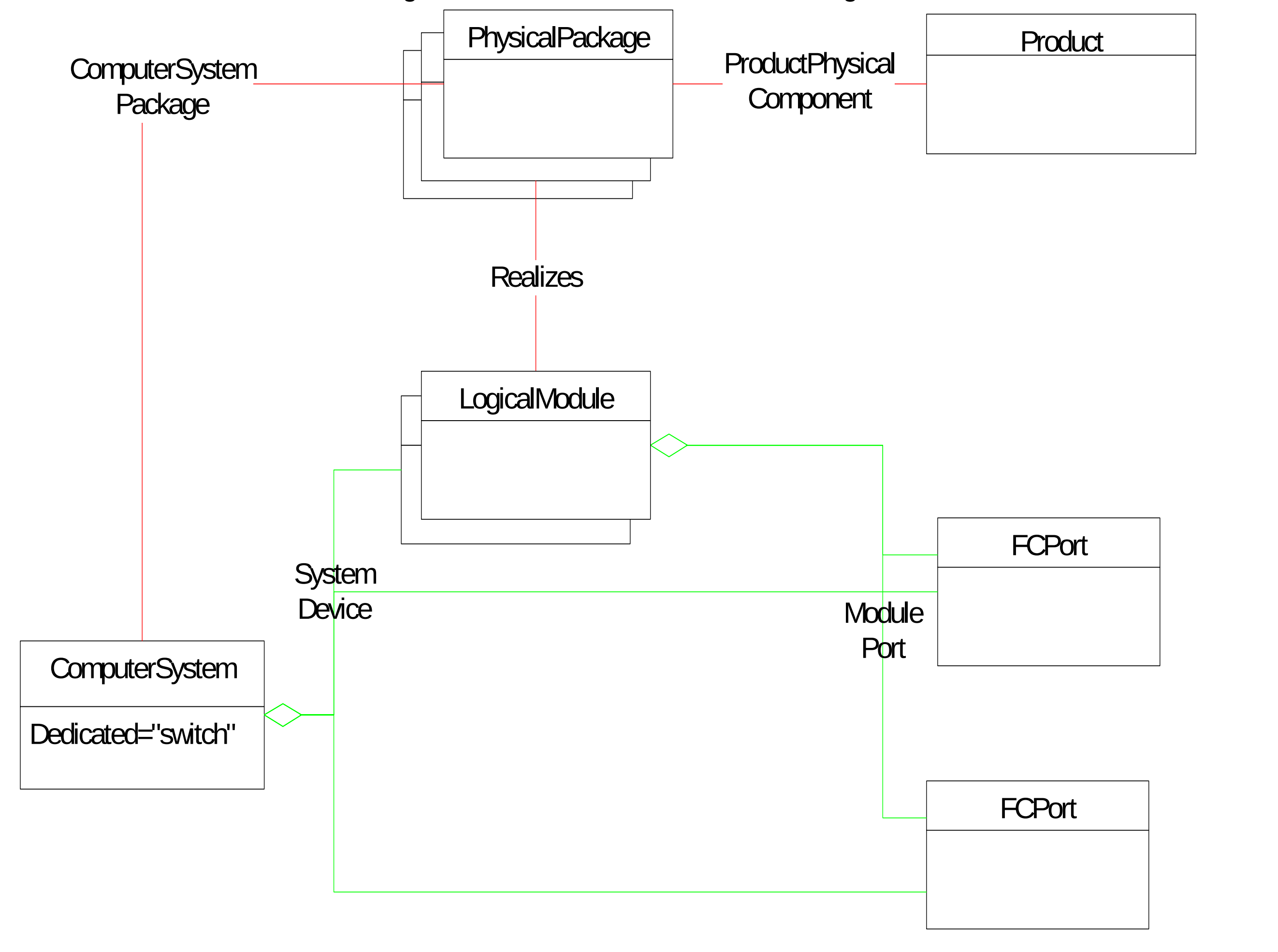
8.2.6.8 Blades Subprofile

8.2.6.8.1 Description

This subprofile describes how blades in a director class switch can be discovered and managed.

Instance Diagram

Figure 98: Switch Blade Instance Diagram



8.2.6.8.2 Health and Fault Management

None

8.2.6.8.3 Cascading Considerations

None

8.2.6.8.4 Methods of this Profile

None

8.2.6.8.5 Client Considerations and Recipes

None

8.2.6.8.6 Registered Name and Version

Blades version 1.1.0

8.2.6.8.7 CIM Server Requirements

Table 700: CIM Server Requirements for Blades

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.8.8 CIM Elements

Table 701: CIM Elements for Blades

Element Name	Description
Mandatory Classes	
CIM_LogicalModule (8.2.6.8.8.1)	The Blade
CIM_ModulePort (8.2.6.8.8.2)	Associates the LogicalModule to the FCPort
CIM_PhysicalPackage (8.2.6.8.8.3)	The physical package that the LogicalModule is contained within
CIM_Realizes (8.2.6.8.8.6)	Associates the LogicalModule to its PhysicalPackage
CIM_SystemDevice (8.2.6.8.8.7)	Associates the LogicalModule to the ComputerSystem representing the Switch
Optional Classes	
CIM_Product (8.2.6.8.8.4)	The product information for the Blade
CIM_ProductPhysicalComponent (8.2.6.8.8.5)	Associates the Product to the PhysicalPackage
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalModule	Creation of an Creation LogicalModule instance. This indication is recommended.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalModule	Deletion of an LogicalModule instance. This indication is recommended.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalModule AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Change in status of LogicalModule. This indication is recommended.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalModule AND SourceInstance.CIM_LogicalModule::OperationalStatus <> PreviousInstance.CIM_LogicalModule::OperationalStatus	CQL - Change in status of LogicalModule. This indication is recommended.

8.2.6.8.8.1 CIM_LogicalModule

The Blade

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 702: SMI Referenced Properties/Methods for CIM_LogicalModule

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque
ElementName		string	
OperationalStatus		uint16[]	

Table 702: SMI Referenced Properties/Methods for CIM_LogicalModule

Property	Flags	Type	Description & Notes
ModuleNumber		uint16	

8.2.6.8.8.2 CIM_ModulePort

Associates the LogicalModule to the FCPort

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 703: SMI Referenced Properties/Methods for CIM_ModulePort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_LogicalModule	The reference to the Computer System representing the Switch
PartComponent		CIM_NetworkPort	The reference to the FCPort

8.2.6.8.8.3 CIM_PhysicalPackage

The physical package that the LogicalModule is contained within.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 704: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
Manufacturer		string	
Model		string	
Optional Properties/Methods			
ElementName		string	
Name		string	
SerialNumber		string	
Version		string	
PartNumber		string	

8.2.6.8.8.4 CIM_Product

The product information for the Blade

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

Class Mandatory: false

Table 705: SMI Referenced Properties/Methods for CIM_Product

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	Commonly used Product name.
IdentifyingNumber		string	Product identification such as a serial-number.
Vendor		string	The manufacturer or the OEM.
Version		string	Product version information.
ElementName		string	User Friendly name. Suggested use isVendor, Version and product name.

8.2.6.8.8.5 CIM_ProductPhysicalComponent

Associates the Product to the PhysicalPackage. This is necessary to link the Product information to the Blade.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 706: SMI Referenced Properties/Methods for CIM_ProductPhysicalComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_Product	
PartComponent		CIM_PhysicalElement	

8.2.6.8.8.6 CIM_Realizes

Associates the LogicalModule to its PhysicalPackage

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 707: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	The reference to the PhysicalPackage.
Dependent		CIM_LogicalDevice	The reference to the LogicalModule representing the Blade.

8.2.6.8.8.7 CIM_SystemDevice

Associates the LogicalModule to the ComputerSystem representing the Switch

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

Class Mandatory: true

Table 708: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The reference to the Computer System representing the Switch
PartComponent		CIM_LogicalDevice	The reference to the LogicalModule

8.2.6.8.9 Related Standards

Table 709: Related Standards for Blades

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.6.9 Extender Profile

8.2.6.9.1 Description

A FC Extender is a logical entity representing an inter-switch link consisting of two FC Extender Node devices and the Network pipes that connect them.

A FC Extender is used to connect two Fabrics across a LAN, MAN, WAN, or other network communications media.

A FC Extender Node is a physical device that converts Fibre Channel protocol for transmission over different network communication technologies.

The domain of the Extender Group is defined by Network, which is a subclass of AdminDomain.

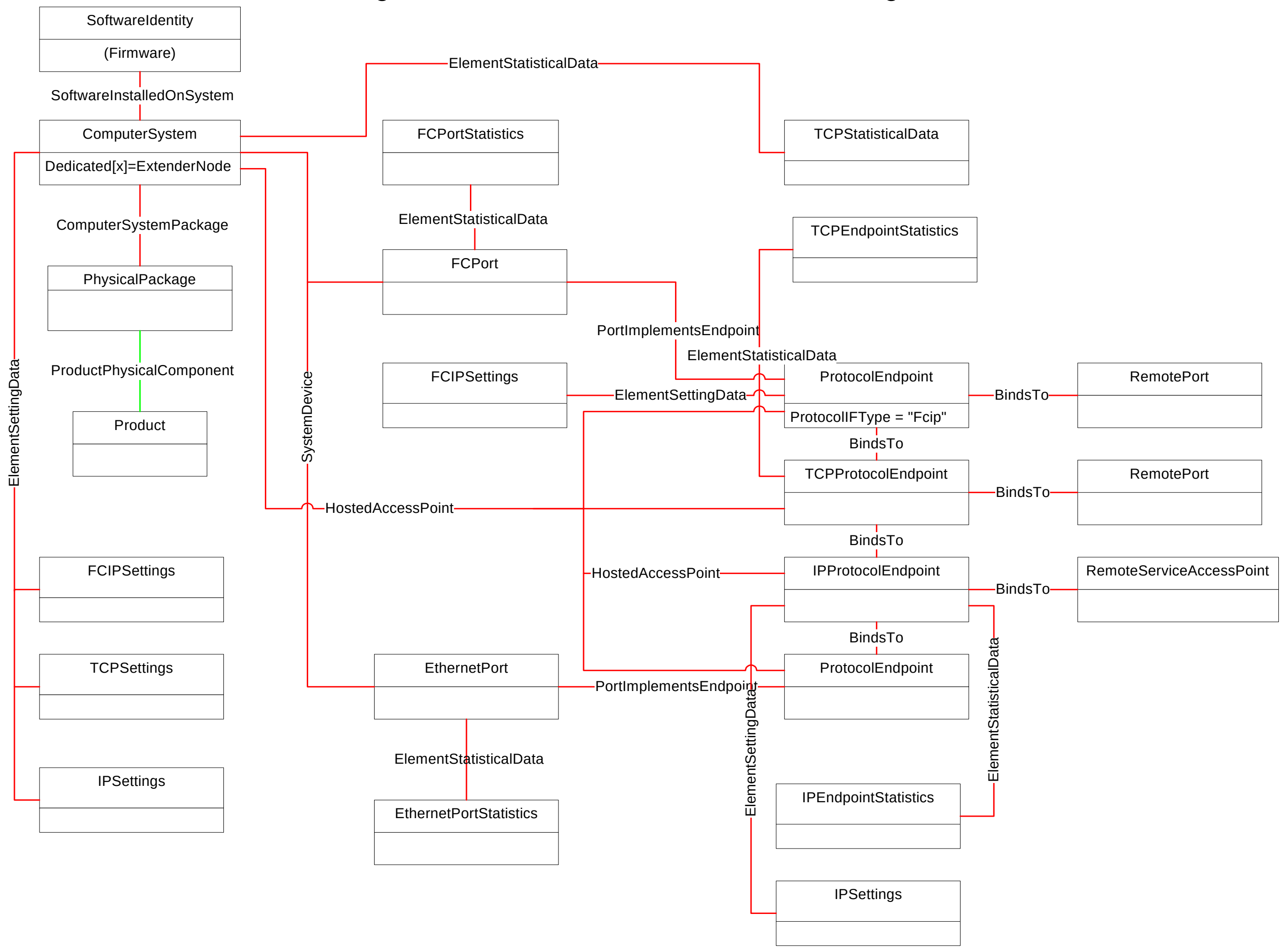
8.2.6.9.1.1 FC Extender Node Topology Classes

The ComputerSystem class is the core of the model. It is identified as an Extender node by the dedicated attribute being set to ExtenderNode.

The TCPSettings and IPSettings classes represent the global configuration of the FC Extender transport layer.

The Port group of classes contains the following classes: FCPort, and EthernetPort. The FCPort class represents the connection of a FC Extender to a SAN. This class connects to other FCPort classes to represent Fibre channel connections. This class could be replaced with other port types to represent SANs based on other interconnect technology. The EthernetPort class represents an Ethernet link between FC Extender nodes.

Figure 99: FC Extender Node Instance Diagram



8.2.6.9.1.2 FC Extender Node Network Connectivity Classes

Each FC Extender node local ProtocolEndpoint (e.g., FCProtocolEndpoints, TCPProtocolEndpoints) has a BindsTo dependency on a RemotePort that describes access or addressing information to a remote ProtocolEndpoint for a specific connection.

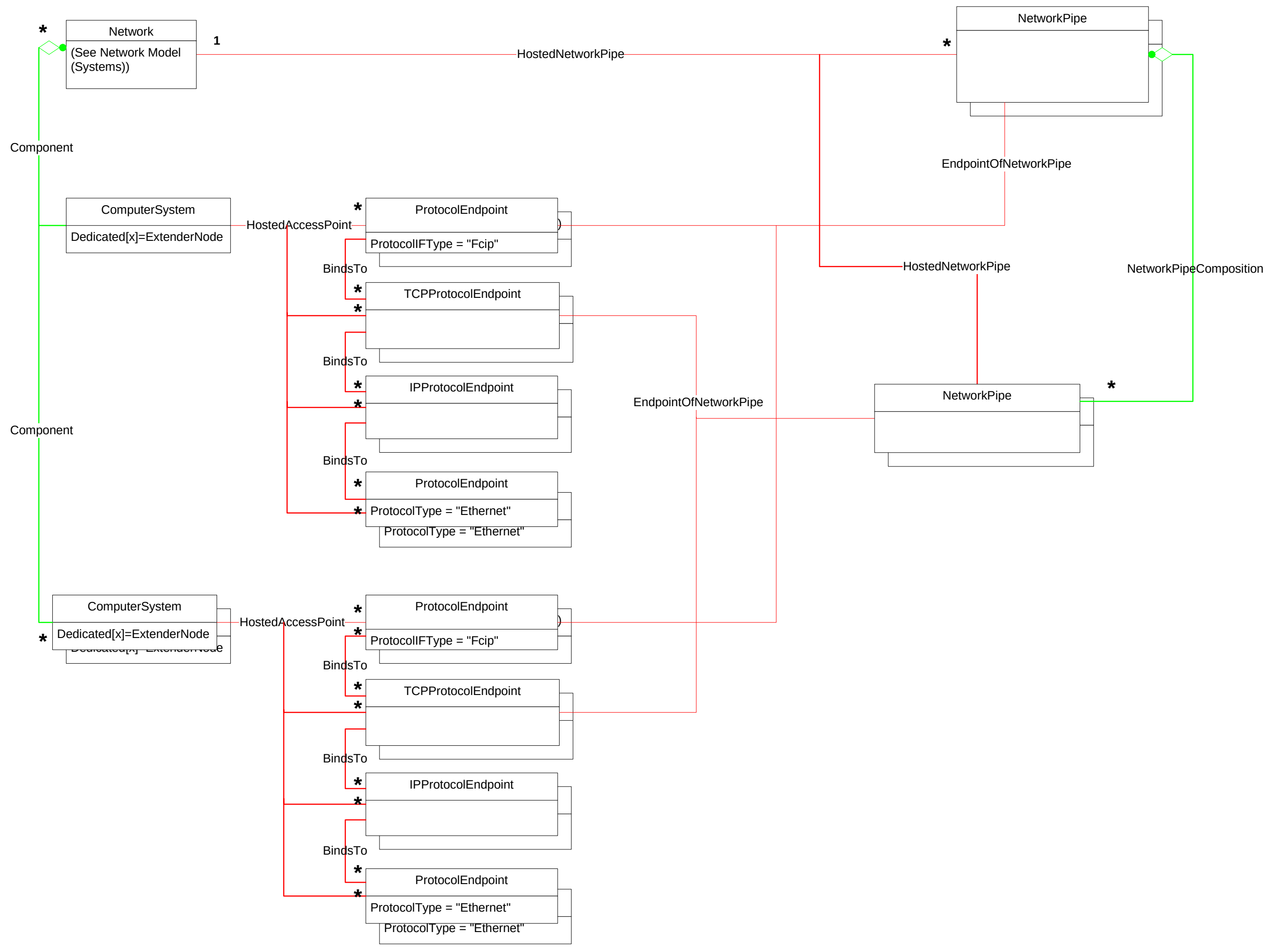
The Extender node represents ProtocolEndpoints dependencies (e.g., FC ProtocolEndpoint on TCPProtocolEndpoints, TCPProtocolEndpoint on IPProtocolEndpoint, IPProtocolEndpoint on EthernetProtocolEndpoint) with a BindsTo association.

8.2.6.9.1.3 FC Extender Group Network Connectivity Classes

A FC Extender connection is represented by a NetworkPipe class associated with FCProtocolEndpoints. A FCExtender Network class groups multiple NetworkPipes.

The NetworkPipe between FCProtocolEndpoints is composed of lower-level TCP network pipes.

Figure 100: FC Extender Group Instance Diagram



8.2.6.9.2 Health and Fault Management

None

8.2.6.9.3 Cascading Considerations

None

8.2.6.9.4 Supported Subprofiles and Packages

None

8.2.6.9.5 Methods of this Profile

None

8.2.6.9.6 Client Considerations and Recipes

8.2.6.9.6.1 Extender Connectivity Settings

```
// Description
// Collecting settings
// of Extender node connectivity elements participating in the Extender
// of interest.
// $extenderNodeFCIPSettings(fcip protocol endpoint settings)
// $extenderNodeTCPSettings (transport layer settings)
// $extenderIPSettings (ip protocol endpoint settings)
// PREEXISTING CONDITIONS AND ASSUMPTIONS
// The Extender fcip ProtocolEndpoint has been previously
// identified and defined in the $fcipProtocolEndpoint-> variable
// 1. Get ComputerSystem associated with $fcipProtocolEndpoint

$extenderNodes[] = Associators(
     $fcipProtocolEndpoint->,
     CIM_HostedAccessPoint,
     CIM_ComputerSystem,
     Antecedent,
     Dependent,
     false,
     false,
     [Dedicated])
if (contains(23, $extenderNodes[0].Dedicated))
{
     #extenderNodeAccessA = true
     $extenderNode-> = $extenderNodes[0].getObjectPath()
}
if(#extenderNodeAccessA)
{
//2. Get fcip protocol endpoint

 // find FCIP Settings
     $fcipSettings[]= Associators(
        $fcipProtocolEndpoint,
        CIM_ElementSettingData,
        CIM_FCIPSettings,
        ManagedElement,
        SettingData,
        false,
        false,
        null)
     if ($fcipSettings[].length != 0)
        $extenderNodeFCIPSettings = $fcipSettings[0]


//3. Get transport layer settings

     $tcpSettings[]= Associators(
        $extenderNode->,
        CIM_ElementSettingData,
        CIM_TCPSettings,
        ManagedElement,
        SettingData,
        false,
        false,
        null)
     if ($tcpSettings[].length != 0)
        $extenderNodeTCPSettings = $tcpSettings[0]


//4. Find TCPProtocolEndpoint bound to the extender fcip ProtocolEndpoint


     $tcpProtocolEndpoint->[]= AssociatorNames(
        $fcipProtocolEndpoint->,
        CIM_BindTo,
        CIM_TCPProtocolEndpoint,
        Dependent,
        Antecedent,
        false,
        false,
        null))

//5. Find IPProtocolEndpoint bound to the extender tcp ProtocolEndpoint


     $ipProtocolEndpoints->[]= AssociatorNames(
        $tcpProtocolEndpoint->,
        CIM_BindTo,
        CIM_IPProtocolEndpoint,
        Dependent,
        Antecedent,
        false,
        false,
        null))
        $ipProtocolEndpoint-> = $ipProtocolEndpoints->[0]

//6. Find IPProtocolEndpoint settings


     $ipSettings[]= Associators(
        $ipProtocolEndpoint->,
        CIM_ElementSettingData,
        CIM_IPSettings,
        ManagedElement,
        SettingData,
        false,
        false,
        null)
     if ($ipSettings[].length != 0)
```

```
                $extenderNodeIPSettings = $ipSettings[0]

        }
```

8.2.6.9.6.2 Extender Connective Statistics

```
// Description
// Collecting statistical data
// of Extender node conectivity elements participating in the Extender
// of interest.
// $extenderNodeTCPStatisticalData (transport layer stats)
// $extenderIPEndpointStatistics (IP protocol endpoint stats)
// PREEXISTING CONDITIONS AND ASSUMPTIONS
// The Extender fcip ProtocolEndpoint has been previously
// identified and defined in the $fcipProtocolEndpoint-> variable
// 1. Get ComputerSystem associated with $fcipProtocolEndpoint

$extenderNodes[] = Associators(
     $fcipProtocolEndpoint->,
     CIM_HostedAccessPoint,
     CIM_ComputerSystem,
     Antecedent,
     Dependent,
     false,
     false,
     [Dedicated])
if (contains(23, $extenderNodes[0].Dedicated))
{
     #extenderNodeAccess = true
     $extenderNode-> = $extenderNodes[0].getObjectPath()
}
if(#extenderNodeAccess)
{
//2. Get transport layer statistics

     $tcpStatistics[] = Associators(
            $extenderNode->,
            CIM_ElementStatisticalData,
            CIM_TCPStatisticalData,
            ManagedElement,
            Stats,
            false,
            false,
            null))
     $extenderNodeTCPStatisticalData = $tcpStatistics[0]

//3. Find TCPProtocolEndpoint bound to the extender fcip ProtocolEndpoint


     $tcpProtocolEndpoint->[]= AssociatorNames(
         $fcipProtocolEndpoint->,
         CIM_BindTo,
         CIM_TCPProtocolEndpoint,
         Dependent,
         Antecedent,
         false,
         false,
         null))

//4. Find IPProtocolEndpoint bound to the extender tcp ProtocolEndpoint


     $ipProtocolEndpoint->[]= AssociatorNames(
         $tcpProtocolEndpoint->,
         CIM_BindTo,
         CIM_IPProtocolEndpoint,
         Dependent,
         Antecedent,
         false,
         false,
         null))

//5. Find IPProtocolEndpoint statistics


     $ipStatistics[]= Associators(
         $ipProtocolEndpoint->,
         CIM_ElementStatisticalData,
         CIM_IPEndpointStatistics,
         ManagedElement,
         Stats,
         false,
         false,
         null)

     $extenderIPEndpointStatistics = $ipStatistics[0]

}
```

8.2.6.9.6.3 Extender Port Group Information

```
// Description
// Collecting configuration and statistical data
// of Extender node ports participating in the Extender
// of interest.
// $extenderNodeFCPort (connected to a switch)
// $extenderNodeFCPortStatistics
// $extenderNodeEthernetPort (connected to a peer Extender node)
// $extenderNodeEthernetStatistics
// PREEXISTING CONDITIONS AND ASSUMPTIONS
// The Extender fcip ProtocolEndpoint has been previously
// identified and defined in the $fcipProtocolEndpoint-> variable

// 1. Get ComputerSystem associated with $fcipProtocolEndpoint

$extenderNodes[] = Associators(
     $fcipProtocolEndpoint->,
     CIM_HostedAccessPoint,
     CIM_ComputerSystem,
     Antecedent,
     Dependent,
     false,
     false,
     [Dedicated])
if (contains(23, $extenderNodes[0].Dedicated))
{
     #extenderNodeAccess = true
}
if(#extenderNodeAccess)
{
     // 2. Get FC port

     $fcPorts[] = Associators(
        $fcipProtocolEndpoints->,
        CIM_DeviceSAPImplementation,
        CIM_FCPort,
        Dependent,
        Antecedent,
        false,
        false,
        null)
     $extenderNodeFCPort = $fcPorts[0]

     // 2. Get FC port statistics

     $fcPortStatistics->[] = Associators(
        $extenderNodeFCPort.getObjectPath(),
        CIM_ElementStatisticalData,
        CIM_FCPortStatistics,
        ManagedElement,
        Stats,
        false,
        false,
        null))
        $extenderNodeFCPortStatistics = $fcPortsStatistics[0]

     //3. Find TCPProtocolEndpoint bound to the extender FCIP ProtocolEndpoint

     $tcpProtocolEndpoints->[]= AssociatorNames(
        $fcipProtocolEndpoint->,
        CIM_BindTo,
        CIM_TCPProtocolEndpoint,
        Dependent,
        Antecedent,
        false,
        false,
        null))
     // at least one should exist
     $tcpProtocolEndpoint->=$tcpProtocolEndpoints->[0]


//4. Find IPProtocolEndpoint bound to the extender TCP ProtocolEndpoint

     $ipProtocolEndpoint->[]= AssociatorNames(
        $tcpProtocolEndpoint->,
        CIM_BindTo,
        CIM_IPProtocolEndpoint,
        Dependent,
        Antecedent,
        false,
        false,
        null))
     $ipProtocolEndpoint->=$ipProtocolEndpoints->[0]

//5. Get Ethernet port

     $ethernetPorts[] = Associators(
        $ipProtocolEndpoints->,
        CIM_DeviceSAPImplementation,
        CIM_EthernetPort,
        Dependent,
        Antecedent,
        false,
        false,
        null)
     $extenderNodeEthernetPort = $ethernetPorts[0]

//6. Get Ethernet port statistics

     $ethernetPortStatistics->[] = Associators(
        $extenderNodeEthernetPort.getObjectPath(),
        CIM_ElementStatisticalData,
        CIM_EthernetStatistics,
        ManagedElement,
        Stats,
        false,
        false,
        null))
     $extenderNodeEthernetPortStatistics = $ethernetPortsStatistics[0]


}
```

8.2.6.9.6.4 Extender Topology Mapping

```
// This recipe describes how to build a topology graph of a fabric.
//
// 1. Identifies all the Switches and adds their objects paths and the
// object paths of the FC Ports belonging to these Switches to the $nodes
// array
//
// 2. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Switch and a FC Port), setting its
// GroupComponent and PartComponent. Adds the object path of the
// Association to the $links array
//
// 3. Creates a map of all connected FC Ports (i.e., belonging to Switches
// that are ISL'd together and to Host HBAs and Storage System Front End
// Controllers)
//
// In this map, the FC Ports (i.e., the ones that are connected) are
// cross-connected.
//
// e.g., For a pair of FC Ports, one belonging to a Switch and the other
// belonging to a Host (HBA), the map indexed by the Switch Port WWN returns
// the Host (HBA) FC Port object path and the map indexed by the Host (HBA)
// FC Port WWN returns the Switch FC Port object path.
//
```

```
// Similar relationship exists between the pairs of FC Ports where one
// belongs to a Switch and the other belonging belongs to a Storage System
// Front End Controller and for FC Ports each of which belongs to a Switch.
//
// 4. Identifies all the Hosts and adds their objects paths to the $nodes
// array. Note that the object paths of the FC Ports (HBA Ports) belonging
// to these Hosts are already added to the $nodes array in step-3.
//
// 5. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Host and a FC Port), setting its
// GroupComponent and PartComponent. Adds the object path of the Association
// to the $links array.
//
// 6. Identifies all the Storage Systems and adds their objects paths to the
// $nodes array.
// Note that the object paths of the FC Ports (i.e., Front End Controller
// FC Ports) belonging to these Storage Systems are already added to the
// $nodes array in step-3.
//
// 7. Creates a suitable Association instance (e.g. a SystemDevice
// Association instance between a Storage System and a FC Port), setting
// its GroupComponent and PartComponent. Adds the object path of the
// Association to the $links array.

// DESCRIPTION
// Create a map of how elements in a SAN are connected together via
// Fibre-ChannelFC ports.
//
// The map is built in array $attachedFcPorts->[], where the index is a
// WWN of any device port on the SAN, and the value at that index is
// the object path of the connected Switch or HBA or Storage System FC port.
//
// First find all the switches in a SAN. Get all the FC Ports for each
// switch and get the Attached FC Ports for each Switch FC Port. Save these
// device FC ports in the map described above.

// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1. All agents/namespaces supporting Fabric Profile previously identified
// using SLP. Do this for each CIMOM supporting Fabric Profile

switches[] = enumerateInstances("CIM_ComputerSystem", true, false, true, true,
null)
for #i in $switches[]
{
    if (!contains(5, $switches[#i].Dedicated))
        continue

    // only process switches, not other computer systems
    // Add the switch to the $nodes array

    $nodes.addIfNotAlreadyAdded ($switches[#i].getObjectPath();

    // Get all the SystemDevice associations between this switch and its
    // FC Ports

    $sysDevAssoc[] = ReferenceNames($switches[#i],
                                “CIM_FCPort”,
                                “GroupComponent”);

    // Add these associations to the $links array

    for #a in $sysDevAssoc->[]
    $links.addIfNotAlreadyAdded ($sysDevAssoc->[#a];

    $fcPorts->[] = AssociatorNames(
        $switches[#i].getObjectPath(),
        “CIM_SystemDevice”,
        “CIM_FCPort”,
        “GroupComponent”,
        “PartComponent”)
    for #j in $fcPorts->[]
    {

        // Add the FC Port in $nodes array

        $nodes.addIfNotAlreadyAdded (fcPorts->[#j];

        $protocolEndpoints->[] = AssociatorNames(
            fcPorts->[#j],
            “CIM_DeviceSAPImplementation”,
            “CIM_ProtocolEndpoint”,
            “Antecedent”,
        “Dependent”);

        // NOTE - It is possible for this collection to be empty (i.e., ports
        // that are not connected). It is possible for this collection to
        // have more than one element (loops attached to a switch port is the
        // most common example).

        if ($protocolEndpoints->[].length == 0)
            continue

        // Add the Protocol End Point to the nodes array.
        // Currently this recipe is designed to only save one
        // ProtocolEndpoint.

        $nodes.addIfNotAlreadyAdded (protocolEndpoints[0]);

        // Add the associations between the fcPort and the Protocol end point
        // to the links array

        $devSAPImplassoc[]  = ReferenceNames($fcPorts->[#j],
                                     “CIM_ProtocolEndpoint”,
                                     null);
        for #a in $devSAPImplassoc->[]
            $links.addIfNotAlreadyAdded ($devSAPImplassoc->[#a];

        $attachedProtocolEndpoints->[] = AssociatorNames(
            $protocolEndpoints->[0],
            “CIM_ActiveConnection”,
            “CIM_ProtocolEndpoint”,
            null, null)

        // Add the Attached Protocol End Point to the nodes array

        $nodes.addIfNotAlreadyAdded (attachedProtocolEndpoints->[0]);

        // Add the associations between the Protocol end point and the
        // Attached protocol endpoint to the links array

        $actConnassoc[]  = ReferenceNames($protocolEndpoint->[#0],
                                  “CIM_ActiveConnection”,
                                   null);
        for #a in $actConnassoc->[]
            $links.addIfNotAlreadyAdded ($actConnassoc->[#a];

        // NOTE: role & resultRole are null as the direction of the
        // association is not dictated by the specification

        // $attachedFcPort is either a device FC port or an ISL’d switch FC
        // port from another switch. We store this result is stored (i.e.,
        // which device FC Port is connected // to which switch FC Port) in
        // a suitable data structure for subsequent correlation to ports
        // discovered on devices.

        for #k in $attachedProtocolEndpoints->[]
        {
            $attachedFcPorts->[] = Associators(
                $attachedProtocolEndpoints->[#k],
                “CIM_DeviceSAPImplementation”,
                “CIM_FCPort”,
                “Dependent”,
                “Antecedent”,
                false,
                false,
                [“PermanentAddress”])
           $attachedFcPort = $attachedFcPorts[0] // Exactly one member guaranteed
by model

            // Add the attached FC Port to the $nodes array
            if $attachedFcPort != null
                $nodes.addIfNotAlreadyAdded ($attachedFcPort);
        }
    }
}
```

8.2.6.9.7 Registered Name and Version

Extender version 1.1.0

8.2.6.9.8 CIM Server Requirements

Table 710: CIM Server Requirements for Extender

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.6.9.9 CIM Elements

Table 711: CIM Elements for Extender

Element Name	Description
Mandatory Classes	
CIM_BindsTo (8.2.6.9.9.1)	Associates Extender Node ProtocolEndpoints from different layers in the protocol stack
CIM_Component (8.2.6.9.9.2)	Aggregates Extender Nodes in the Network that represents the group of Extenders
CIM_ComputerSystem (8.2.6.9.9.3)	Represents the Extender Node
CIM_ComputerSystemPackage (8.2.6.9.9.4)	Associated PhysicalPackage to the ComputerSystem (Extender)
CIM_ElementSettingData (8.2.6.9.9.5)	Associates SettingData to Extender Node orProtocolEndpoints
CIM_ElementStatisticalData (8.2.6.9.9.6)	Associates StatisticalData to Extender Node or ProtocolEndpoints
CIM_EndpointOfNetworkPipe (8.2.6.9.9.7)	
CIM_EthernetPort (8.2.6.9.9.8)	
CIM_EthernetPortStatistics (8.2.6.9.9.9)	
CIM_FCIPSettings (8.2.6.9.9.10)	Defines FCIP settings for a group of ProtocolEndpoints (ProtocolIFType - "Fcip") which belongs to the ComputerSystem (Extender Node)
CIM_FCPort (8.2.6.9.9.11)	
CIM_FCPortStatistics (8.2.6.9.9.12)	
CIM_HostedAccessPoint (8.2.6.9.9.13)	Associates the ProtocolEndpoint to the ComputerSystem or Network
CIM_HostedNetworkPipe (8.2.6.9.9.14)	Associates NetworkPipe to the Network
CIM_IPEndpointStatistics (8.2.6.9.9.15)	
CIM_IPProtocolEndpoint (8.2.6.9.9.16)	
CIM_IPSettings (8.2.6.9.9.17)	Defines IP settings for a group of IPProtocolEndpoints which belongs to the ComputerSystem
CIM_Network (8.2.6.9.9.18)	Network represents a network connectivity domain. It groups NetworkPipes.
CIM_NetworkPipe (8.2.6.9.9.19)	NetworkPipe represents state, configuration of a connection between endpoints in the context of a Network
CIM_NetworkPipeComposition (8.2.6.9.9.20)	
CIM_PortImplementsEndpoint (8.2.6.9.9.21)	
CIM_ProtocolEndpoint (8.2.6.9.9.22)	ProtocolEndpoint shall be implemented when an ActiveConnection or NetworkPipe exists. It may be implemented if no ActiveConnection or NetworkPipe exists.
CIM_RemotePort (8.2.6.9.9.23)	
CIM_RemoteServiceAccessPoint (8.2.6.9.9.24)	
CIM_SystemDevice (8.2.6.9.9.25)	Associated FCPort and EthernetPort to the ComputerSystem
CIM_TCPEndpointStatistics (8.2.6.9.9.26)	Opaque

Table 711: CIM Elements for Extender

Element Name	Description
CIM_TCPProtocolEndpoint (8.2.6.9.9.27)	
CIM_TCPSettings (8.2.6.9.9.28)	Defines TCP settings for a group of TCPProtocolEndpoints which belongs to the ComputerSystem
CIM_TCPStatisticalData (8.2.6.9.9.29)	
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Creation of a ComputerSystem instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance CIM_ComputerSystem	Deletion of a computer system instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.Operationalstatus ** PreviousInstance.Operationalstatus	Deprecated WQL - Change of OperationalStatus for a Computer System
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::Operationalstatus ** PreviousInstance.CIM_ComputerSystem::Operationalstatus	CQL - Change of OperationalStatus for a Computer System
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.Operationalstatus ** PreviousInstance.Operationalstatus	Deprecated WQL - Change of OperationalStatus for a Computer System
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::Operationalstatus ** PreviousInstance.CIM_ComputerSystem::Operationalstatus	CQL - Change of OperationalStatus for a Computer System

8.2.6.9.9.1 CIM_BindsTo

Associates Extender Node ProtocolEndpoints from different layers in the protocol stack
Class Mandatory: true

Table 712: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolEndpoint	TCPProtocolEndpoint, IPProtocolEndpoint
Dependent		CIM_ServiceAccessPoint	ProtocolEndpoint.ProtocolIFType=""Fcip", TCPProtocolEndpoint

8.2.6.9.9.2 CIM_Component

Aggregates Extender Nodes in the Network that represents the group of Extenders

Class Mandatory: true

Table 713: SMI Referenced Properties/Methods for CIM_Component

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ManagedElement	Network ref.
PartComponent		CIM_ManagedElement	ComputerSystem ref.

8.2.6.9.9.3 CIM_ComputerSystem

Represents the Extender Node
Class Mandatory: true

Table 714: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	IP Address
NameFormat		string	IP Address
OperationalStatus		uint16[]	Status of Computer System.
Dedicated		uint16[]	ExtenderNode
Optional Properties/Methods			
ElementName		string	User-friendly name
OtherIdentifyingInfo		string[]	DNS name
IdentifyingDescriptions		string[]	Fully qualified domain name

8.2.6.9.9.4 CIM_ComputerSystemPackage

Associated PhysicalPackage to the ComputerSystem (Extender)
Class Mandatory: true

Table 715: SMI Referenced Properties/Methods for CIM_ComputerSystemPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalPackage	
Dependent		CIM_ComputerSystem	

8.2.6.9.9.5 CIM_ElementSettingData

Associates SettingData to Extender Node orProtocolEndpoints
Class Mandatory: true

Table 716: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	ComputerSystem orProtocolEndpoint
SettingData		CIM_SettingData	
IsDefault		uint16	

Table 716: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
IsCurrent		uint16	

8.2.6.9.9.6 CIM_ElementStatisticalData

Associates StatisticalData to Extender Node or ProtocolEndpoints
Class Mandatory: true

Table 717: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	ComputerSystem orProtocolEndpoint
Stats		CIM_StatisticalData	

8.2.6.9.9.7 CIM_EndpointOfNetworkPipe

Class Mandatory: true

Table 718: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	
Dependent		CIM_NetworkPipe	ProtocolEndpoint.ProtocolIFType=""Fcip", TCPProtocolEndpoint

8.2.6.9.9.8 CIM_EthernetPort

Class Mandatory: true

Table 719: SMI Referenced Properties/Methods for CIM_EthernetPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
Speed		uint64	
MaxSpeed		uint64	
PortType		uint16	Supported port mode 10BaseT,10-100BaseT, 100BaseT, 1000BaseT, etc.
PortNumber		uint16	System level port or bus identification number
NetworkAddresses		string[]	MAC addresses
LinkTechnology		uint16	Ethernet

Table 719: SMI Referenced Properties/Methods for CIM_EthernetPort

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
ElementName		string	User-friendly name

8.2.6.9.9.9 CIM_EthernetPortStatistics

Class Mandatory: true

Table 720: SMI Referenced Properties/Methods for CIM_EthernetPortStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ElementName		string	
BytesTransmitted		uint64	
BytesReceived		uint64	
PacketsTransmitted		uint64	
PacketsReceived		uint64	
SymbolErrors		uint32	
CarrierSenseErrors		uint32	
Optional Properties/Methods			
StatisticTime		datetime	
AlignmentErrors		uint32	
FCSErrors		uint32	
SingleCollisionFrames		uint32	
MultipleCollisionFrames		uint32	
DeferredTransmissions		uint32	
LateCollisions		uint32	
ExcessiveCollisions		uint32	
InternalMACTransmitErrors		uint32	
InternalMACReceiveErrors		uint32	
FrameTooLongs		uint32	
ResetSelectedStats()			

8.2.6.9.9.10 CIM_FCIPSettings

Defines FCIP settings for a group of ProtocolEndpoints (ProtocolIFType - "Fcip") which belongs to the Computer-System (Extender Node)
Class Mandatory: true

Table 721: SMI Referenced Properties/Methods for CIM_FCIPSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
ConnectionUsageFlags		uint16	

Table 721: SMI Referenced Properties/Methods for CIM_FCIPSettings

Property	Flags	Type	Description & Notes
SpecialFrameTimeout		uint32	
KeepAliveTimeout		uint32	
Optional Properties/Methods			
ElementName		string	User-friendly name. In addition, it can be used as a index property for a search or query.

8.2.6.9.9.11 CIM_FCPort

Class Mandatory: true

Table 722: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
OperationalStatus		uint16[]	
Speed		uint64	Speed of zero represents a link notestablished. 1Gb is 1062500000 bps. 2Gb is 2125000000 bps. 4Gb is 4250000000 bps. 10Gb single channel variants are 10518750000 bps. 10Gb four channel variants are 12750000000 bps. This is the raw bit rate.
MaxSpeed		uint64	The max speed of the Port in Bits per second using the same algorithm as Speed.

Table 722: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
PortType		uint16	FC-GS Port.Type The specific mode currently enabled for the Port. The values: "N" = Node Port, "NL" = Node Port supporting FC arbitrated loop, "E" = Expansion Port connecting fabric elements (for example, FC switches), "F" = Fabric (element) Port, "FL" = Fabric (element) Port supporting FC arbitrated loop, and "B" = Bridge Port. PortTypes are defined in the ANSI INCITS FC-GS standards. When set to 1 ("Other"), the related property OtherPortType contains a string description of the port's type. PortType is defined to force consistent naming of the 'type' property in subclasses and to guarantee unique enum values for all instances of NetworkPort. A range of values, DMTF_Reserved, has been defined that allows subclasses to override and define their specific port types. Vendor Reserved = 16000..65535 can be used if the PortType is not one already defined in the above enumerations and a vendor subclass is defined specifying the appropriate value and valuemap.
PortNumber		uint16	System level port or busidentification number
PermanentAddress		string	For FibreChannel, it is the Fibre Channel Port WWN.
LinkTechnology		uint16	FC
SupportedCOS		uint16[]	
SupportedMaximumTransmissionUnit		uint64	
Optional Properties/Methods			
ElementName		string	User-friendly Name
ActiveCOS		uint16[]	
ActiveMaximumTransmissionUnit		uint64	

8.2.6.9.9.12 CIM_FCPortStatistics

Class Mandatory: true

Table 723: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
StatisticTime		datetime	

Table 723: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
BytesTransmitted		uint64	
BytesReceived		uint64	
PacketsTransmitted		uint64	
PacketsReceived		uint64	
CRCErrors		uint64	
LinkFailures		uint64	
PrimitiveSeqProtocolErrCount		uint64	
LossOfSignalCounter		uint64	
InvalidTransmissionWords		uint64	
StatisticTime		datetime	
SampleInterval		datetime	
LIPCount		uint64	
NOSCount		uint64	
ErrorFrames		uint64	
DumpedFrames		uint64	
LossOfSyncCounter		uint64	
FramesTooShort		uint64	
FramesTooLong		uint64	
AddressErrors		uint64	
BufferCreditNotProvided		uint64	
DelimiterErrors		uint64	
EncodingDisparityErrors		uint64	
LinkResetsReceived		uint64	
LinkResetsTransmitted		uint64	
MulticastFramesReceived		uint64	
MulticastFramesTransmitted		uint64	
RXBroadcastFrames		uint64	
TXBroadcastFrames		uint64	
FBSYFrames		uint64	
PBSYFrames		uint64	
FRJTFrames		uint64	
PRJTFrames		uint64	
RXClass1Frames		uint64	
TXClass1Frames		uint64	
RXClass2Frames		uint64	
TXClass2Frames		uint64	
Class2FBSY		uint64	
Class2PBSY		uint64	
Class2FRJT		uint64	

Table 723: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
Class2PRJT		uint64	
RXClass3Frames		uint64	
TXClass3Frames		uint64	
Class3FramesDiscarded		uint64	
Optional Properties/Methods			
ElementName		string	
ResetSelectedStats()			

8.2.6.9.9.13 CIM_HostedAccessPoint

Associates the ProtocolEndpoint to the ComputerSystem or Network
Class Mandatory: true

Table 724: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	ProtocolEndpoint.ProtocolIFType=""Fcip", TCPProtocolEndpoint, IPProtocolEndpoint
Dependent		CIM_ServiceAccessPoint	

8.2.6.9.9.14 CIM_HostedNetworkPipe

Associates NetworkPipe to the Network
Class Mandatory: true

Table 725: SMI Referenced Properties/Methods for CIM_HostedNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Network	Network
Dependent		CIM_NetworkPipe	NetworkPipe

8.2.6.9.9.15 CIM_IPEndpointStatistics

Class Mandatory: true

Table 726: SMI Referenced Properties/Methods for CIM_IPEndpointStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
StatisticTime		datetime	
ReceivedPDUs		uint32	
ReceivedPDUHeaderErrors		uint32	
ReceivedPDUAddressErrors		uint32	
ReceivedPDUForwards		uint32	

Table 726: SMI Referenced Properties/Methods for CIM_IPEndpointStatistics

Property	Flags	Type	Description & Notes
ReceivedPDUUnknownProtocol-Errors		uint32	
ReceivedPDUDiscards		uint32	
PDUDelivers		uint32	
SentPDUs		uint32	
SentPDUDiscards		uint32	
SentPDUNoRouteErrors		uint32	
ReassemblyRequired		uint32	
ReassembledPackets		uint32	
ReassemblyFailed		uint32	
Fragmentation		uint32	
FragmentationFails		uint32	
FragmentedPDUsCreates		uint32	
RouteEntriesDiscards		uint32	
Optional Properties/Methods			
ElementName		string	User-friendly name. In addition, it can be used as a index property for a search or query.
ResetSelectedStats()			

8.2.6.9.9.16 CIM_IPProtocolEndpoint

Class Mandatory: true

Table 727: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
NameFormat		string	
IPv4Address		string	
IPv6Address		string	
SubnetMask		string	
ProtocolIFType		uint16	IPv4, IPv6, IPv4/v6
Optional Properties/Methods			
PrefixLength		uint8	

8.2.6.9.9.17 CIM_IPSettings

Defines IP settings for a group of IPProtocolEndpoints which belongs to the ComputerSystem

Class Mandatory: true

Table 728: SMI Referenced Properties/Methods for CIM_IPSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
FragmentationTimeout		uint32	
EnableIPForwarding		boolean	
Optional Properties/Methods			
ElementName		string	User-friendly name. In addition, it can be used as a index property for a search or query.

8.2.6.9.9.18 CIM_Network

Network represents a network connectivity domain. It groups NetworkPipes.
Class Mandatory: true

Table 729: SMI Referenced Properties/Methods for CIM_Network

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	IP Address
NameFormat		string	IP Address

8.2.6.9.9.19 CIM_NetworkPipe

NetworkPipe represents state, configuration of a connection between endpoints in the context of a Network
Class Mandatory: true

Table 730: SMI Referenced Properties/Methods for CIM_NetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Optional Properties/Methods			
Directionality		uint16	
OperationalStatus		uint16[]	
AggregationBehavior		uint16	
EnabledState		uint16	
RequestedState		uint16	

8.2.6.9.9.20 CIM_NetworkPipeComposition

Class Mandatory: true

Table 731: SMI Referenced Properties/Methods for CIM_NetworkPipeComposition

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_NetworkPipe	
PartComponent		CIM_NetworkPipe	
AggregationSequence		uint16	

8.2.6.9.9.21 CIM_PortImplementsEndpoint

Class Mandatory: true

Table 732: SMI Referenced Properties/Methods for CIM_PortImplementsEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_LogicalPort	
Dependent		CIM_ProtocolEndpoint	

8.2.6.9.9.22 CIM_ProtocolEndpoint

ProtocolEndpoint shall be implemented when an ActiveConnection or NetworkPipe exists. It may be implemented if no ActiveConnection or NetworkPipe exists.
Class Mandatory: true

Table 733: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
NameFormat		string	
ProtocolIFType		uint16	Fibrechannel, Fcip

8.2.6.9.9.23 CIM_RemotePort

Class Mandatory: true

Table 734: SMI Referenced Properties/Methods for CIM_RemotePort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemName		string	
CreationClassName		string	
Name		string	Opaque
AccessInfo		string	
InfoFormat		uint16	

Table 734: SMI Referenced Properties/Methods for CIM_RemotePort

Property	Flags	Type	Description & Notes
OtherInfoFormatDescription		string	WWN
PortProtocol		uint16	
OtherProtocolDescription		string	
Optional Properties/Methods			
PortInfo		string	WWN or TCP port number

8.2.6.9.9.24 CIM_RemoteServiceAccessPoint

Class Mandatory: true

Table 735: SMI Referenced Properties/Methods for CIM_RemoteServiceAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
AccessInfo		string	
InfoFormat		uint16	IPv4 Address OR IPv6 Address
Optional Properties/Methods			
OtherInfoFormatDescription		string	

8.2.6.9.9.25 CIM_SystemDevice

Associated FCPort and EthernetPort to the ComputerSystem
Class Mandatory: true

Table 736: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.6.9.9.26 CIM_TCPEndpointStatistics

Opaque
Class Mandatory: true

Table 737: SMI Referenced Properties/Methods for CIM_TCPEndpointStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
StatisticTime		datetime	
ReceivedSegmentsInError		uint32	
SentResetSegments		uint32	

Table 737: SMI Referenced Properties/Methods for CIM_TCPEndpointStatistics

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
ElementName		string	User-friendly name. In addition, it can be used as a index property for a search or query.
ResetSelectedStats()			

8.2.6.9.9.27 CIM_TCPProtocolEndpoint

Class Mandatory: true

Table 738: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
ProtocolIFType		uint16	
Optional Properties/Methods			
NameFormat		string	
PortNumber		uint32	

8.2.6.9.9.28 CIM_TCPSettings

Defines TCP settings for a group of TCPProtocolEndpoints which belongs to the ComputerSystem
Class Mandatory: true

Table 739: SMI Referenced Properties/Methods for CIM_TCPSettings

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
RetransmissionTimeoutAlgorithm		uint16	
RetransmissionTimeoutMin		uint16	
RetransmissionTimeoutMax		uint16	
Optional Properties/Methods			
ElementName		string	User-friendly name. In addition, it can be used as a index property for a search or query.

8.2.6.9.9.29 CIM_TCPStatisticalData

Class Mandatory: true

Table 740: SMI Referenced Properties/Methods for CIM_TCPStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
StatisticTime		datetime	
ActiveOpenConnections		uint32	
PassiveOpenConnections		uint32	
AttemptsFails		uint32	
EstablishedResets		uint32	
EstablishedConnections		uint32	
ReceivedSegments		uint32	
SentSegments		uint32	
RetransmittedSegments		uint32	
ReceivedSegmentsInError		uint32	
SentResetSegments		uint32	
ResetSelectedStats()			
Optional Properties/Methods			
ElementName		string	User-friendly Name

8.2.6.9.10 Related Standards

Table 741: Related Standards for Extender

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.7 Host Profiles

8.2.7.1 FC HBA Profile

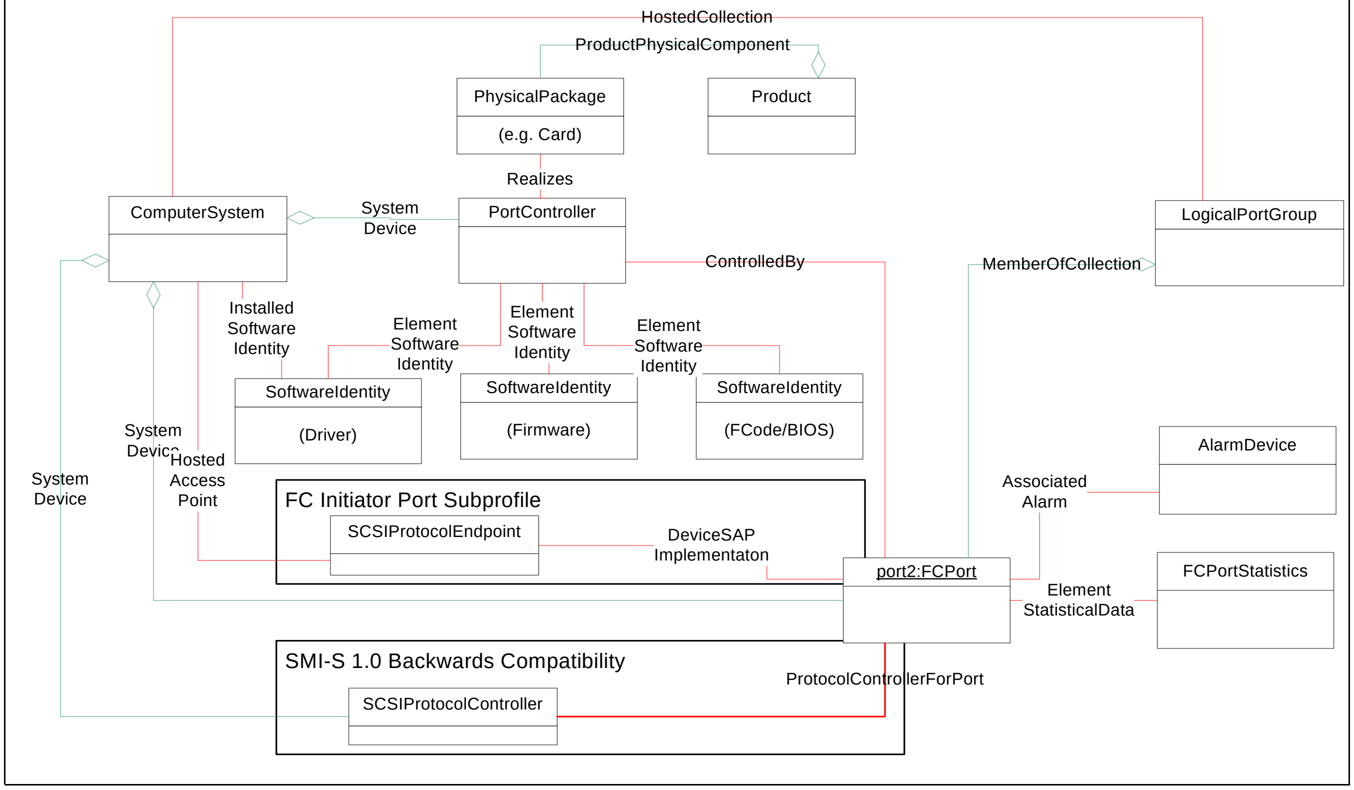
8.2.7.1.1 Description

A Fibre Channel adapter used in a host system is called a Host Bus Adapter (HBA). An HBA is a physical device that contains one or more Fibre Channel ports. A single system contains one or more HBAs.

An HBA is represented in CIM by FCPorts associated to a ComputerSystem through the SystemDevice association. To understand the containment to the HBAs physical implementation the FCPorts are associated to PhysicalPackage through the Realizes association. The PortController represents the logical behavior of the HBA card, It is associated to the ComputerSystem through the SystemDevice association and associated to the ports through the ControlledBy association. PortController's PhysicalPackage is associated with Product - which holds information about the HBA (including vendor and model names).

If the FCPorts reside on a motherboard (rather than a separate card), the same model is used - PortController and PhysicalPackage represent the motherboard. Attributes of Product refer the vendor and model names of the FCPorts, not the motherboard or system.

Figure 101: FC HBA Instance Diagram



Separate instances of SoftwareIdentity represent driver, firmware, and FCODE/BIOS associated with the HBA and includes properties for the vendor, product, and version names (see Table 767: "SMI Referenced Properties/Methods for CIM_SoftwareIdentity" for details). The Classifications property identifies the type (driver, firmware,...). The SoftwareIdentity instance for the driver is mandatory; the others are optional. Note that a separate instance of SoftwareIdentity representing the SMI-S/CIM instrumentation is required by the Server Profile.

DEPRECATED

Modeling SCSI Protocol Support

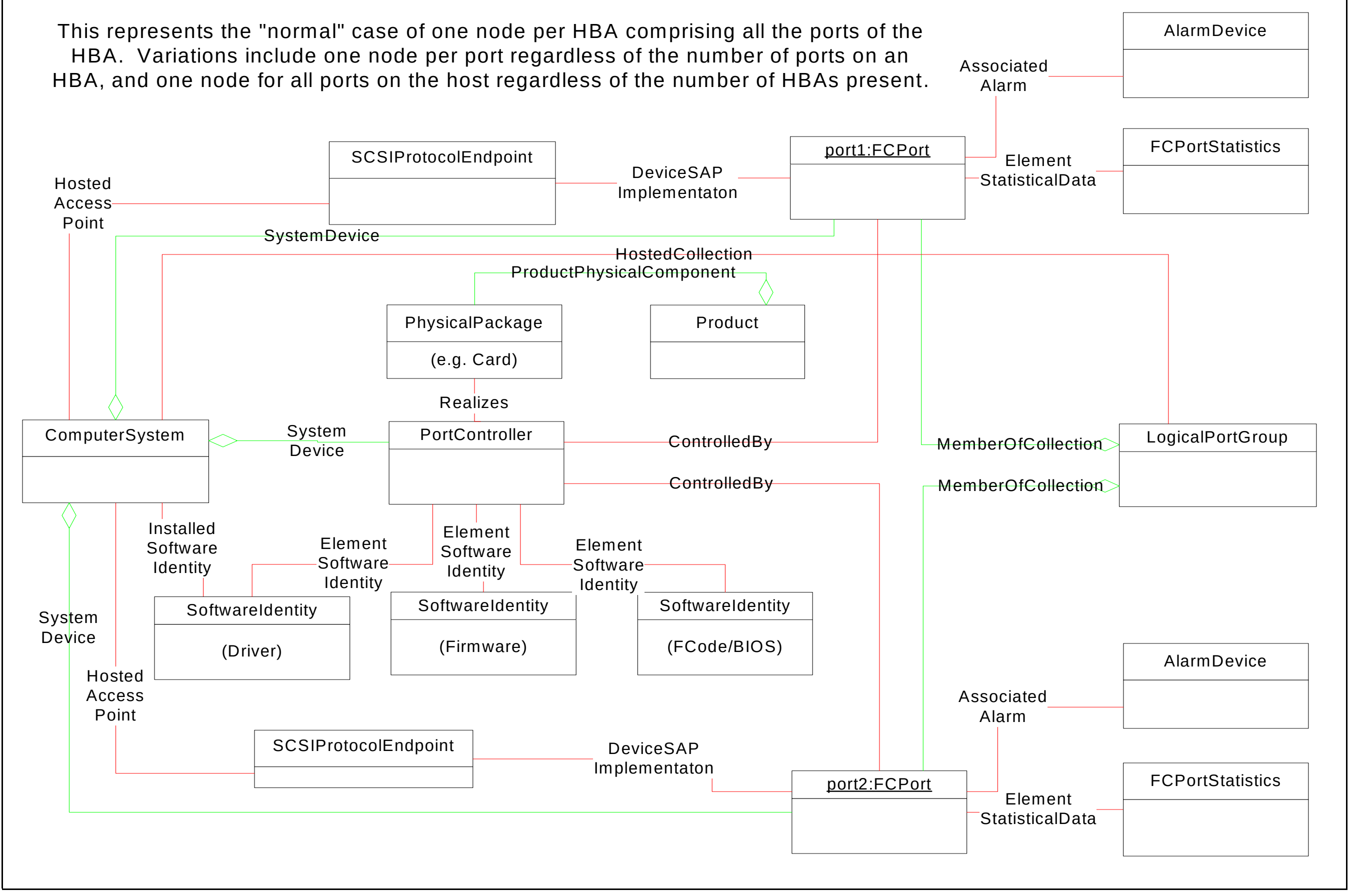
The SMI-S 1.0 model (as specified in IS24775-2006, *Storage Management*) for ports and protocols addressed FCP (SCSI over Fibre Channel). As other configurations were considered, the general pattern of initiator port subprofiles (see 8.2.3, "Common Initiator Port Subprofiles Overview") emerged. For this version of SMI-S, any initiator port that is configured for SCSI protocol shall use the model in the instance diagram above (ComputerSystem-Hosted Access Point-SCSIProtocoEndpoint-DeviceSAPImplementation-FCPort).

For backwards compatibility, the FC HBA Profile also exposes the IS24775-2006, *Storage Management* classes (SCSIProtocolController and ProtocolControllerForPort association). In the future, SCSIProtocolController and ProtocolControllerForPort will not be part of the profile; client applications are encouraged to migrate to the new model.

DEPRECATED

Figure 102: "HBA Card with Two Ports" depicts the model for an HBA card with two ports. The LogicalPortGroup represents the collection of ports that shared a Node WWN (in this case, both ports on a card, but other implementations are in use).

Figure 102: HBA Card with Two Ports



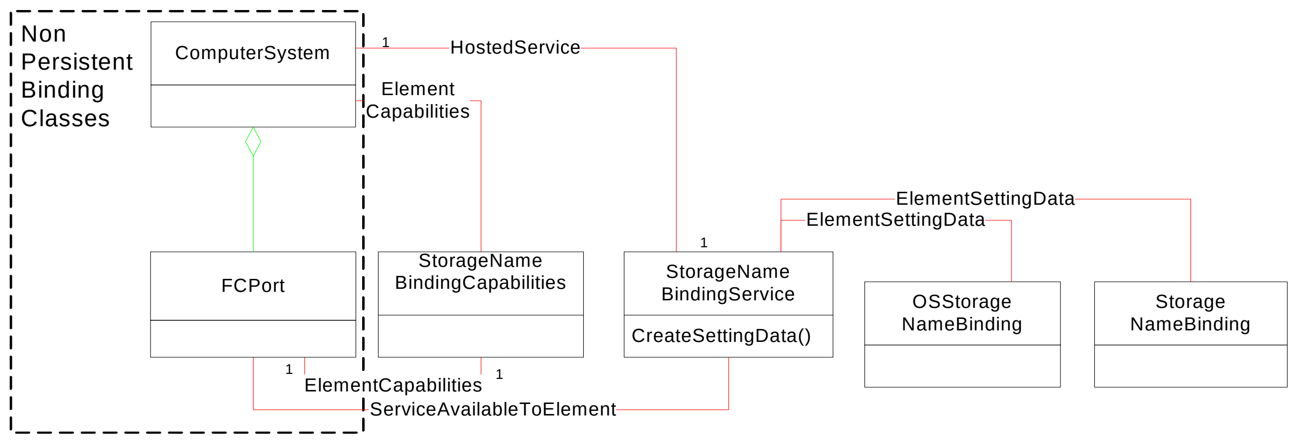
Persistent Binding

Persistent Binding describes the capability of host adapters to persist user preferences regarding which target logical units are mapped to which OS device names. Persistent Binding for Fibre Channel HBAs is documented in detail in the FC API specification.

The term “Persistent Binding” technically refers to the data structure that maps the association from target device correlatable IDs to an OS device name. The collection of these bindings is persisted by the HBA and/or drivers. A persistent binding structure can be defined while the referenced hardware is offline or uninstalled. When the drivers discover attached hardware that matches a persistent binding, the mapping takes place. In many cases, a newly defined persistent binding has no impact until the system is rebooted. The impact will cause target logical units to be attached to initiator SCSIProtocolEndpoints. These associations and target objects are modeled with the Host Discovered Resources Profile.

The persistent binding data structure for bindings that specify OS device names is modeled as OSStorageNametBinding. A persistent binding lets the OS determine the device name is modeled as StorageNameBinding. StorageNameBindingService includes methods to create instances of the setting data subclasses, and StorageNameBindingCapabilities provides information about the capabilities of the implementation

Figure 103: Persistent Binding Model



Persistent Binding is optional. An implementation that does not support persistent binding (and any of the classes in the diagram above) shall not instantiate an instance of StorageNameBindingService. An implementation that does support persistent binding shall:

- Instantiate a single instance of StorageNameBindingService and associate it to the ComputerSystem
- Instantiate an instance of StorageNameBindingCapabilities for each FCPort instance, associated via ElementCapabilities
- At initialization, the implementation shall instantiate instances of OSStorageNameBinding or StorageNameBinding for each previously defined binding.
- implement the CreateOSStorageNameBindingMethod if any StorageNameBindingCapabilities exists with CanSetOSDeviceName set to true
- implement the CreateStorageNameBindingMethod if StorageNameBindingCapabilities instance exists with CanSetOSDeviceName set to false
- support DeleteInstance for StorageNameBinding and OSStorageNameBinding

- support ModifyInstance of StorageNameBindingCapabilities. Not all properties are modifiable, see the “M” flags in Table 771: "SMI Referenced Properties/Methods for CIM_StorageNameBindingCapabilities".

<u>LED Blink</u>

Implementations may optionally support LED blinking by instantiating a AlarmDevice instance and associating it via AssociatedAlarm to Port instances.

AlarmDevice.VisibleAlarm sound be set to true.

AlarmDevice.Urgency should be set to 3 (Informational).

The instrumentation shall provide the SetAlarmState method on AlarmDevice. This method has a single parameter RequestedAlarmState. The only value for this parameter shall be 3 (Alternating).

8.2.7.1.2 Health and Fault Management

8.2.7.1.3 Supported Subprofiles and Packages

The FC HBA profile requires the FC Initiator Port Subprofile.

8.2.7.1.4 Methods of this Profile

The following extrinsic methods are available, but only required if the specific capability (persistent binding or LED blink) is supported.

8.2.7.1.4.1 StorageNameBindingService.CreateStorageNameBinding

This method requests that the driver create a name binding from a target (and optional logical unit) and lets the OS assign the name.

```
uint32 CreateStorageNameBinding (
   [IN, Description (“The value to assign to BindingType.”),
  uint16 BindingType,

    [IN, Description ("The value to assign to BindAllLogicalUnits.")]
  boolean BindAllLogicalUnits,

    [IN, Description ("The value to assign to Hide.")]
  boolean Hide,

    [IN, Description ("The value to assign to TargetName.")]
  string TargetName,

    [IN, Description ("The value to assign to LogicalUnitNumber.")]
  string LogicalUnitNumber,

    [IN, Description ("The type of the ports in LocalPortNames."),
    // shall be "2" "FC Port WWN"
  uint16 LocalPortNameType,

    [IN, Description ("The values to assign to LocalPortNames.")]
  string LocalPortName,
     [IN (false), OUT, Description ("A reference to the created name binding
instance.")]
 StorageNameBinding REF Binding);
```

8.2.7.1.4.2 StorageNameBindingService.CreateOSStorageNameBinding

This method requests that the driver create a name binding from a target (and option logical unit) to a specified OS Device Name or addresses.".

```
 uint32 CreateOSStorageNameBinding (
     [IN, Description ("The value to assign to BindingType."),
   uint16 BindingType,

     [IN, Description ("The value to assign to BindAllLogicalUnits.")]
   boolean BindAllLogicalUnits,

     [IN, Description ("The value to assign to Hide.")]
   boolean Hide,

     [IN, Description ("The value to assign to TargetName.")]
   string TargetName,

     [IN, Description ("The value to assign to LogicalUnitNumber.")]
   string LogicalUnitNumber,

     [IN, Description (“The value to assign to OSDeviceName.")]
   string OSDeviceName,

    [IN, Description ("The value to assign to OSAddressesValid.")]
   boolean OSAddressesValid,

     [IN, Description ("The value to assign to OSBusNumber.")]
   uint32 OSBusNumber,

     [IN, Description (“The value to assign to OSTargetNumber.")]
   uint32 OSTargetNumber,

     [IN, Description ("The value to assign to OSLUN.")]
   uint32 OSLUN,

     [IN, Description ("The type of the ports in LocalPortNames."),
     // shall be "2" "FC Port WWN"
   uint16 LocalPortNameType,

     [IN, Description ("The values to assign to LocalPortNames.")]
   string LocalPortName,

     [IN (false), OUT, Description ("A reference to the created name binding
instance.")]
   CIM_StorageNameBinding REF Binding);
```

CIM_AlarmDevice.SetAlarmState

8.2.7.1.5 Client Considerations and Recipes

Different HBA vendors may have separate implementations of this profile installed on the same server; the instrumentation may be running under the same or different CIM servers.

8.2.7.1.5.1 Discovery HBA Topology and Attributes

```
// DESCRIPTION
//
// This recipe discovers the topology of an FC HBA. Noteworthy information
// such as installed firmware/software and port information is retrieved.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. A reference to the top-level ComputerSystem in the FC HBA Profile,
//    which represents the system hosting the HBA, is known as $Host->
//

// Step 1. Get name(s) of the HBA’s on the host system. Note that there
// MAY be more than one HBA on the host.
//
$HBA->[] = AssociatorNames($Host->,// ObjectName
“CIM_SystemDevice”,// AssocClass
“CIM_PortController”,// ResultClass
“GroupComponent”,// Role
“PartComponent”)// ResultRole

if ($HBA->[] == null || $HBA->[].length == 0) {
<EXIT: No HBAs on the host system!>
}

// Determine the topology and retrieve noteworthy information for each HBA.
//
for (#i in $HBA->[]) {
// Step 2. Determine the vendor and product information of the HBA.
//
$PhysicalPackage[] = Associators(
     $HBA->[#i],// ObjectName
     “CIM_Realizes”,// AssocClass
     “CIM_PhysicalPackage”,// ResultClass
     “Antecedent”,// ResultRole
     “Dependent”,// Role
     false,       // IncludeQualifiers
     false,       // IncludeClassOrigin
     {“Manufacturer”, “Model”})// PropertyList
// Exactly one PhysicalPackage MUST be returned
if ($PhysicalPackage[] == null || $PhysicalPackage[].length == 0) {
     <ERROR! Improper Physical Package information!>
}
// NOTE: The Product properties of interest are all Key qualified
// properties, thus the instance name rather the instance itself
// is retrieved.
//
$Product->[] = AssociatorNames(
     $PhysicalPackage[0],// ObjectName
     “CIM_ProductPhysicalComponent”,// AssocClass
     “CIM_Product”,// ResultClass
     “GroupComponent”,// ResultRole
     “PartComponent”)// Role
// Exactly one PhysicalPackage MUST be returned
if ($Product->[] == null || $Product->[].length == 0) {
     <ERROR! Improper Product information!>
}
// Step 3. Determine the software (e.g. firmware, driver(s), BIOS,
// FCode) installed on the HBA.
//
#PropList = {“VersionString”, “Manufacturer”, “Classifications”}
$Software[] = Associators(
     $HBA->[#i],// ObjectName
     “CIM_ElementSoftwareIdentity”,// AssocClass
     “CIM_SoftwareIdentity”,// ResultClass
     “Antecedent”,// ResultRole
     “Dependent”,// Role
     false,       // IncludeQualifiers
     false,       // IncludeClassOrigin
     #PropList)// PropertyList
if ($Software[] != null && $Software[].length > 0) {
     for (#j in $Software[]) {
         // Retrieve relevant property instance data
                        // These properties are not used in the recipe,
                        // this just demostrates how to locate this
                        // information
         #VersionString = $Software[#j].VersionString
         #Manufacturer = $Software[#j].Manufacturer
         #Classifications[] = $Software[#j].Classifications
     }
}
// Step 4. Locate the Fibre Channel ports on the HBA and determine
// each port’s speed and WWN.
#PropList = {“Speed”, “PermanentAddress”}
$Ports[] = Associators(
     $HBA->[#i],// ObjectName
     “CIM_ControlledBy”,// AssocClass
     “CIM_FCPort”,// ResultClass
     “Dependent”,// ResultRole
     “Antecedent”,// Role
     false,    // IncludeQualifiers
     false,    // IncludeClassOrigin
     #PropList)// PropertyList
if ($Ports[] != null && $Ports[].length > 0) {
     for (#j in $Ports[]) {
        // Retrieve relevant Port instance data
        #Speed = $Ports[#j].Speed
        #PermanentAddress[] = $Ports[#j].PermanentAddress
        // Step 5. Determine the Node WWN of the port.
        $PortGroup[] = Associators(
           $Ports[#j].getObjectPath(),// ObjectName
           “CIM_MemberOfCollection”,// AssocClass
           “CIM_LogicalPortGroup”,// ResultClass
           “Collection”,// ResultRole
           “Member”, // Role
           false,       // IncludeQualifiers
           false,       // IncludeClassOrigin
           {“Name”}) // PropertyList
        // Exactly one PhysicalPackage MUST be returned
        if ($PortGroup[] == null || $PortGroup[].length == 0) {
           <ERROR! Improper Port Group information!>
        }
        #NodeWWN = $PortGroup[0].Name
     }
}
}
```

8.2.7.1.5.2 Get the statistics for each port

```
//
// DESCRIPTION
//
// Find the FCPortStatistics associated with FC ports
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. A reference to the top-level ComputerSystem in the FC HBA Profile,
//    which represents the system hosting the HBA, is known as $Host->
//
// Get a list of all the ports
$Ports->[] = AssociatorNames($Host->,// ObjectName
     “CIM_SystemDevice”,// AssocClass
     “CIM_FCPort”,// ResultClass
     “GroupComponent”,   // Role
     “PartComponent”)    // ResultRole

if ($Ports->[] == null || $Ports->[].length == 0) {
     <ERROR! No FC Ports on the host system!>
}

for (#i in $Ports->[] ) {
    // Get a list of FCPortStatistics associated with each port
    // Should only be exactly one FCPortStatistics instance
    $Stats->[] = AssociatorNames($Ports->[#i],// ObjectName
     “CIM_ElementStatisticalData”,// AssocClass
     “CIM_FCPortStatistics”,// ResultClass
     “ManagedElement”,// Role
     “Stats”)     // ResultRole
    if ($Stats->[] == null || $Ports->[].length == 0) {
     <ERROR! Each FCPort shall have an associated FCPortStatistics>
    } else {
        if ( $Stats->[].length > 1) {
         <ERROR: More than 1 FCPortStatistics associated with a port>
     }
    }
    // $Stats[0]-> holds that stats
}
```

8.2.7.1.5.3 Define a persistent binding to a target PWWN

```
// DESCRIPTION
//
// This recipe creates a persistent binding based on a PWWN
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. A reference to the top-level ComputerSystem in the FC HBA Profile,
//    which represents the system hosting the HBA, is known as $Host->
//
// 2. The name of the target port WWN is known as #TargetWWN.  The
//    easiest way to discover this is to use an FC Switch or Fabric
//    client application.  The host should have a single HBA Profile
//    implmentation running and the target MUST be connected to an
//    HBA supported by this profile implementation.
//
// 3. A reference to an FCPort on the local system - $LocalPort->
//

// Get a list of initiator ports
// First get all the initiator ports
$Ports->[] = AssociatorNames($Host->,// ObjectName
     “CIM_SystemDevice”,// AssocClass
     “CIM_FCPort”,// ResultClass
     “GroupComponent”,   // Role
     “PartComponent”)    // ResultRole
```

```
if ($Ports->[] == null || $Ports->[].length == 0) {
     <ERROR! No FC Ports on the host system!>
}
if (!contains($LocalPort->, $Ports->[]) {
     <ERROR! The input local port is not on the host system!>
}

$Services->[] = AssociatorNames($Host->,// ObjectName
     “CIM_HostedService”,// AssocClass
     “CIM_StorageNameBindingService”,// ResultClass
     null,null)

if ($Services == null || $Servicse[].length == 0) {
    <ERROR: HBA Instrumentation does not instantiate StorageNameBindingService>
}
if ($Services[].length != 1) {
    <ERROR! Must be exactly one StorageNameBindingService>
}

$Capabilities->[] = AssociatorNames($FCPort->,// ObjectName
     “CIM_ElementCapabilities”,// AssocClass
     “CIM_StorageNameBindingCapabilities”,// ResultClass
        null, null)

If ($Capabilities == null || $Capabilities[].length != 1) {
    <ERROR! must be exactly one StorageNameBindingCapabilities per FCPort>
}

If ($Capabilities->[0].CanBindAllLuns != true) {
    <EXIT: HBA Instrumentation does not support CanBindAllLuns>
}

If contains(“FcApiBindToWWN”, $Capabilities->[0].ValidBindingTypes) {
    // All checks done, perform the binding
    // set up the arguments and invoke CreateStorageNameBinding
    %InArguments[“BindingType”] = “FcApiBindToWWPN”
    %InArguments[“BindAllLogicalUnits”]=true
    %InArguments[“Hide”]=false
    %InArguments[“TargetName”]=#TargetPWWN
    %InArguments[“LocalPortNameType”]=”2”// FC Port WWN
    %InArguments[“LocalPortName”]=$LocalPort->[].PermanentAddress
    #MethodReturn = InvokeMethod(
        $Services->[0],
        “CreateStorageNameBinding”,
        %InArguments,
        %OutArguments)
    if(#MethodReturn != 0) {
     <ERROR! CreateStorageNameBinding method Failed >
    }

    If ($Capabilities->[0].ActivateBindingRequiresReset) {
        <EXIT: Persistent Binding request okay; Reboot Required>
    }

} else {
    <EXIT: HBA instrumentation does not support BindtoWWPN>
}
```

8.2.7.1.5.4 Define a persistent binding to an LUID

```
// DESCRIPTION
//
// This recipe creates a persistent binding based on a LUID
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
//
// 1. A reference to the top-level ComputerSystem in the FC HBA Profile,
//    which represents the system hosting the HBA, is known as $Host->
//
// 2. The name of the logical unit (VPD pg 83 ID) is known as #LUID.
//    The easiest way to discover this is to use an array management
//    client application.  The host should have a single HBA Profile
//    implmentation running and the LU shall be in a target connected
//    to an HBA supported by this profile implementation.
//
// 3. A reference to an FCPort on the local system - $LocalPort->
//

// Get a list of initiator ports
// First get all the initiator ports
$Ports->[] = AssociatorNames($Host->,// ObjectName
     “CIM_SystemDevice”,// AssocClass
     “CIM_FCPort”,// ResultClass
     “GroupComponent”,   // Role
     “PartComponent”)    // ResultRole

if ($Ports->[] == null || $Ports->[].length == 0) {
     <ERROR! No FC Ports on the host system!>
}
if (!contains($LocalPort->, $Ports->[]) {
     <ERROR: The input local port is not on the host system!>
}

$Services->[] = AssociatorNames($Host->,// ObjectName
     “CIM_HostedService”,// AssocClass
     “CIM_StorageNameBindingService”,// ResultClass
     null,null)

if ($Services == null || $Servicse[].length == 0) {
    <ERROR: HBA Instrumentation does not instantiate StorageNameBindingService>
}
if ($Services[].length != 1) {
    <ERROR! Must be exactly one StorageNameBindingService>
}

$Capabilities->[] = AssociatorNames($FCPort->,// ObjectName
     “CIM_ElementCapabilities”,// AssocClass
     “CIM_StorageNameBindingCapabilities”,// ResultClass
        null, null)

If ($Capabilities == null || $Capabilities[].length != 1) {
    <ERROR! must be exactly one StorageNameBindingCapabilities per FCPort>
}

If contains(“BindToLUID”, $Capabilities->[0].ValidBindingTypes) {
    // All checks done, perform the binding
    // set up the arguments and invoke CreateStorageNameBinding
    %InArguments[“BindingType”]=”BindToLUID”
    %InArguments[“BindAllLogicalUnits”]=true
    %InArguments[“Hide”]=false
    %InArguments[“TargetName”]=#LUID
    %InArguments[“LocalPortNameType”]=”2”// FC Port WWN
    %InArguments[“LocalPortName”]=$LocalPort->[].PermanentAddress
    #MethodReturn = InvokeMethod(
        $Services->[0],
        “CreateStorageNameBinding”,
        %InArguments,
        %OutArguments)
    if(#MethodReturn != 0) {
        <ERROR! CreateStorageNameBinding method Failed>
    }

    If ($Capabilities->[0].ActivateBindingRequiresReset) {
        <EXIT: Persistent Binding request okay; Reboot Required>
    }

} else {
    <EXIT: HBA instrumentation does not support BindtoLUID>
}
```

8.2.7.1.5.5 Blink the LED

```
//
```

```
// DESCRIPTION
//
// Blink LEDs associated with FC ports
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. A reference to the top-level ComputerSystem in the FC HBA Profile,
//    which represents the system hosting the HBA, is known as $Host->
//
//    The host should have a single HBA Profile implmentation running
//
// Get a list of all the ports
$Ports->[] = AssociatorNames($Host->,// ObjectName
     “CIM_SystemDevice”,// AssocClass
     “CIM_FCPort”,// ResultClass
     “GroupComponent”,   // Role
     “PartComponent”)    // ResultRole

if ($Ports->[] == null || $Ports->[].length == 0) {
     <ERROR! No FC Ports on the host system!>
}

for (#i in $Ports->[] ) {
    // Get a list of Alarms associated with each port
    // Should only be one (or zero) alarms
    $Alarms->[] = AssociatorNames($Ports->[#i],// ObjectName
     “CIM_AssociatedAlarm”,// AssocClass
     “CIM_Alarm”,// ResultClass
     “Antecedent”,   // Role
     “Dependent”)    // ResultRole
    if ($Alarms->[] == null || $Ports->[].length == 0) {
     <EXIT: HBA Instrumentation does not support LED blink>
    } else {
        if ( $Alarms->[].length > 1) {
         <ERROR! More than 1 alarm associated with a port>
     }
    }

    // invoke the method to blink the alarm
    %InArguments[“RequestedAlarmState”] = “Alternating”
    #MethodReturn = InvokeMethod(
     $Alarms->[0],
     “SetAlarmState”,
     %InArguments)

    if(#MethodReturn != 0)
    {
```

```
                <ERROR! SetAlarmState (blink LED) method Failed >
            }
        }
```

8.2.7.1.6 Registered Name and Version

FC HBA version 1.1.0

8.2.7.1.7 CIM Server Requirements

Table 742: CIM Server Requirements for FC HBA

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.7.1.8 CIM Elements

Table 743: CIM Elements for FC HBA

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.7.1.8.3)	
CIM_ControlledBy (8.2.7.1.8.4)	
CIM_ElementSoftwareIdentity (8.2.7.1.8.7)	
CIM_ElementStatisticalData (8.2.7.1.8.8)	
CIM_FCPort (8.2.7.1.8.9)	
CIM_FCPortStatistics (8.2.7.1.8.10)	
CIM_PhysicalPackage (8.2.7.1.8.17)	
CIM_PortController (8.2.7.1.8.18)	
CIM_Product (8.2.7.1.8.19)	
CIM_ProductPhysicalComponent (8.2.7.1.8.20)	
CIM_Realizes (8.2.7.1.8.22)	
CIM_SoftwareIdentity (8.2.7.1.8.24)	Driver
CIM_SystemDevice (8.2.7.1.8.30)	
Optional Classes	
CIM_AlarmDevice (8.2.7.1.8.1)	optional
CIM_AssociatedAlarm (8.2.7.1.8.2)	optional
CIM_ElementCapabilities (8.2.7.1.8.5)	
CIM_ElementSettingData (8.2.7.1.8.6)	
CIM_HostedCollection (8.2.7.1.8.11)	Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System.
CIM_HostedService (8.2.7.1.8.12)	
CIM_InstalledSoftwareIdentity (8.2.7.1.8.13)	
CIM_LogicalPortGroup (8.2.7.1.8.14)	Collection of Fibre Channel ports that share a Node WWN
CIM_MemberOfCollection (8.2.7.1.8.15)	Associates FCPort to the LogicalPortGroup
CIM_OSStorageNameBinding (8.2.7.1.8.16)	
CIM_ProtocolControllerForPort (8.2.7.1.8.21)	
CIM_ServiceAvailableToElement (8.2.7.1.8.23)	
CIM_SoftwareIdentity (8.2.7.1.8.25)	Firmware
CIM_SoftwareIdentity (8.2.7.1.8.26)	FCODE/BIOS
CIM_StorageNameBinding (8.2.7.1.8.27)	
CIM_StorageNameBindingCapabilities (8.2.7.1.8.28)	
CIM_StorageNameBindingService (8.2.7.1.8.29)	
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_PortController	PortController (HBA) Creation. See 8.2.7.1.5.1
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_PorController	PortController (HBA) Removal

8.2.7.1.8.1 CIM_AlarmDevice

optional
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 744: SMI Referenced Properties/Methods for CIM_AlarmDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
VisibleAlarm		boolean	shall be "true"
Urgency		uint16	shall be 3 (Alternating)
SetAlarmState()			

8.2.7.1.8.2 CIM_AssociatedAlarm

optional
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 745: SMI Referenced Properties/Methods for CIM_AssociatedAlarm

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_AlarmDevice	Reference to AlarmDevice
Dependent		CIM_LogicalDevice	Reference to FCPort

8.2.7.1.8.3 CIM_ComputerSystem

Created By : Static
Modified By : Static
Deleted By : Static
Standard Names: The Name and NameFormat properties shall follow the requirements in 6.2.4.5.4
Class Mandatory: true

Table 746: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	The name of the host containing the HBA.
ElementName		string	
NameFormat		string	
OtherIdentifyingInfo	C	string[]	

Table 746: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Dedicated		uint16[]	0 (Not Dedicated)
OperationalStatus		uint16[]	
Optional Properties/Methods			
OtherDedicatedDescriptions		string[]	

8.2.7.1.8.4 CIM_ControlledBy

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 747: SMI Referenced Properties/Methods for CIM_ControlledBy

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Controller	Reference to PortController
Dependent		CIM_LogicalDevice	Reference to FCPort

8.2.7.1.8.5 CIM_ElementCapabilities

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 748: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Capabilities		CIM_Capabilities	Reference to StorageNameBindingCapabilities
ManagedElement		CIM_ManagedElement	Reference to FCPort

8.2.7.1.8.6 CIM_ElementSettingData

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 749: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	Reference to StorageNameBindingService
SettingData		CIM_SettingData	Reference to StorageNameBinding or OSStorageNameBinding

8.2.7.1.8.7 CIM_ElementSoftwareIdentity

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 750: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SoftwareIdentity	Reference to SoftwareIdentity
Dependent		CIM_ManagedElement	Reference to the PortController

8.2.7.1.8.8 CIM_ElementStatisticalData

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 751: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Stats		CIM_StatisticalData	

8.2.7.1.8.9 CIM_FCPort

Class Mandatory: true

Table 752: SMI Referenced Properties/Methods for CIM_FCPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PermanentAddress		string	Override PermanentAddress to be mandatory in this profile.

8.2.7.1.8.10 CIM_FCPortStatistics

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 753: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	
InstanceID		string	

Table 753: SMI Referenced Properties/Methods for CIM_FCPortStatistics

Property	Flags	Type	Description & Notes
BytesTransmitted		uint64	From NetworkPortStatistics Superclass. Maps to HBA_PortStatistics,TxWords. Multiply word count by 4
BytesReceived		uint64	From NetworkPortStatistics Superclass. Maps to HBA_PortStatistics.RxWords. Multiply word count by 4
PacketsTransmitted		uint64	From NetworkPortStatistics Superclass. Maps to HBA_PortStatistics.TxFrames
PacketsReceived		uint64	From NetworkPortStatistics Superclass. Maps to HBA_PortStatistics.RxFrames
CRCErrors		uint64	Maps to HBA_PortStatistics.InvalidCRCCount
LinkFailures		uint64	Maps to HBA_PortStatistics.LinkFailureCount
PrimitiveSeqProtocolErrCount		uint64	
LossOfSignalCounter		uint64	Maps to HBA_PortStatistics.LossOfSignalCount
InvalidTransmissionWords		uint64	Maps to HBA_PortStatistics.InvalidCRCCount
LIPCount		uint64	
NOSCount		uint64	
ErrorFrames		uint64	
DumpedFrames		uint64	
LossOfSyncCounter		uint64	Maps to HBA_PortStatistics.LossOfSynchCount
Optional Properties/Methods			
StatisticTime		datetime	optional - time last measurement was taken

8.2.7.1.8.11 CIM_HostedCollection

Associates the LogicalPortGroup (Fibre Channel Node) to the hosting System. The hosting System is either a ComputerSystem for the Platform or the AdminDomain for those systems not registered in the Platform Database or discovered through FDMI.

Class Mandatory: false

Table 754: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The reference to the System

Table 754: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Dependent		CIM_SystemSpecificCollection	The reference to the LogicalPortGroup (Fibre Channel Node)

8.2.7.1.8.12 CIM_HostedService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 755: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	Reference to ComputerSystem
Dependent		CIM_Service	Reference to StorageNameBindingService

8.2.7.1.8.13 CIM_InstalledSoftwareIdentity

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 756: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.7.1.8.14 CIM_LogicalPortGroup

Represents the Fibre Channel Node. Associated to the host system by the HostedCollection Association.

Class Mandatory: false

Table 757: SMI Referenced Properties/Methods for CIM_LogicalPortGroup

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque
Name	D	string	Fibre Channel Node WWN
NameFormat		string	"WWN"
ElementName	N	string	Node Symbolic Name if available. Otherwise NULL. If the underlying implementation includes characters that are illegal in CIM strings, then truncate before the first of those characters.

8.2.7.1.8.15 CIM_MemberOfCollection

Associates FCPort to the LogicalPortGroup

Class Mandatory: false

Table 758: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The reference to the LogicalPortGroup representing the Fibre Channel Node
Member		CIM_ManagedElement	The reference to FCPort.

8.2.7.1.8.16 CIM_OSStorageNameBinding

The structure representing an FC persistent binding when the caller specifies the OS Device name. Description column includes mapping to FC API properties.

Created By : StaticExtrinsic(s): CIM_StorageNameBindingService.CreateOSStorageNameBinding
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 759: SMI Referenced Properties/Methods for CIM_OSStorageNameBinding

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
BindingType		uint16	API HBA_BIND_TYPE. 2=FCApiBind-ToDID, 3=FCApiBindToWWPN, 4=FCApiBindToWWNN, 5=BindToLUID
BindAllLogicalUnits		boolean	API HBA_BIND_TARGETS. If true, then all target logical units are bound to the OS.Not valid to set this if Binding-Type is BindToLUID.
Hide		boolean	Must be false
TargetName	CD	String	API FCID, NodeWWN, PortWWN or HBA_LUID. If BindingType is FcApiB-indToDID, TargetName holds a hexa-decimal-encoded representation of the 32-bit D_ID and corresponds to FC API HBA_FCPID.FcId. If BindingType is FcApiBindToWWPN or FcApiBindToW-WNN, TargetName holds a hexadeci-mal-encoded representation of the 64-bit FC Port or Node World Wide Name. If BindingType is BindToLUID, Target-Name holds a SCSI Logical Unit Name from Inquiry VPD page 83, Association 0 as defined in SCSI Primary Com-mands. If the identifier descriptor (in the SCSI response) has Code Set Binary, then TargetName is its hexadecimal-encoded value.
Status		uint32	HBA_FCPBINDING2.Status

Table 759: SMI Referenced Properties/Methods for CIM_OSStorageNameBinding

Property	Flags	Type	Description & Notes
OSDeviceName		string	
OSAddressesValid		boolean	Indicates whether OSBusNumber, OSTargetNumber, and OSLUN properties are valid.
OSBusNumber		uint32	API SCSIBusNumber
OSTargetNumber		uint32	API osTargetId
OSLUN		uint32	API osLUN
LocalPortNameType		uint16	Must be 2 - FC Port WWN
LocalPortName		string	initiator port WWN

8.2.7.1.8.17 CIM_PhysicalPackage

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 760: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Manufacturer		string	
Model		string	
Tag		string	
CreationClassName		string	

8.2.7.1.8.18 CIM_PortController

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 761: SMI Referenced Properties/Methods for CIM_PortController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ControllerType		uint16	

8.2.7.1.8.19 CIM_Product

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

Class Mandatory: true

Table 762: SMI Referenced Properties/Methods for CIM_Product

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	
Name		string	
IdentifyingNumber		string	
Vendor		string	
Version		string	

8.2.7.1.8.20 CIM_ProductPhysicalComponent

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 763: SMI Referenced Properties/Methods for CIM_ProductPhysicalComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_Product	
PartComponent		CIM_PhysicalElement	

8.2.7.1.8.21 CIM_ProtocolControllerForPort

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 764: SMI Referenced Properties/Methods for CIM_ProtocolControllerForPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalPort	Reference to FCPort
Antecedent		CIM_ProtocolController	Reference to SCSIProtocolController

8.2.7.1.8.22 CIM_Realizes

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 765: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalDevice	
Antecedent		CIM_PhysicalElement	

8.2.7.1.8.23 CIM_ServiceAvailableToElement

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 766: SMI Referenced Properties/Methods for CIM_ServiceAvailableToElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ServiceProvided		CIM_Service	Reference to StorageNameBindingSer-vice
UserOfService		CIM_ManagedElement	Reference to FCPort

8.2.7.1.8.24 CIM_SoftwareIdentity

Driver

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 767: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	The name used to identify this Softwa-reIdentity.
VersionString		string	Software Version should be in the form [Major], [Minor].[Revision] or [Major].[Minor][letter][revision].
Manufacturer		string	Manufacturer of this Software.
Classifications		uint16[]	

8.2.7.1.8.25 CIM_SoftwareIdentity

Firmware

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 768: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	The name used to identify this Softwa-reIdentity.
VersionString		string	Software Version should be in the form [Major], [Minor].[Revision] or [Major].[Minor][letter][revision].
Manufacturer		string	Manufacturer of this Software.

Table 768: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Classifications		uint16[]	

8.2.7.1.8.26 CIM_SoftwareIdentity

FCODE/BIOS

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 769: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	The name used to identify this SoftwareIdentity.
VersionString		string	Software Version should be in the form [Major], [Minor].[Revision] or [Major].[Minor][letter][revision].
Manufacturer		string	Manufacturer of this Software.
Classifications		uint16[]	

8.2.7.1.8.27 CIM_StorageNameBinding

The structure representing an FC persistent binding when the driver/platform implicitly creates the OS device name. Description column includes mapping to FC API properties.

Created By : StaticExtrinsic(s): CIM_StorageNameBindingService.CreateStorageNameBinding
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 770: SMI Referenced Properties/Methods for CIM_StorageNameBinding

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
BindingType		uint16	API HBA_BIND_TYPE. 2=FCApiBindToDID, 3=FCApiBindToWWPN, 4=FCApiBindToWWNN, 5=BindToLUID
BindAllLogicalUnits		boolean	API HBA_BIND_TARGETS. If true, then all target logical units are bound to the OS.Not valid to set this if BindingType is BindToLUID.
Hide		boolean	Must be false

Table 770: SMI Referenced Properties/Methods for CIM_StorageNameBinding

Property	Flags	Type	Description & Notes
TargetName	CD	String	API FCID, NodeWWN, PortWWN or HBA_LUID. If BindingType is FcApiBindToDID, TargetName holds a hexadecimal-encoded representation of the 32-bit D_ID and corresponds to FC API HBA_FCPID.FcId. If BindingType is FcApiBindToWWPN or FcApiBindToWWNN, TargetName holds a hexadecimal-encoded representation of the 64-bit FC Port or Node World Wide Name. If BindingType is BindToLUID, TargetName holds a SCSI Logical Unit Name from Inquiry VPD page 83, Association 0 as defined in SCSI Primary Commands. If the identifier descriptor (in the SCSI response) has Code Set Binary, then TargetName is its hexadecimal-encoded value.
Status		uint32	HBA_FCPBINDING2.Status
LocalPortNameType		uint16	Must be 2 - FC Port WWN
LocalPortName	CD	string	initiator port WWN

8.2.7.1.8.28 CIM_StorageNameBindingCapabilities

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 771: SMI Referenced Properties/Methods for CIM_StorageNameBindingCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ValidBindingTypes	M	uint16[]	API HBA_BIND_TYPE. Must include a subset of 2(FcApiBindToDID), 3(FcApiBindToWWPN), 4(FcApiBindToWWNN), or 5(BindToLUID)
ActivateBindingRequiresReset		boolean	True if creating a binding requires a system reboot
CanMapAddresses	M	boolean	True if the implementation allows overriding OS bus/target/LUN numbers.
CanBindAllLuns	M	boolean	
AutoDiscovery		boolean	
CanSetOSDeviceName		boolean	

8.2.7.1.8.29 CIM_StorageNameBindingService

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

Class Mandatory: false

Table 772: SMI Referenced Properties/Methods for CIM_StorageNameBindingService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreateStorageNameBinding()			
CreateOSStorageNameBinding()			

8.2.7.1.8.30 CIM_SystemDevice

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 773: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.7.1.9 Related Standards

Table 774: Related Standards for FC HBA

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.7.2 iSCSI Initiator Profile

8.2.7.2.1 Description

An iSCSI initiator is the hardware and driver combination that acts as a client to an iSCSI target device. iSCSI initiators may utilize general –purpose Network Interface Cards (NICs) or hardware optimized for storage such as TCP Offload Engines (TOEs). iSCSI initiators may be running on a customer server or the “back end” of a bridge or virtualizer.

iSCSI terminology spans SCSI and network concepts and introduces new terms. Table 775 is a summary of some key iSCSI terms, their equivalent CIM classes, and definitions (from the IETF iSCSI RFC).

Table 775: iSCSI Terminology

iSCSI Term	CIM Class Name	Notes
Network Entity	ComputerSystem	The Network Entity represents a device or gateway that is accessible from the IP network. A Network Entity shall have one or more Network Portals, each of which can be used to gain access to the IP network by some iSCSI Nodes contained in that Network Entity.
Session	iSCSISession	The group of TCP connections that link an initiator with a target form a session (loosely equivalent to a SCSI I-T nexus). TCP connections can be added and removed from a session. Across all connections within a session, an initiator sees one and the same target.
Connection	iSCSIConnection	A connection is a TCP connection. Communication between the initiator and target occurs over one or more TCP connections. The TCP connections carry control messages, SCSI commands, parameters, and data within iSCSI Protocol Data Units (iSCSI PDUs).
SCSI Port	iSCSIProtocolEndpoint	A SCSI Port using an iSCSI service delivery subsystem. A collection of Network Portals that together act as a SCSI initiator or target.
Network Portal	TCPProtocolEndpoint, IPProtocolEndpoint, EthernetPort	The Network Portal is a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. A Network Portal in an initiator is identified by its IP address. A Network Portal in a target is identified by its IP address and its listening TCP port.
Node	SCSIProtocolController	The iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals. An iSCSI Node is identified by its iSCSI Name. The separation of the iSCSI Name from the addresses used by and for the iSCSI Node allows multiple iSCSI nodes to use the same address, and the same iSCSI node to use multiple addresses.

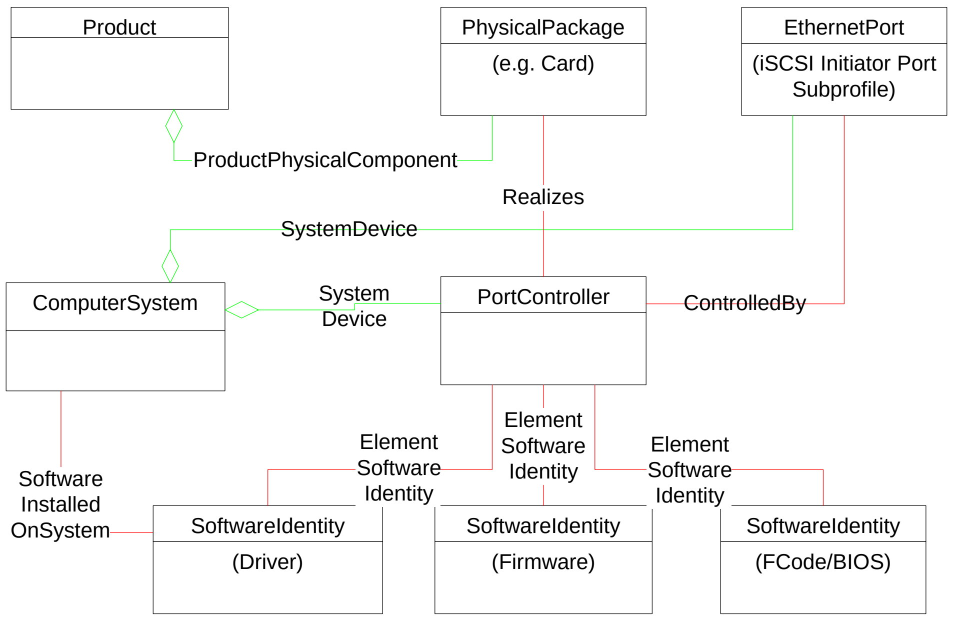
This profile requires the iSCSI Initiator Port Subprofile (see 8.2.3.3) that includes classes (EthernetPort, iSCSIProtocolEndoint) that model SCSI ports and network portals.

Figure 104: "iSCSI Product and Package Model" models the relationships between the iSCSI port classes and physical and product classes. A single iSCSI card may contain multiple Ethernet ports PhysicalPackage subclass Card models an add-in card with multiple Ethernet ports. Other PhysicalPackage subclasses may be used to model Ethernet ports embedded on a motherboard. PortController models a common management interface to multiple Ethernet ports.

ComputerSystem models the system hosting the initiator components. This is the same instance as iSCSI Network Entity in the previous diagram.

An implementation includes single instances of PhysicalPackage, Product, and PortController, plus SoftwareIdentity instances for the driver, firmware, and Fcode/BIOS. The Product instance may be shared across cards with the same make and model

Figure 104: iSCSI Product and Package Model

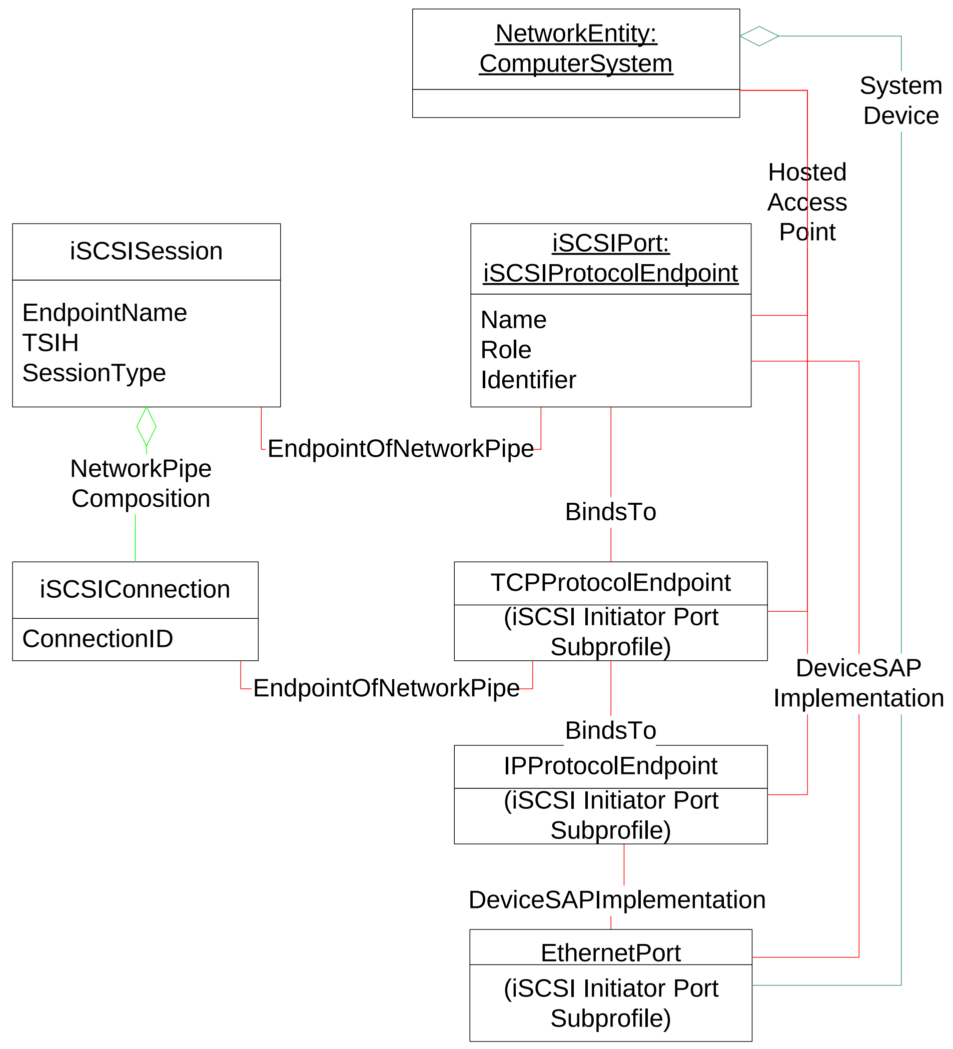


Sessions and Connections

A session is an active communication stream between an iSCSI initiator port and an iSCSI target port. However, any given session may contain part or all of the TCP/IP addresses within a Portal Group. Conceptually, a Portal Group is a pool of addresses which may be used to create/receive a session.

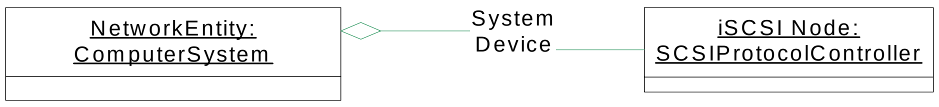
The implementation may optionally model iSCSI sessions and connections with instances of iSCSISession and iSCSIConnection classes associate to iSCSIProtocolEndpoint and TCPProtocolEndpoint (respectively) using EndpointOfNetworkPipe association.

Figure 105: iSCSI Sessions and Connections Model



There should be a single instance of SCSIProtocolController representing the initiator iSCSI node. This is associated via SystemDevice to the ComputerSystem. See Figure 106: "iSCSI Initiator Node"

Figure 106: iSCSI Initiator Node


NetworkEntity: ComputerSystem
System Device
iSCSI Node: SCSIProtocolController

8.2.7.2.2 Durable Names and Correlatable IDs of the Profile

The Name property for the iSCSI node (SCSIProtocolController) shall be a compliant iSCSI name as described in 6.2.4.9, "iSCSI Names" and NameFormat shall be set to “iSCSI Name”.

The Name property for iSCSIProtocolEndpoint shall be a compliant iSCSI name as described in 6.2.4.9, "iSCSI Names" and ConnectionType shall be set to “iSCSI”.

The Name property for EthernetPort shall be a compliant iSCSI name as described in 6.2.4.9, "iSCSI Names".

8.2.7.2.3 Health and Fault Management Considerations

The status of an Ethernet port may be determined by the value of the OperationalStatus property. Table 776, “OperationalStatus Values” defines the possible states that shall be supported for EthernetPort.OperationalStatus. The main OperationalStatus shall be the first element in the array

Table 776: OperationalStatus Values

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

8.2.7.2.4 Supported Subprofiles and Packages

Table 777: Supported Subprofiles for iSCSI Initiator

Registered Subprofile Names	Mandatory	Version
iSCSI Initiator Ports	No	1.1.0

8.2.7.2.5 Methods of the Profile

None

8.2.7.2.6 Client Considerations and Recipes

8.2.7.2.6.1 Add an additional NIC port

See 8.2.2.3.5.7, "Add a Network Portal to a Target Port." in the iSCSI Target Ports subprofile.

8.2.7.2.6.2 Find the health of an initiator

See 8.2.2.3.5.9, "Determine the health of a Session on a target system." in the iSCSI Target Ports subprofile.

8.2.7.2.6.3 Enable/disable header and data digest

See 8.2.7.2.6.3, "Enable/disable header and data digest" in the iSCSI Target Ports subprofile.

8.2.7.2.7 Registered Name and Version

iSCSI Initiator version 1.1.0

8.2.7.2.8 CIM Server Requirements

Table 778: CIM Server Requirements for iSCSI Initiator

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.7.2.9 CIM Elements

Table 779: CIM Elements for iSCSI Initiator

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.7.2.9.1)	
CIM_ElementSoftwareIdentity (8.2.7.2.9.3)	
CIM_EndpointOfNetworkPipe (8.2.7.2.9.4)	
CIM_PhysicalPackage (8.2.7.2.9.6)	
CIM_Product (8.2.7.2.9.8)	
CIM_ProductPhysicalComponent (8.2.7.2.9.9)	
CIM_ProtocolControllerForPort (8.2.7.2.9.10)	
CIM_Realizes (8.2.7.2.9.11)	
CIM_SCSIProtocolController (8.2.7.2.9.12)	
CIM_SystemDevice (8.2.7.2.9.14)	
CIM_iSCSISession (8.2.7.2.9.16)	
Optional Classes	
CIM_ControlledBy (8.2.7.2.9.2)	
CIM_InstalledSoftwareIdentity (8.2.7.2.9.5)	
CIM_PortController (8.2.7.2.9.7)	
CIM_SoftwareIdentity (8.2.7.2.9.13)	
CIM_iSCSIConnection (8.2.7.2.9.15)	
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_PortController	PortController (HBA) Creation
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_PorController	PortController (HBA) Removal

8.2.7.2.9.1 CIM_ComputerSystem

Created By : Static
Modified By : Static

Deleted By : Static
Class Mandatory: true

Table 780: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	The name of the host containing the iSCSI initiator.
ElementName		string	
NameFormat		string	
OtherIdentifyingInfo	C	string[]	
OperationalStatus		uint16[]	
Dedicated		uint16[]	"Not Dedicated"
Optional Properties/Methods			
OtherDedicatedDescriptions		string[]	

8.2.7.2.9.2 CIM_ControlledBy

Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 781: SMI Referenced Properties/Methods for CIM_ControlledBy

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Controller	
Dependent		CIM_LogicalDevice	

8.2.7.2.9.3 CIM_ElementSoftwareIdentity

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true
No specified properties or methods.

8.2.7.2.9.4 CIM_EndpointOfNetworkPipe

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 782: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	
Dependent		CIM_NetworkPipe	

8.2.7.2.9.5 CIM_InstalledSoftwareIdentity

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 783: SMI Referenced Properties/Methods for CIM_InstalledSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
System		CIM_System	
InstalledSoftware		CIM_SoftwareIdentity	

8.2.7.2.9.6 CIM_PhysicalPackage

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 784: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Manufacturer		string	Maps to IMA_PHBA_PROPERTIES.vendor
Model		string	Maps to IMA_PHBA_PROPERTIES.model

8.2.7.2.9.7 CIM_PortController

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 785: SMI Referenced Properties/Methods for CIM_PortController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ControllerType		uint16	

8.2.7.2.9.8 CIM_Product

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

Class Mandatory: true

Table 786: SMI Referenced Properties/Methods for CIM_Product

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	
Name		string	
IdentifyingNumber		string	Maps to IMA_PHBA_PROPERTIES, serialNumber
Vendor		string	Maps to IMA_PHBA_PROPERTIES, vendor
Version		string	Maps to IMA_PHBA_PROPERTIES, hardwareVersion

8.2.7.2.9.9 CIM_ProductPhysicalComponent

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 787: SMI Referenced Properties/Methods for CIM_ProductPhysicalComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_Product	
PartComponent		CIM_PhysicalElement	

8.2.7.2.9.10 CIM_ProtocolControllerForPort

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 788: SMI Referenced Properties/Methods for CIM_ProtocolControllerForPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalPort	
Antecedent		CIM_ProtocolController	

8.2.7.2.9.11 CIM_Realizes

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true
No specified properties or methods.

8.2.7.2.9.12 CIM_SCSIProtocolController

Created By : External
Modified By : External

Deleted By : External
Class Mandatory: true

Table 789: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ElementName		string	iSCSI Alias
Name	CD	string	Maps to IMA_NODE_PROPERTIES, name
NameFormat		uint16	

8.2.7.2.9.13 CIM_SoftwareIdentity

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 790: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	Maps to IMA_PHBA_PROPERTIES, driverVersion/firmwareVersion/option-RomVersion as per the Classifications property
Manufacturer		string	Maps to IMA_PHBA_PROPERTIES.vendor
Classifications		uint16[]	Either Driver','Firmware',or'BIOS/FCode'(2,10,or11)'

8.2.7.2.9.14 CIM_SystemDevice

Created By : Static
Class Mandatory: true

Table 791: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.7.2.9.15 CIM_iSCSIConnection

Created By : External
Modified By : External

Deleted By : External
Class Mandatory: false

Table 792: SMI Referenced Properties/Methods for CIM_iSCSIConnection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ConnectionID		uint32	
MaxReceiveDataSegmentLength		uint32	Maps to IMA_GetMaxRecvDataSegmentLength Properties, IMA_SetMaxRecvDataSegmentLength
MaxTransmitDataSegmentLength		uint32	
HeaderDigestMethod		uint16	
DataDigestMethod		uint16	
ReceivingMarkers		boolean	
SendingMarkers		boolean	
ActiveiSCSIVersion		boolean	
AuthenticationMethodUsed		uint16	Maps to IMA_GetInUseInitiatorAuthMethods.
MutualAuthentication		boolean	
Optional Properties/Methods			
OtherHeaderDigestMethod		string	
OtherDataDigestMethod		string	

8.2.7.2.9.16 CIM_iSCSISession

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 793: SMI Referenced Properties/Methods for CIM_iSCSISession

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
Directionality		uint16	
SessionType		uint16	
TSIH		uint32	
EndPointName		string	Maps to IMA_TARGET_PROPERTIES, name
CurrentConnections		uint32	
InitialR2T		boolean	Maps to IMA_GetInitialR2TProperties, IMA_SetInitialR2T.

Table 793: SMI Referenced Properties/Methods for CIM_iSCSISession

Property	Flags	Type	Description & Notes
ImmediateData		boolean	Maps to IMA_GetImmediateDataProperties, IMA_SetImmediateData.
MaxOutstandingR2T		uint32	Maps to IMA_GetMaxOutstandingR2TProperties, IMA_SetMaxOutstandingR2T.
MaxUnsolicitedFirstDataBurst-Length		uint32	Maps to IMA_GetMaxFirstBurstLengthProperties, IMA_SetMaxFirstBurstLength.
MaxDataBurstLength		uint32	Maps to IMA_GetMaxBurstLengthProperties, IMA_SetMaxBurstLength.
DataSequenceInOrder		boolean	Maps to IMA_GetDataSequenceInOrderProperties, IMA_SetDataSequenceInOrder.
DataPDUInOrder		boolean	Maps to IMA_GetDataPDUInOrderProperties, IMA_SetDataPDUInOrder.
ErrorRecoveryLevel		uint32	Maps to IMA_GetErrorRecoveryLevelProperties, IMA_SetErrorRecoveryLevel.
MaxConnectionsPerSession		uint32	Maps to IMA_GetMaxConnectionsProperties, IMA_SetMaxConnections.
DefaultTimeToWait		uint32	Maps to IMA_GetDefaultTime2WaitProperties, IMA_SetDefaultTime2Wait.
DefaultTimeToRetain		uint32	Maps to IMA_GetDefaultTime2RetainProperties, IMA_SetDefaultTime2Retain.

8.2.7.2.10 Related Standards

Table 794: Related Standards for iSCSI Initiator

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

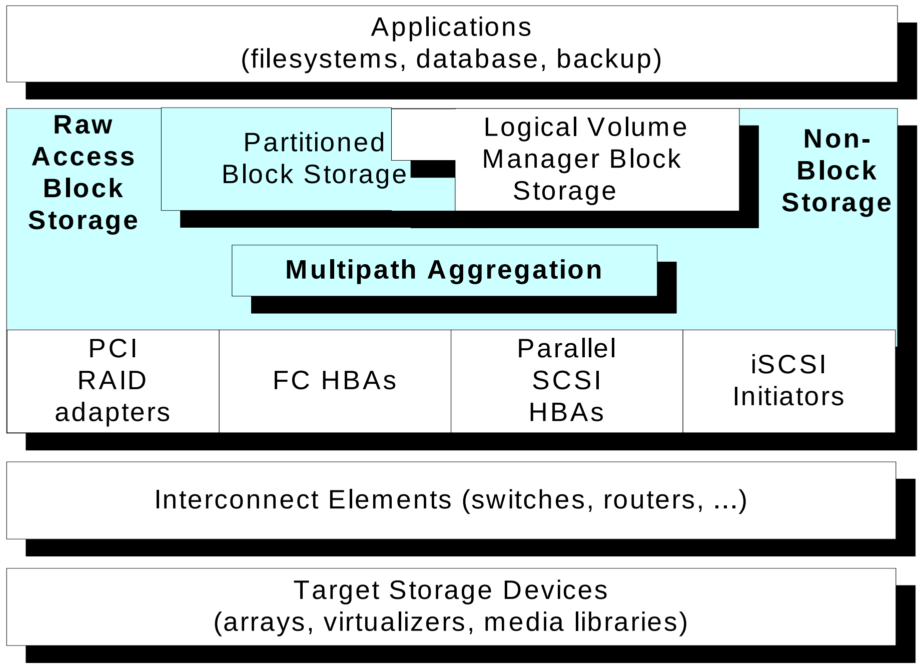
8.2.7.3 Host Discovered Resources Profile

8.2.7.3.1 Description

The Host Discovered Resources profile allows a client application to discover the storage hardware resources attached to a host system, the logical storage resources available through the OS, and the relationship between these hardware and logical resources. The hardware resources include host adapters and storage devices. The logical resources include the OS special files that represent storage devices. In some cases, there is a one-to-one relationship between the logical and physical device. But multipath and disk partitioning introduce resource fan-in and fan-out that are also modeled in this profile.

Figure 107: "Host Discovered Resources Block Diagram" depicts the relationship between the Host Discovered Resources profile and these other profiles. The areas with the shaded background are covered by the Host Discovered Resources profile – including partitioned and multipath storage.

Figure 107: Host Discovered Resources Block Diagram


Applications
(filesystems, database, backup)
Raw Access Block Storage
Partitioned Block Storage
Logical Volume Manager Block Storage
Non-Block Storage
Multipath Aggregation
PCI RAID adapters
FC HBAs
Parallel SCSI HBAs
iSCSI Initiators
Interconnect Elements (switches, routers, ...)
Target Storage Devices
(arrays, virtualizers, media libraries)

Applications and Logical Volume Manager are consumers of Host Discovered Resources. The diagram depicts how an application can use Logical Volume Manager resources or use Host Discovered Resources directly. For example, a server may have some filesystems using LVM volumes and some filesystems using OS volumes.

The blocks at the bottom of the diagram represent resources (HBAs and target devices) for which the Host Discovered Resources profile provides a host view. Note that interconnect elements between the HBAs and target devices are not part of the Host Discovered Resources profile.

The Host Discovered Resources Profile provides a minimal amount of information about the discovered hardware resources; this includes the connectivity and correlatable IDs. The Host Discovered Resources profile does not act as the canonical profile for any particular hardware resource; even host-resident elements like FC HBAs, iSCSI initiators, and Logical Volume Managers have separate profiles.

The correlatable IDs exposed by The Host Discovered Resources profile allow an application to associate host-discovered resources with resources from these other profiles.

For example, an array profile can describe the redundancy characteristics and performance statistics of a RAID volume. But the array profile will use a SCSI logical unit identifier as the volume’s name. By combining information from the array and host-discovered resources profiles, a client can display the host special file name(s) associated with that volume. This additional name information can help the administrator (or client software) determine which applications are associated with volumes.

Host Disk Extent Class Name Conventions

The Host Discovered Resources profile uses several different CIM classes to represent disk extents. **LogicalDisk** models an extent exposed by the OS to applications such as filesystems, databases, or logical volume managers. **GenericDiskPartition** represents a partition or slice of a disk as supported directly by the OS. **StorageVolume** represents disks or virtual volumes exported from disk arrays and virtualizers in the array or virtualizer profiles. **StorageExtent** represents disk extents that do fit these other classes; these will be intermediate extents that are neither consumed volumes nor exported logical disks.

Note that Logical Volume Managers are described in a separate profile. Logical Volume Managers may also expose partitions, but these are independent of partitions integrated into some OSes. The Host Discovered Resources profile just addresses OS partitions.

To make it easier for clients of this profile, all consumable storage exported by this profile are modeled as instances of LogicalDisk.

The functionality of host resources discovery is broken into three areas:

- Disk partition discovery and management. See 8.2.7.4, "Disk Partition Subprofile".
- Multipath Management. See 8.2.7.5, "SCSI Multipath Management Subprofile".
- DIscovery of Hardware Resouces. See “Discovered Hardware Resources” on page 714 in 8.2.7.3.1.

Discovered Hardware Resources

This profile presents a view of discovered resources with a common model based on the SCSI model, extended for different transports. Ports are modeled as instances of SCSIProtocolEndpoint - representing the SCSI (or ATA) protocol, not the physical interconnect.

The Host Discovered Resource profile could be implemented using standard APIs (such as the HBA API, or SNIA iSCSI Management API) to create a generic model of the host-storage controllers and storage attached to those controllers. The model includes elements also exposed by HBA and storage agents; the details are included in these other profiles. A client uses correlatable IDs to equate objects from different agents.

The correlatable IDs for logical units (LogicalDisk, StorageExtent, TapeDrive) are the identifiers assigned by the hosting operating system (see Table 8 for the name requirements for OS names of disk logical units). An implementation of this profile shall also provide the correlatable names associated with the underlying devices. The requirements specified in 6.2.4.5.1, "Standard Formats for Logical Unit Names" apply, but instead of using the Name and NameNamespace properties, the information is corresponding elements in the OtherIdentifyingInfo and IdentifyingDescriptions array properties. The valid strings for IdentifyingDescriptions are exactly those described for NameNamespace in 6.2.4.5.1, "Standard Formats for Logical Unit Names".

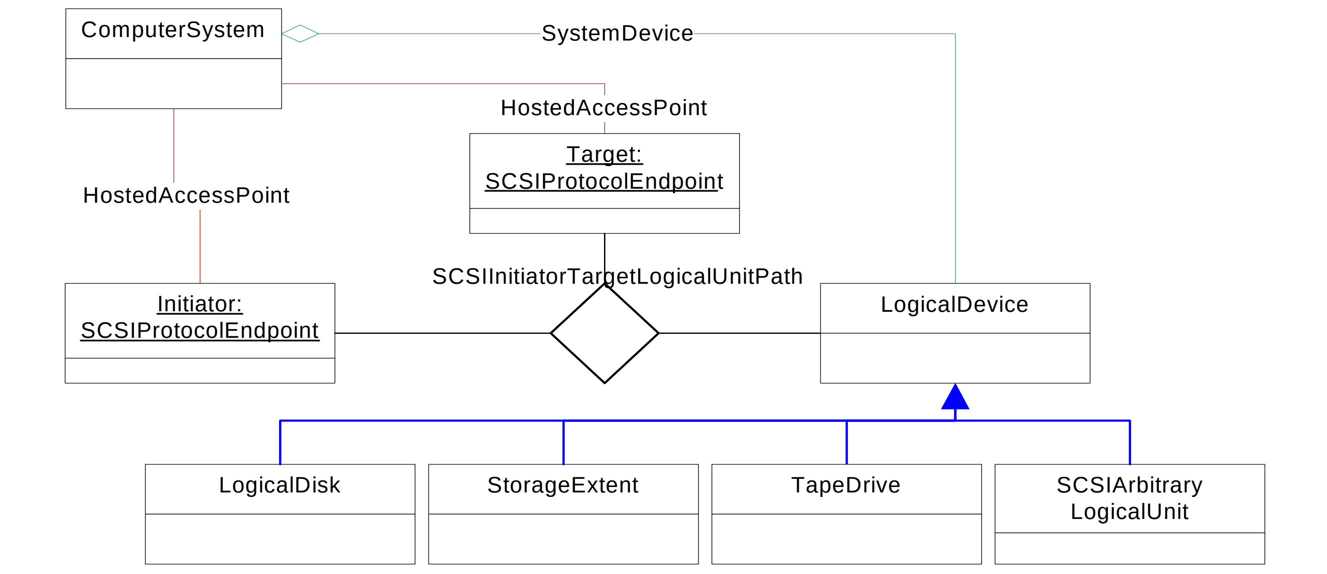
This profile is restricted to discovery of I/O devices and does not include remote filesystems. The SCSI and ATA models are discussed separately.

Model for SCSI Protocol Resources

The SCSI protocol is used in several transports - Fibre Channel, iSCSI, Parallel SCSI (SPI), and Serial Attached SCSI (SAS). SCSI Protocol includes initiator and target ports, and a logical units (RAID volumes, tape drives) in a many-to-many-to-many relationship - in other words, an initiator port may connect to many target ports (and vice versa), and each target device many have many logical units connected to initiator and target ports. “Figure 60: "Generic Initiator Port Model"” provides a general controller/device SCSI model. LogicalDevice subclasses model different types of SCSI logical unit, e.g., TapeDrive.

SCSIProtocolEndpoint represents the SCSI logical port, either initiator or target. The transports type (e.g., FC, iSCSI) is specified in SCSIProtocolEndpoint ConnectionType property.

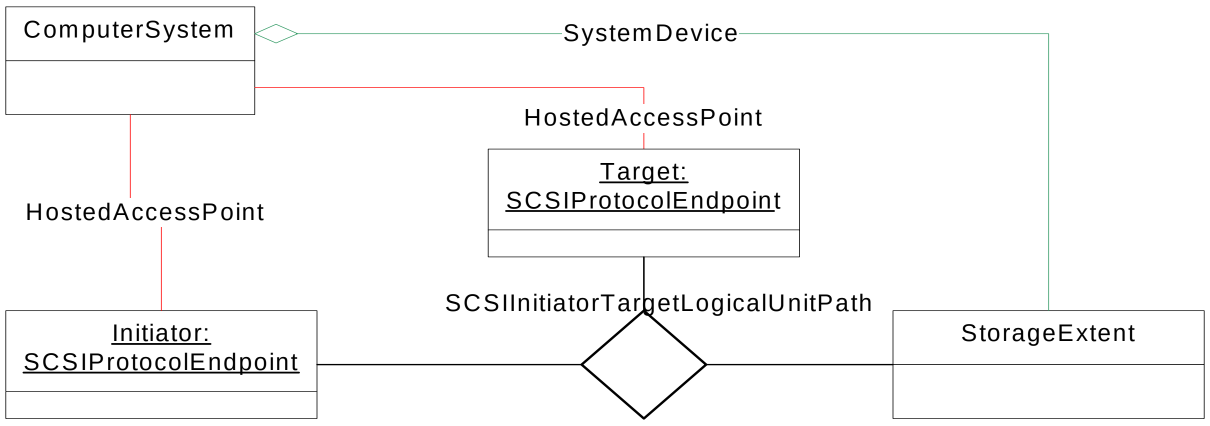
Figure 108: Host Discovered Resources Class Diagram


ComputerSystem
SystemDevice
HostedAccessPoint
Target: SCSIProtocolEndpoint
HostedAccessPoint
SCSIInitiatorTargetLogicalUnitPath
Initiator: SCSIProtocolEndpoint
LogicalDevice
LogicalDisk
StorageExtent
TapeDrive
SCSIArbitrary LogicalUnit

The initiator ProtocolEndpoint and each target ProtocolEndpoint and LogicalDevice are associated by SCSIInitiatorRagetLogicalUnitPath.

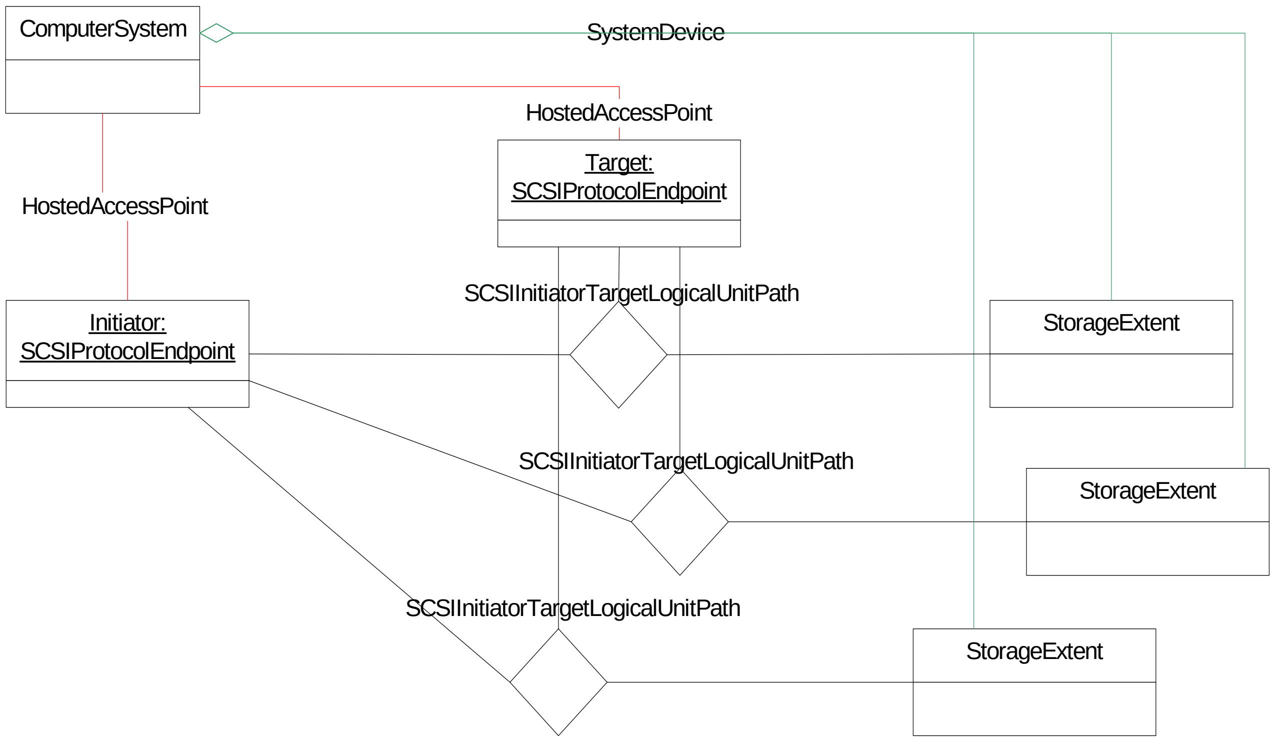
Consider a few concrete cases. The first is a single parallel SCSI disk. In general, Host APIs cannot differentiate a “real” disk from a virtual disk as exposed by a RAID controller, so the StorageExtent subclass of LogicalDevice is used.

Figure 109: Single SPI Disk Model


ComputerSystem
SystemDevice
HostedAccessPoint
Target: SCSIProtocolEndpoint
HostedAccessPoint
SCSIInitiatorTargetLogicalUnitPath
Initiator: SCSIProtocolEndpoint
StorageExtent

The second case is a Fibre Channel RAID controller exposing three virtual disks to a single host/ initiator port.There is a single initiator and target that share access to three StorageExtent instances.

Figure 110: Three FC Logical Unit Instance Diagram

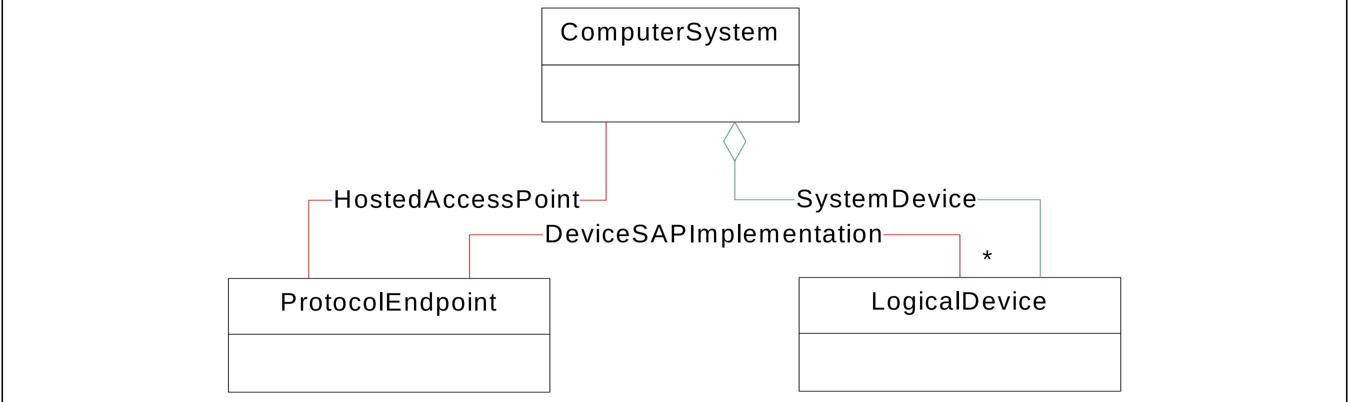

ComputerSystem
SystemDevice
HostedAccessPoint
Target: SCSIProtocolEndpoint
HostedAccessPoint
SCSIInitiatorTargetLogicalUnitPath
Initiator: SCSIProtocolEndpoint
StorageExtent
SCSIInitiatorTargetLogicalUnitPath
StorageExtent
SCSIInitiatorTargetLogicalUnitPath
StorageExtent

The Multipath Subprofile describes more complicated multipath configurations. See Figure 119: "Four Path Instance Diagram".

<u>Model for non-SCSI Protocol Resources</u>

The model for non-SCSI transports such as ATA is simpler because multipath support is not included and because there is a single controller (ProtocolEndpoint) rather than separate initiator and target controllers.

Figure 111: Non-SCSI Discovered Resource Model



<u>Associating Hardware and OS Devices</u>

There are two variations for disks and virtual disks - configurations with or without disk partitions.

1) With no partitions, each discovered (virtual) disk is modeled as LogicalDisk

2) With disk partitions, each partition exposed to an application or LVM is modeled as LogicalDisk. Any disk (or intermediate partition) that contains partitions is modeled as StorageExtent. DiskPartition instances are modeled between the StorageExtents and LogicalDisks. For more details, see the 8.2.7.4 "Disk Partition Subprofile". The requirement for disk partitions is reflected by the presence of DiskPartitionConfigurationCapabilities.

Tape drive configurations are similar to case 1 above, with TapeDrive rather than LogicalDisk.

8.2.7.3.2 Health and Fault Management Considerations

Not defined in this standard

8.2.7.3.3 Cascading Considerations

Not defined in this standard

Supported Subprofiles and Packages

Table 795: Supported Subprofiles for Host Discovered Resources

Registered Subprofile Names	Mandatory	Version
SCSI Multipath Management	No	1.1.0
Disk Partition	No	1.1.0

8.2.7.3.4 Extrinsic Methods of the Profile

StorageConfigurationService.SCSIScan

This method requests that the system rescan SCSI devices for changes in their configuration. If called on a general-purpose host, the changes are reflected in the list of devices available to applications (for example, the UNIX 'device tree').

This operation can be disruptive; optional parameters allow the caller to limit the scan to a single or set of SCSI device elements. All parameters are optional; if parameters other than Job are passed in as null, a full scan is invoked. If the caller specifies a connection type, the scan is limited to that connection type.

Job - a reference to a Job

ConnectionType - The type of connection (transport, such as FC or iSCSI), constrains the scan to initiator ports of this type. Only used if the Initiators parameter is null.

OtherConnectionType - The connection type if the ConnectionType parameter is Other.

Initiators - A list of references to initiators. Scanning will be limited to SCSI targets attached to these initiators. If this parameter is null and connection is specified, all initiators of that connection type are scanned. If this parameter and ConnectionType are null, all targets on all system initiators are probed.

Targets - A list of names or numbers for targets. These should be formatted to match the appropriate connection type. For example, PortWWNs would be specified for Fibre Channel targets.

LogicalUnits - A list of SCSI logical unit numbers representing logical units hosted on the targets specified in the Targets argument.

ScsiScan() support is optional. support for ScsiScan() can be determined based on the inclusion of “SCSI Scan” in the SupportedAsynchronousActions array in StorageConfigurationCapabilities.

8.2.7.3.5 Client Considerations and Recipes

8.2.7.3.5.1 Determine which exported extents are impacted by removal of a physical extent

```
//
// Description:
// Determine which exported extents are impacted by removal of a
// physical extent.  Note that in this Profile, “exported extent”
// is synonymous with LogicalDisk.
//
// Pre-Contitions:
// $Host holds a ref to the (top-level) ComputerSystem
// $Disk holds a reference to the StorageExtent to be removed.
//
// In SMI-S, anything exposed to applications (or LVMs) as an OS
// disk is modeled as LogicalDisk.  On platforms that support partitions,
// if a disk is partitioned, the disk itself is modeled as StorageExtent.
// Each partition that is exposed is modeled as LogicalDisk Based on a
// GenericDiskPartition BasedOn StorageExtent (the disk).  Some platforms
// allow a partition to be sub-partitioned; this is modeled as
// LogicalDisk (exposed) BasedOn DiskPartition (top-tier) BasedOn
// DIskPartition (bottom tier) BasedOn StorageExtent (the disk).
// On systems without disk partitions, a LogicalDisk instance models
// the entire usable disk capacity.
//
// CIM models each exposed partition as a LogicalDisk BasedOn a
// DiskPartition (mapped 1-1).  Many DiskPartitions can be based
// on the same underlying StorageExtent (either a disk or another
// partition).  The valid configurations
// are
// 1 - $Disk is actually exposed as a LogicalDisk (LD)
// 2 - Single-tier partitioning, LD based on DiskPartition (DP) BasedOn SE
//     (StorageExtent)
// 3 - Two-tier partitioning - LD BasedOn DP BasedOn DP BasedOn SE
//
/  The recipe below uses recursion to find all StorageExtents (the
// super-class of LogicalDisk and DiskPartition) based on $Disk, then
// follows BasedOn associations untill it hits LogicalDisks.


sub REF[] GetImpactedExtents($Extent)
{
        // A logical disk can’t contain any partitions - if $Extent
        // is a LogicalDisk, add it to the $ImpactedExtents list
        // and return.
     if ($Extent ISA CIM_LogicalDisk) {
          push ($ImpactedExtents[], $Extent)
        return ($ImpactedExtents[]->)
        }

        // For non LogicalDisks, get the list of all extents based on $Extent
     $SuperExtents[]  = AssociatorNames(
        $Disk,
        “CIM_BasedOn”,
        null,  // ResultClass
        “Antecedent”    // Role
        “Dependent”)// ResultRole

     // For each extent that depends on $Extent, recurse
     for #i in $SuperExtents[] {
         $ImpactedExtents = &GetImpactedExtents($SuperExtents[#i])
     }
     return $ImpactedExtents[]->
}

$ImpactedLDs = &GetImpactedExtents($Disk)
```

8.2.7.3.6 Registered Name and Version

Host Discovered Resources version 1.1.0

8.2.7.3.7 CIM Server Requirements

Table 796: CIM Server Requirements for Host Discovered Resources

Profile	Mandatory
Association Traversal	No
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.7.3.8 CIM Elements

Table 797: CIM Elements for Host Discovered Resources

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.7.3.8.1)	'Top level' system that hosts the resources.
CIM_HostedAccessPoint (8.2.7.3.8.2)	This association links all **SCSIProtocolEndpoints** to the scoping system.
CIM_LogicalDisk (8.2.7.3.8.3)	Represents a block logical unit that is exposed to applications such as file systems without being partitioned.
CIM_SCSIInitiatorTargetLogicalUnitPath (8.2.7.3.8.5)	Associates initiator and target SCSIProtocolEndpoints to a SCSI logical unit (LogicalDevice)
CIM_SCSIProtocolEndpoint (8.2.7.3.8.6)	
CIM_StorageExtent (8.2.7.3.8.7)	Represents a block logical unit in the host that is partitioned before being exposed to applications.
CIM_SystemDevice (8.2.7.3.8.8)	This association links all **LogicalDevices** to the scoping system.
CIM_TapeDrive (8.2.7.3.8.9)	Represents a tape drive logical unit in the host
Optional Classes	
CIM_SCSIArbitraryLogicalUnit (8.2.7.3.8.4)	A SCSI Logical Unit that exists only for management.

8.2.7.3.8.1 CIM_ComputerSystem

'Top level' system that hosts the resources.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 798: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	

Table 798: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Name	C	string	Unique identifier for the system. E.g., IP address.
ElementName		string	User-friendly name
NameFormat		string	Format for Name property.
Dedicated		uint16[]	Indicates that this computer system is not a dedicated storage system

8.2.7.3.8.2 CIM_HostedAccessPoint

This association links all **SCSIProtocolEndpoints** to the scoping system.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 799: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The Hosting System
Dependent		CIM_ServiceAccessPoint	The ProtocolEndpoint

8.2.7.3.8.3 CIM_LogicalDisk

Represents a block logical unit that is exposed to applications such as file systems without being partitioned.
Standard Names: The Name, NameFormat, and NameNamespace properties shall follow the requirements in 6.2.4.5.1 OtherIdentifyingInfo holds the correlatable ID of the underlying logical unit.
Class Mandatory: true

Table 800: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
ElementName		string	User-friendly name
Name	C	string	OS device name
NameFormat	C	uint16	
NameNamespace	C	uint16	
OtherIdentifyingInfo	C	string[]	The correlatable ID of the underlying logical unit
IdentifyingDescriptions	C	string[]	
OperationalStatus		uint16[]	

8.2.7.3.8.4 CIM_SCSIArbitraryLogicalUnit

A SCSI Logical Unit that exists only for management.
Standard Names: The Name property shall follow the requirements in 6.2.4.5.1 OtherIdentifyingInfo holds the correlatable ID of the underlying logical unit.
Class Mandatory: false

Table 801: SMI Referenced Properties/Methods for CIM_SCSIArbitraryLogicalUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
ElementName		string	User-friendly name
Name	C	string	OS device name
OtherIdentifyingInfo	C	string[]	The correlatable ID of the underlying logical unit
IdentifyingDescriptions	C	string[]	
OperationalStatus		uint16[]	

8.2.7.3.8.5 CIM_SCSIInitiatorTargetLogicalUnitPath

Associates initiator and target SCSIProtocolEndpoints to a SCSI logical unit (LogicalDevice)
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 802: SMI Referenced Properties/Methods for CIM_SCSIInitiatorTargetLogicalUnitPath

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
LogicalUnit		CIM_LogicalDevice	A reference to a LogicalDevice
Initiator		CIM_SCSIProtocolEndpoint	A reference to the initiator CIM_SCSIProtocolEndpoint
Target		CIM_SCSIProtocolEndpoint	A reference to the target CIM_SCSIProtocolEndpoint
AdministrativeWeight	M	uint32	
State		uint32	
AdministrativeOverride		uint16	
Optional Properties/Methods			
OSDeviceName		string	

8.2.7.3.8.6 CIM_SCSIProtocolEndpoint

Created By : Static
Modified By : Static
Deleted By : Static
Standard Names: The Name Property follows the requirements in 6.2.4.5.2

Class Mandatory: true

Table 803: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name	C	string	
ProtocolIFType		uint16	
ConnectionType		uint16	
Role		uint16	

8.2.7.3.8.7 CIM_StorageExtent

Represents a block logical unit in the host that is partitioned before being exposed to applications.
Standard Names: The Name, NameFormat, and NameNamespace properties shall follow the requirements in 6.2.4.5.1 OtherIdentifyingInfo holds the correlatable ID of the underlying logical unit.
Class Mandatory: true

Table 804: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
ElementName		string	User-friendly name
Name	C	string	OS device name
NameFormat	C	uint16	
NameNamespace	C	uint16	
OtherIdentifyingInfo	C	string[]	The correlatable ID of the underlying logical unit
IdentifyingDescriptions	C	string[]	
OperationalStatus		uint16[]	

8.2.7.3.8.8 CIM_SystemDevice

This association links all **LogicalDevices** to the scoping system.
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 805: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.7.3.8.9 CIM_TapeDrive

Represents a tape drive logical unit in the host

Standard Names: The Name property shall follow the requirements in 6.2.4.5.1 OtherIdentifyingInfo holds the correlatable ID of the underlying logical unit.

Class Mandatory: true

Table 806: SMI Referenced Properties/Methods for CIM_TapeDrive

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
ElementName		string	User-friendly name
Name	C	string	OS device name
OtherIdentifyingInfo	C	string[]	The correlatable ID of the underlying logical unit
IdentifyingDescriptions	C	string[]	
OperationalStatus		uint16[]	

8.2.7.3.9 Related Standards

Table 807: Related Standards for Host Discovered Resources

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.7.4 Disk Partition Subprofile

8.2.7.4.1 Description

This subprofile models partition (or slice) configuration services provided by operating systems on some platforms. Some operating systems do not use this type of partitioning. On the operating systems that do, the operating system disk drivers treat partitions as virtual disks. The types of valid partitions are determined by the operating system and the partitioning tools.

We need to consider several operating system variants related to operating system partitions

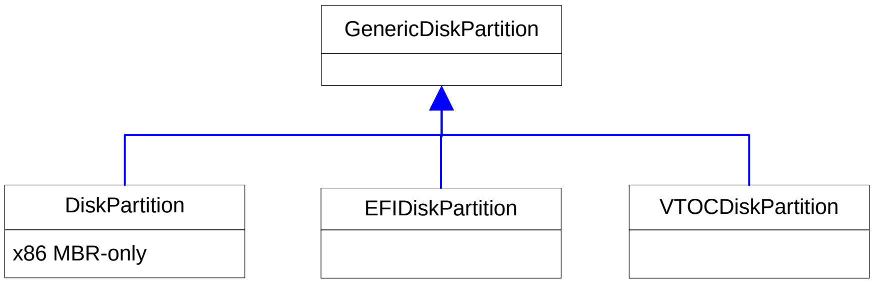
- On some platforms (e.g., Solaris, Windows), a raw disk volume needs to be partitioned before an application (i.e., a filesystem) uses it. There may be just a single partition on the volume. In these platforms, there is not a name that represents an entire disk volume if that disk volume has multiple partitions.
- On other platforms (e.g., Linux), an application resides on a partition or on the entire disk volume.
- Different operating systems have incompatible partitioning approaches and on-disk data structures (e.g disk labels or partition tables). This specification refers to these approaches as styles. Each style may be supported by multiple operating systems, and most operating systems support multiple styles. The styles supported in this subprofile are MBR (used on all operating systems running on X86 hardware), vtoc (Solaris and other operating systems with a BSD heritage), and EFI (an emerging style that supports multi-terabyte disk volumes).
- Some styles support multiple tiers of partitions - a partition at one tier may have sub-partitions. On Windows, extended partitions are also a second tier with MBR partitions at each tier.
- Some operating systems utilize two tiers of partitions with different styles at different tiers. For example, BSD-derived Unix variants running on X86 platforms: the lower tier is the X86 BIOS-supported MBR partitions; BSD-style slices can be installed on one of the MBR partitions.
- Some operating systems (AIX, HP_UX) have no equivalent to partitioning.
- Some partition styles have a fixed number of partitions (dependent on the partition type); the user can’t create or delete partitions, just adjust the properties of one of the pre-defined partitions.

A partitioned disk volume has an associated partition table. The partition table contains information about the partitions on the disk volume – the starting address, length, and (in some cases) the type of the partition. In certain cases, a partition table can be associated with a partition; allowing multiple tiers of partitions.

In order for storage applications (e.g., logical volume managers, filesystems, databases) to use a disk volume, the operating system provides a name for the volume. These names appear to be filenames but are part of one (or a few) special namespaces managed by the operating system. Windows drive letters and Unix /dev/ directories are examples of the special namespaces. Any extent that is consumable by storage applications is modeled a LogicalDisk; the LogicalDisk.Name property provides this special filename. The exported extent resulting from a partition is a LogicalDisk; on systems that do not require partitions, each usable disk volume has a LogicalDisk instance that models the operating system name. Extents that are not available for storage applications are modeled as StorageExtent (or StorageExtent subclasses other than LogicalDisk) instances and have a name derived from the underlying hardware and partition number.

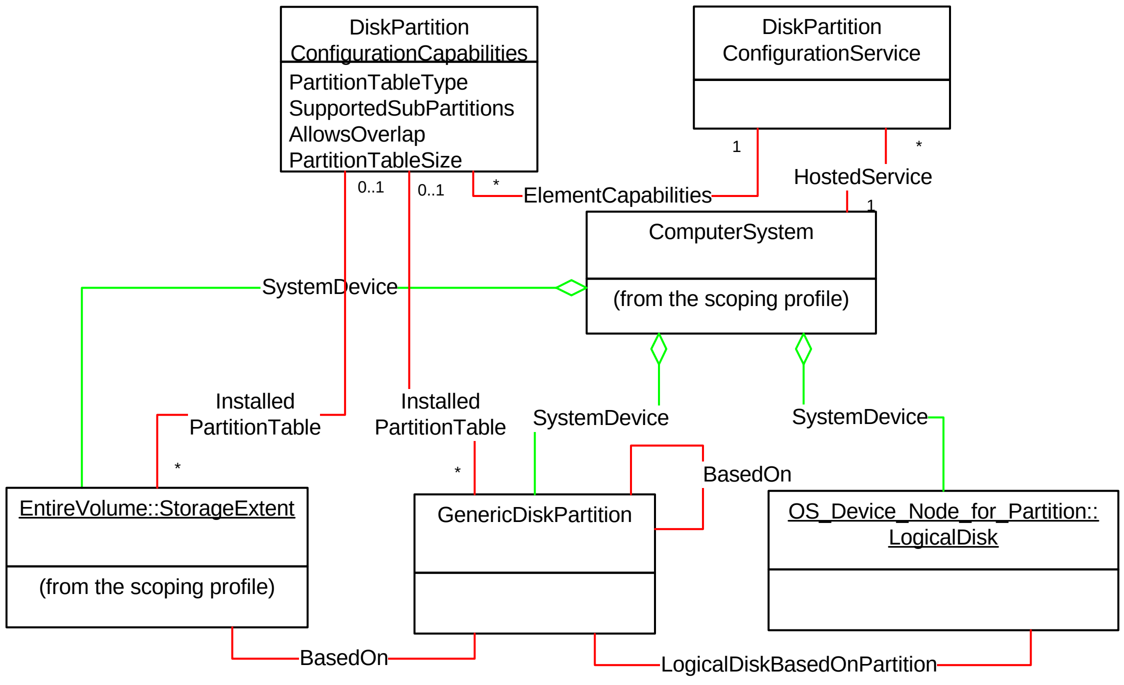
Operating systems may have different partition styles. The most common style is the MBR (Master Boot Record) style used on x86 PCs. This style supports four primary partitions on a disk volume with an optional second-tier (extended/logical partitions). Solaris uses a style called VTOC that is derived from and similar to BSD partitions. VTOC supports eight partitions. On Solaris X86, VTOC is installed in one X86 MBR primary partition for compatibility with other x86 operating systems. EFI is a new set of interfaces for x86 64-bit environments and includes a partitioning style. Of particular note is that EFI partitions can exceed the two-terabyte limit associated with other partition styles. So many vendors are migrating towards EFI as an option for supporting larger volumes. This profile includes separate specialized subclasses for MBR, VTOC, and EFI partitions. Their relationship is summarized in Figure 112: "Disk Partition Class Hierarchy".

Figure 112: Disk Partition Class Hierarchy



This profile includes a partition configuration service class that allows a client to create partition tables and modify partitions. It also includes a partition configuration capabilities class that describes the partition configuration capabilities of the system. Separate capabilities instances describe each partition style supported on the system. There should be at most one instance of DiskPartitionConfigurationService, as shown in Figure 113: "Disk Partition Class Diagram".

Figure 113: Disk Partition Class Diagram

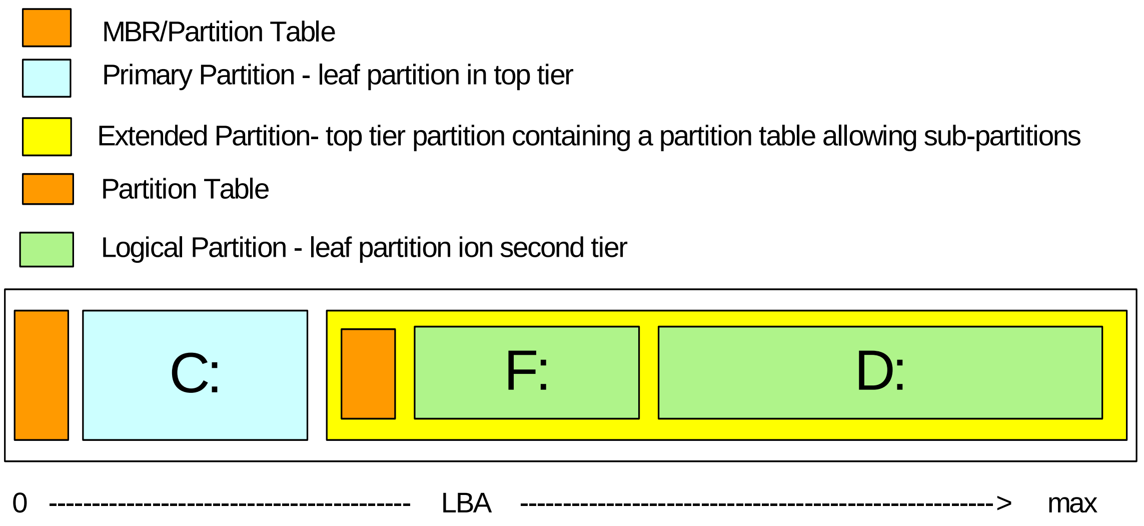


Background on X86 MBR Partitions

The terminology used in X86 partition applications is somewhat confusing and masks the actual configurations. The MBR style supports two tiers of partitions; up to four partitions at the entire disk volume tier and up to four partitions within each of these top-tier partitions. An MBR *primary* partition is a top-tier partition that is not sub-partitioned. An MBR *extended* partition is a top-tier partition that is sub-partitioned. An MBR *logical* partition is a sub-partition of an extended partition.

Figure 114: "Disk MBR Partition Example" represents the actual layout of an MBR drive with three usable partitions – with Windows/DOS driver letter names.

Figure 114: Disk MBR Partition Example


MBR/Partition Table
Primary Partition - leaf partition in top tier
Extended Partition- top tier partition containing a partition table allowing sub-partitions
Partition Table
Logical Partition - leaf partition ion second tier
C:
F:
D:
0 -- LBA --> max

C: is a primary partition and F: and D: are logical partitions that share an extended partition. Note that the partitions drive letters (C:, F:, and D:) are not in alphabetical order; the assignment of drive letters under Windows/DOS is decoupled from the partitioning logic.

Figure 115: "MBR Partition Instance Diagram" is an instance diagram of the SMI-S classes describing this configuration. Technically, the MBR/Partition tables could be considered to be small partitions. operating systems generally hide these sectors and treat the effective disk volume as starting just after the MBR. Rather than complicate the SMI-S model, these MBR areas are just ignored and the consumable block size is reduced by the appropriate value (the PartitionTableSize property of DiskPartitionConfigurationCapabilities). In the SMI-S model, the InstalledPartitionTable association to the containing extent indicates the presence of a disk label and/or partition table. In Figure 115: "MBR Partition Instance Diagram", the extent representing the entire disk volume (on the lower left) and the top-tier partition to the right each contain a partition table and are each associated to DiskPartitionConfigurationCapabilities via an InstalledPartitionTable association.

In Figure 115: "MBR Partition Instance Diagram" the StorageExtent at the lower left represents the entire disk volume and the two “top-tier” partitions are based on this extent. The LogicalDisk instances at the top represent the consumable partitions C:, F:, and D:.

Figure 115: MBR Partition Instance Diagram

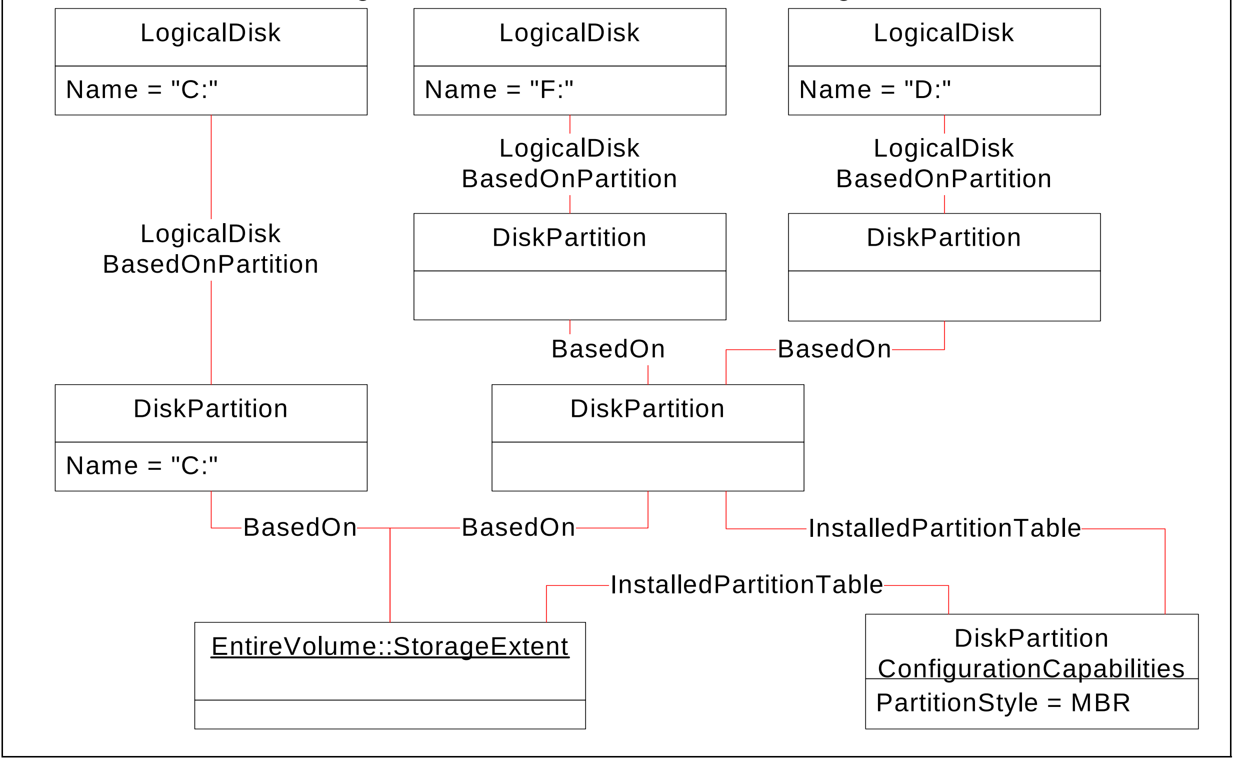


Figure 116: "MBR and VTOC Partition Instance Diagram" models a similar configuration where the one top-tier partition contains a Solaris X86 installation. In this case, the instrumentation instantiates two

instances of DiskPartitionConfigurationCapabilities, one for the top-tier MBR partition table and one for the vtoc partition table.

Figure 116: MBR and VTOC Partition Instance Diagram

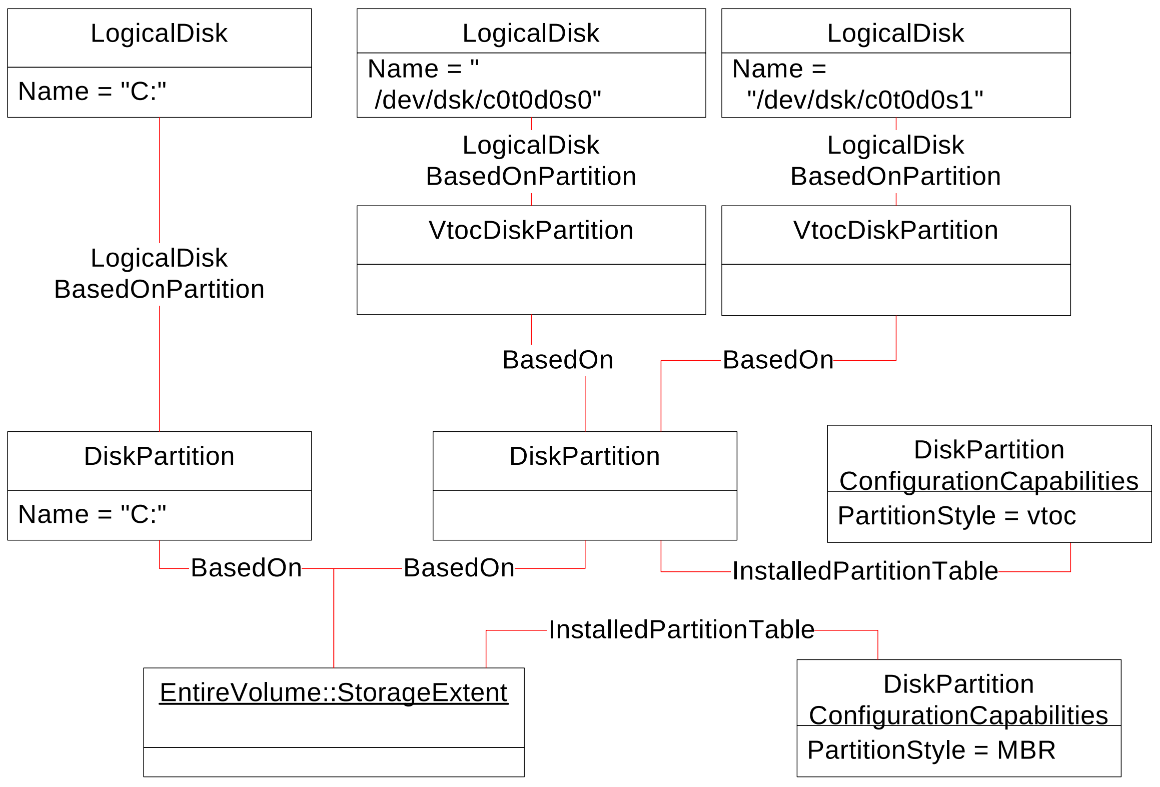

LogicalDisk
Name = "C:"
LogicalDisk
Name = " /dev/dsk/c0t0d0s0"
LogicalDisk
Name = "/dev/dsk/c0t0d0s1"
LogicalDisk BasedOnPartition
LogicalDisk BasedOnPartition
LogicalDisk BasedOnPartition
VtocDiskPartition
VtocDiskPartition
BasedOn
BasedOn
DiskPartition
Name = "C:"
DiskPartition
DiskPartition ConfigurationCapabilities
PartitionStyle = vtoc
BasedOn
BasedOn
InstalledPartitionTable
InstalledPartitionTable
EntireVolume::StorageExtent
DiskPartition ConfigurationCapabilities
PartitionStyle = MBR

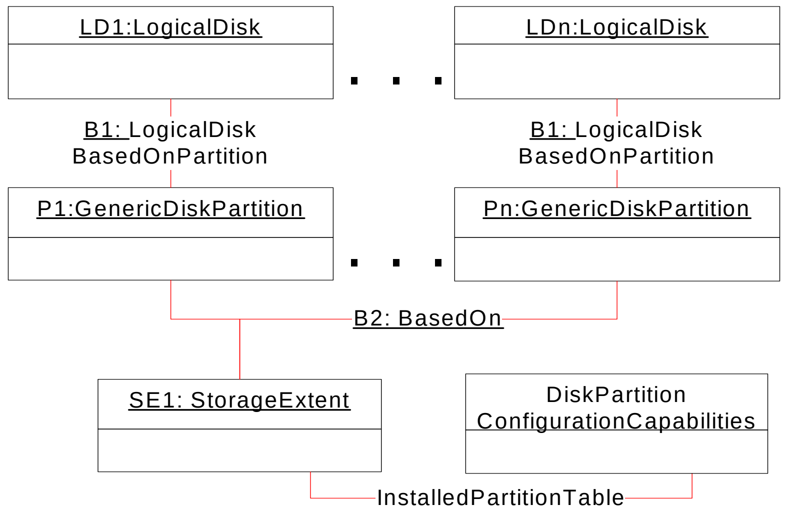
Table 808 summarizes likely values for capabilities properties and suggested Name properties on various operating systems

Table 808: Capabilities Properties

Property	X86 MBR	vtoc	EFI			
			Win	Linux	Solaris SPARC	Solaris X86
Overlap Allowed	Depends on applications	true	false			
MaxCapacity	2 terabytes (2^32 blocks)	2 terabytes	2^64 blocks			
MaxNumberOfPartitions	4	8	128	15	127	127

The sizes and starting/ending addresses shall be consistent between the associated LogicalDisk, DiskPartition, and LogicalDisk instances. Figure 117: "Partition Instance Diagram for Size/Address Rules" shows the classes with size information.

Figure 117: Partition Instance Diagram for Size/Address Rules



In this diagram, partitions P1,... Pn are all based on the same underlying disk volume (or partition) SE1.

- The NumberOfBlocks shall be the same for a LogicalDisk and its underlying partition (for example, LD1 and P1 in the diagram).
- The StartingAddress in the LogicalDiskBasedOnPartition associations (B1 in the diagram) between a LogicalDisk and its underlying partition will be 0. The EndingAddress in this association shall be one less than NumberOfBlocks from either the LogicalDisk or partition.
- The NumberOfBlocks for each partition (P1, ... Pn) shall be equal to the values of EndingAddress-StartingAddress+1 of the underlying LogicalDiskBasedOnPartition association (B2 in the diagram).
- DiskPartitionConfigurationCapabilities.PartitionTableSize shall hold the total number of blocks consumed by metadata (volume label, boot record, and partition tables) for the associated StorageExtent. For MBR and VTOC styles, this is a fixed value. For EFI, this value could in theory be larger for large extents. Separate instances of DiskPartitionConfigurationCapabilities shall be instantiated as needed to allow different values of PartitionTableSize.
- The size of maintenance tracks or cylinders shall not be included StorageExtent.NumberOfBlocks. This size may be included in DiskPartitionConfigurationCapabilities.PartitionTableSize.
- If DiskPartitionConfigurationCapabilities.OverlapAllowed is false, then the sum of the NumberOfBlocks properties for all partitions plus DiskPartitionConfigurationCapabilities.PartitionTableSize shall not exceed the value of NumberOfBlocks for the underlying StorageExtent. Other than that, there is no guaranteed relationship between StorageExtent.NumberOfBlocks and the sum of the NumberOfBlock values for partitions BasedOn the StorageExtent.

8.2.7.4.2 Health and Fault Management Considerations

No health information is required in LogicalDisk or partition instances. Clients should assume that the health-related properties of the underlying StorageExtent apply to all partitions and LogicalDisks based on that extent.

8.2.7.4.3 Supported Subprofiles and Packages

None

8.2.7.4.4 Methods of the Profile

8.2.7.4.4.1 SetPartitionStyle

This method installs a partition table on an extent of the specified partition style, creates DiskPartition instances if SettingStyleInstantiatedPartitions is non-zero, and BasedOn associations between the underlying extent and the new partition instances. As a side effect, the usable block size of the underlying extent is reduced by the block size of the metadata reserved by the partition table and associated metadata. This size is in the PartitionTableSize property of the associated DiskPartitionConfigurationCapabilities instance.

```
uint32 SetPartitionStyle (

      [IN, Description (
          "A reference to the extent (volume or partition) where "
          "this style (partition table) will be installed.")]
   CIM_StorageExtent REF Extent,

      [IN, Description (
          "A reference to the "
          "DiskPartitionConfigurationCapabilities instance "
          "describing the desired partition style.")]
   CIM_DiskPartitionConfigurationCapabilities REF PartitionStyle );
```

8.2.7.4.4.2 CreateOrModifyPartition

This method creates a new partition if the Partition parameter is null or modifies the partition specified. If the starting and ending address parameters are null, the resulting partition will occupy the entire underlying extent. If a the DeviceFileName parameter is non-null, a LogicalDisk instance is created and associated via LogicalDiskBasedOnPartition to the partition. The underlying extent shall be associated to a capabilities class describing the installed partition style (partition table); this association is established using

```
uint32 CreateOrModifyPartition (
       [IN, Description (
           "A reference to the underlying extent the partition is "
           "base on.")]
    CIM_StorageExtent REF extent,

       [IN, Description (
           "The starting block number.")]
    uint64 StartingAddress,

       [IN, Description (
           "The ending block number.")]
    uint64 EndingAddress,

       [IN, Description (
           "The platform-specific special file name to be assigned "
           "to the LogicalDisk instance BasedOn the new "
```

```
                "DiskPartition instance.")]
         string DeviceFileName,

            [IN, OUT, Description (
                "A reference an existing partition instance to modify or "
                "null to request a new partition.")]
         CIM_GenericDiskPartition REF Partition);
```

```
Delete Instance to delete DiskPartition (and everything south)
```

8.2.7.4.5 Client Considerations and Recipes

A client discovers partition configuration support by looking for instances of DiskPartitionConfigurationService. If no service instances are available, then this operating system does not support disk partitions and the client can assume that any LogicalDisk instance is consumable by applications (such as volume managers or filesystems). For operating systems that do support partitioning, the client can discover whether a particular extent is partitioned by looking for a InstalledPartitionTable instance associated with the extent. The client can discover the existing partition configuration by following BasedOn associations between the extent and GenericDiskPartition instances.

For each discovered service, there shall be one or more instances of DiskPartitionConfigurationCapabilities. There is exactly one capabilities instance per partition table type. If multiple capabilities instances are discovered, the client should look at the SupportedExtentTypes property to determine the services that apply to entire disk volumes and those that apply to partitions.

8.2.7.4.5.1 Create New Partition Using All Available Space at End of Volume

```
//
// Description:
// Create New Partition Using All Available Space at End of Volume
//
// Preconditions:
// $Host holds a ref to the (top-level) ComputerSystem
// $Disk holds a reference to the LogicalDisk (or StorageExtent) instance
// representing the disk or disk volume.  $Disk must either be “raw”
// (no volume label), or have some partitioned space at the end.

// Locate instances of CIM_DiskPartitionConfigurationService.
// Note that HDR does not support the multiple system SP,
// so all services must be hosted on $Host.

$Services = AssociatorNames($Host,
     “CIM_HostedService”,
     “CIM_DiskPartitionConfigurationService”,
     “Antecedent”,// Role
     “Dependent”)// Result Role
// If no service instances are found, then this platform does
// not support partitioning - so exit.
if ($Service->[].size == 0) {
  <EXIT This system does not support SMI-S disk partitioning>
  }

// Look for CIM_DiskPartitionConfigurationCapabilities
// associated to $Disk.
$Capabilities->[] = AssociatorNames($Host->,// ObjectName
     “CIM_ElementCapabilities”,// AssocClass
     “CIM_DiskPartitionConfigurationCapabilities”,// ResultClass
     “ManagedElement”,   // Role
     “Capabilities”)    // ResultRole

if ($Capabilities != null && $Capabilities->[].size > 1) {
    <ERROR - must not be more than 1
      CIM_DiskPartitionConfigurationCapabilities
         associated with an extent>

#CreateOneBigPartition = false
if ($Capabilities == null || $Capabilities->[].size == 0 ) {
    // No Capabilities instance found assocaited to $Disk, this disk has no
    // volume label, create a label with SetPartitionStyle() using
    // the first service instance found.

    // Locate the first Capabilities instance associated with the
    // service.  If none, then error.
    $Capabilities->[] =
    // If no capabilities associated to service, then error exit

    %InArguments[“Extent”] = $Disk
    %InArguments[“Capabilities”] - $Capabilities->[0]
    #MethodReturn = $Services[0]->InvokeMethod(
         $Service->[0],
         “SetPartitionStyle”,
         %InArguments)
    if (#MethodReturn != 0) {
        <ERROR - SetPartitionStyle non-zero method return>
    }
    #CreateOneBigPartition = true;
    }

// locate partitons based on this disk
$BasedOns[] = References(
     $Disk->[],
     “CIM_BasedOn”, // Assoc class
     “Antecedent”,// my role
     false,
     false,
     {“StartingAddress”, “EndingAddress”})
    if ($BasedOns[] == null || $BasedOns->[].size == 0) {
     // If $Disk has no associated partitions, create one using
     // entire disk with CreateOrModifyPartition()
     #CreateOneBigPartition = true;
    }

if (#CreateOneBigPartition == true) {
     // null starting and ending address parameters mean
     // “use entire disk”.
     // null Partition REF parameter means Create
        %InArguments[“Extent”] = $Disk
        // all other parms default to “use entire extent”
     #MethodReturn = $Services[0]->InvokeMethod(
         $Services->[0],
         “CreateOrModifyPartition”,
         %InArguments)
        if(#MedthodReturn != 0)  {
            <ERROR! CreateOrModifyPartition full disk method Failed >
        }
    }


// Look for available space at end of disk
// Note that the order of partitions in $BasedOns is not necessarily
// the same as the order of the addresses in the partitions.
#CreatePartPossible = true;
// LastBlockInParts in the highest block address in any partition
#LastBlockInParts = $Capabilities.PartitionTableSize
$Capabilities = <get capabilities instance associated with this disk>
for (#i in $BasedOns->[]) {
    // if this partition goes to the end of the underlying extent ...
    if ($BasedOns[#i].EndingAddress == $Disk.NumberOfBlocks-1) {
     // if OverlapAllowed and this partitions takes up entire
     // consumable disk space, then this is a special backup
     // partition - the condition below is the opposite...
     if ((!$Capabilities.OverlapAllowed) ||
       ($BasedOns->[#i].StartingAddress >
             $Capabilities.PartitionTableSize)) {
            #CreatePartPossible = false
     }
    } else {
        // This partition ends after others we’ve seen (LastBlockInParts)
        // Update LastBlockInParts with the new address
     if ($BasedOns[#i].EndingAddress > #LastBlockInParts) {
       #LastBlockInParts = $BasedOns[#i].EndingAddress
     }
    }
}
if (#CreatePartPossible) {
    if ($BasedOns->[].size() >= $Capabilities.MaxNumberOfPartitions) {
        // then we can’t create any more partitions - exit
        exit
    } else {
     // Get the service associated with $Capabilities
        $Service = ..
        $Services = AssociatorNames($Host,
         “CIM_InstalledPartitionTable”,
            “CIM_DiskPartitionConfigurationService”,
         “Antecedent”,// Role
         “Dependent”)// Result Role
        %InArguments[“Extent”] = $Disk;
     %InArguments[“StartingBlock”] = #LastBlockInParts + 1
        // EndingBLock will default to end of disk
     #MethodReturn = InvokeMethod(
         $Services->[0],
         “CreateOrModifyPartition”,
         %InArguments)

        if(#MedthodReturn != 0)  {
            <ERROR! CreateOrModifyPartition park disk method Failed >
        }
    }
} else {
    <EXIT - no space at end of disk>
}
```

8.2.7.4.6 Registered Name and Version

Disk Partition version 1.1.0

8.2.7.4.7 CIM Server Requirements

Table 809: CIM Server Requirements for Disk Partition

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.7.4.8 CIM Elements

Table 810: CIM Elements for Disk Partition

Element Name	Description
Mandatory Classes	
CIM_BasedOn (8.2.7.4.8.1)	
CIM_DIskPartitionConfigurationCapabilities (8.2.7.4.8.2)	
CIM_DIskPartitionConfigurationService (8.2.7.4.8.3)	
CIM_ElementCapabilities (8.2.7.4.8.4)	
CIM_GenericDiskPartition (8.2.7.4.8.5)	
CIM_HostedService (8.2.7.4.8.6)	
CIM_InstalledPartitionTable (8.2.7.4.8.7)	
CIM_LogicalDisk (8.2.7.4.8.8)	
CIM_LogicalDiskBasedOnPartition (8.2.7.4.8.9)	
CIM_StorageExtent (8.2.7.4.8.10)	
CIM_SystemDevice (8.2.7.4.8.11)	
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_GenericDiskPartition	Partition Creation
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_GenericDiskPartition	Partition Deletion
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_GenericDiskPartition	Partition Modification

8.2.7.4.8.1 CIM_BasedOn

Created By : Static or External
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: true

Table 811: SMI Referenced Properties/Methods for CIM_BasedOn

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageExtent	
Dependent		CIM_StorageExtent	
StartingAddress		uint64	
EndingAddress		uint64	

8.2.7.4.8.2 CIM_DIskPartitionConfigurationCapabilities

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

Class Mandatory: true

Table 812: SMI Referenced Properties/Methods for CIM_DIskPartitionConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
PartitionStyle		uint16	
ValidSubpartitionStyles		uint16[]	
MaxNumberOfPartitions		uint16	
MaxCapacity		uint64	
OverlapAllowed		boolean	
PartitionTableSize		uint32	

8.2.7.4.8.3 CIM_DIskPartitionConfigurationService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 813: SMI Referenced Properties/Methods for CIM_DIskPartitionConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SetPartitionStyle()			
CreateOrModifyPartition()			

8.2.7.4.8.4 CIM_ElementCapabilities

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 814: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Capabilities		CIM_Capabilities	

8.2.7.4.8.5 CIM_GenericDiskPartition

Created By : Static or External
Class Mandatory: true

Table 815: SMI Referenced Properties/Methods for CIM_GenericDiskPartition

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	

Table 815: SMI Referenced Properties/Methods for CIM_GenericDiskPartition

Property	Flags	Type	Description & Notes
DeviceID		string	
Name		string	
OperationalStatus		uint16[]	
Optional Properties/Methods			
NumberOfBlocks		uint64	

8.2.7.4.8.6 CIM_HostedService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true
No specified properties or methods.

8.2.7.4.8.7 CIM_InstalledPartitionTable

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 816: SMI Referenced Properties/Methods for CIM_InstalledPartitionTable

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_DiskPartitionConfigurationCapabilities	
Dependent		CIM_StorageExtent	

8.2.7.4.8.8 CIM_LogicalDisk

Created By : Static or External
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: true

Table 817: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Name		string	
NameFormat		uint16	OS Device Name
NameNamespace		uint16	OS Device Namespace
OperationalStatus		uint16[]	

Table 817: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
NumberOfBlocks		uint64	

8.2.7.4.8.9 CIM_LogicalDiskBasedOnPartition

Created By : Static or External
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: true

Table 818: SMI Referenced Properties/Methods for CIM_LogicalDiskBasedOnPartition

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_GenericDiskPartition	
Dependent		CIM_LogicalDisk	

8.2.7.4.8.10 CIM_StorageExtent

Class Mandatory: true

Table 819: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Name		string	
OperationalStatus		uint16[]	
Optional Properties/Methods			
NumberOfBlocks		uint64	

8.2.7.4.8.11 CIM_SystemDevice

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 820: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.7.4.9 Related Standards

Table 821: Related Standards for Disk Partition

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.7.5 SCSI Multipath Management Subprofile

8.2.7.5.1 Description

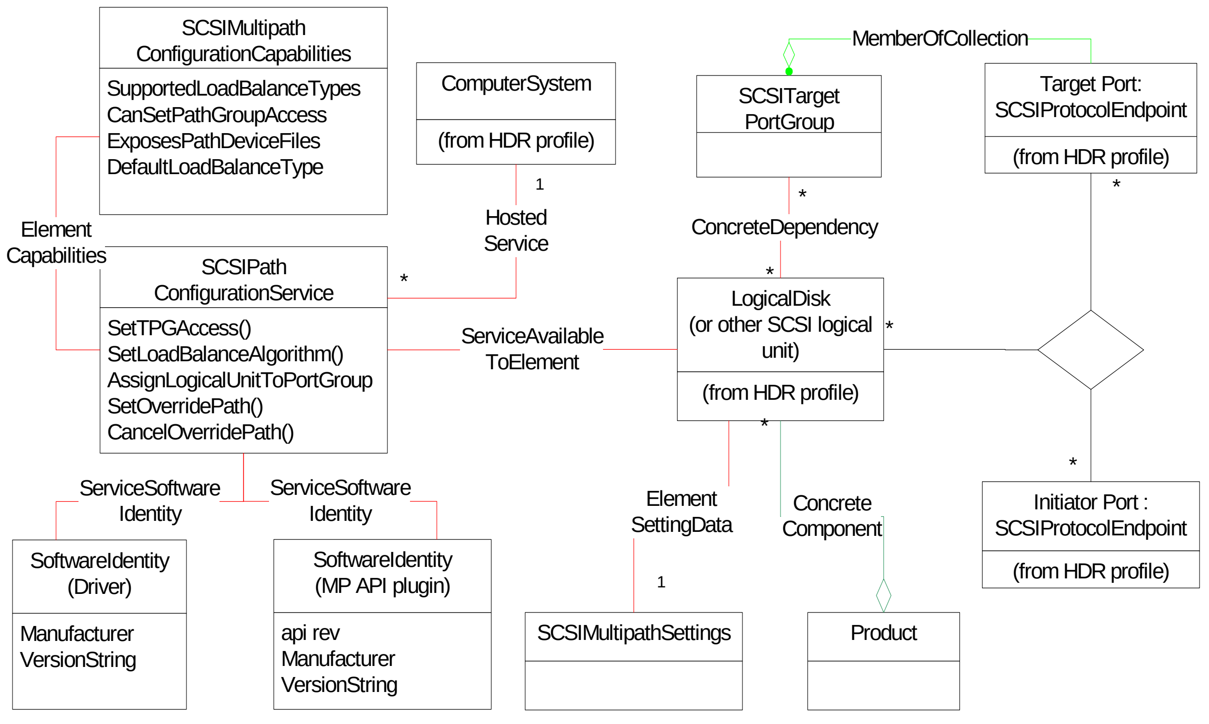
Multipath access to SCSI devices is handled in a similar way on many operating systems. As viewed from host adapters, each combination of host adapter (initiator) port, target device port, and logical unit appears to be a separate logical unit. For example, each path to a multipath device appears to be a separate device. Multipath drivers aggregate these into a single device that acts to storage applications like a single path device, but provides administrative interfaces for load balancing and failback.

Host Discovered Resources incorporates multipath logic as part of the mapping from logical (operating system) resources to hardware resources. If the discovered block storage has a single path, then LogicalIdentity associates the discovered StorageVolume instance with the OS/Partition StorageExtent/ LogicalDisk representing the underlying volume. The subclass of StorageExtent follows the extent naming conventions described in “Host Disk Extent Class Name Conventions” in 8.2.7.3.1.

The rest of the examples in this section use LogicalDisks since multipath disk arrays are more common, but the same approach can be extended to other storage types. For example, a TapeDrive can model multipath access to a tape drive.

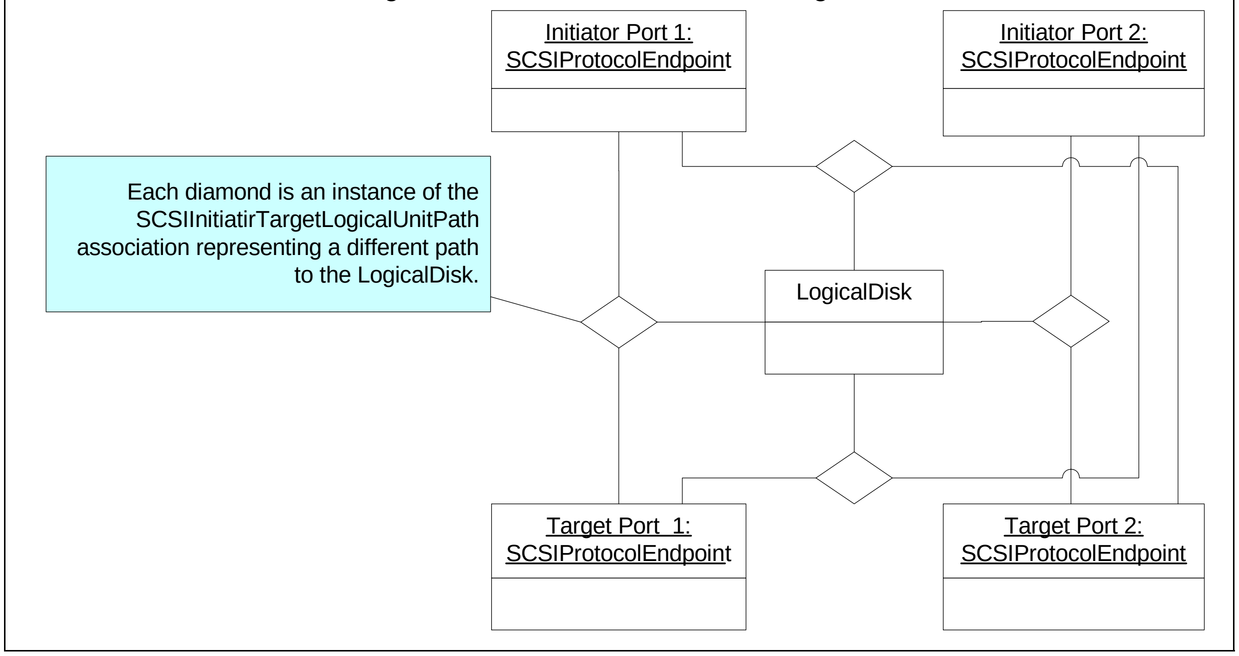
MultipathConfigurationCapabilities allows clients to determine which features and capabilities are exposed. SCSIPathConfigurationService may provide methods for management load balancing and failback. A system may have multiple multipath drivers with different capabilities and interfaces – each driver is modeled with a separate instance of MultipathConfigurationCapabilities and SCSIPathConfigurationService, as illustrated in Figure 118: "Multipath Management Class Diagram".

Figure 118: Multipath Management Class Diagram



.Figure 119: "Four Path Instance Diagram" shows the relationship of target and initiator ports

Figure 119: Four Path Instance Diagram


Initiator Port 1: SCSIProtocolEndpoint
Initiator Port 2: SCSIProtocolEndpoint
Each diamond is an instance of the SCSIInitiatirTargetLogicalUnitPath association representing a different path to the LogicalDisk.
LogicalDisk
Target Port 1: SCSIProtocolEndpoint
Target Port 2: SCSIProtocolEndpoint

(SCSIProtocolEndpoint instances) and a disk (LogicalDisk) with four paths. SCSIInitiatorTargetLogicalUnitPath instances represent each path and associate each permutation of initiator SCSIProtocolEndpoint, target SCSIProtocolEndpoint, and the LogicalDisk.

Asymmetric Multipath Target Devices

Some devices implement asymmetric multipath access, i.e., in non-failover mode, each LUN is only available through certain target ports, but can be access through other ports during failover. The SMI-S model uses the SPC-3 interface for asymmetric access. This model has target port groups – collections of target ports sharing a common access state for a group of logical units. Mutipath drivers for asymmetric access devices optionally provide an interface to “failback” after a failover condition has been corrected. The SMI-S interface follows the SPC-3 interface; the caller shall specify the desired access state for each target port group (TargetPortGroup). This interface is the SetTPGAccess method of SCSIPathConfigurationService. Driver support for this method (and other methods and capabilities) is indicated by properties of MultipathConfigurationCapabilities.

In the past, devices exposed vendor-specific SCSI multipath interfaces, so drivers with device-specific logic were shipped with target devices, logical volume managers, and HBAs. The SPC-3 have been enhanced to allow more interoperability and operating systems are including multipath support for any target that complies with the standards. However, there are still cases where a single customer host includes multiple multipath drivers, each with different capabilities and interfaces. And a single target device may be connected in such a way that multiple multipath drivers are involved at multiple places in the driver stack.

The SNIA Multipath Management API provides an interoperable interface to multipath driver features. Each multipath driver includes a corresponding plug-in for the multipath API. The SNIA Multipath Management Subprofile utilizes the Multipath API to interface to each multipath driver and provide all the associations from the discovered hardware resources to the consumable operating system resources.

The instrumentation shall instantiate SCSIInitiatorTargetLogicalUnitPath instances representing each path to SCSI logical units (LogicalDevice subclasses) attached to the hosting system.

The instrumentation shall instantiate at least one instance of SCSIMultipathConfigurationCapabilities for each multipath API plug-in registered on the system.

If the multipath API plug-ins provide support for interfaces to change load balancing and force failover, the instrumentation should support these methods.

8.2.7.5.2 Health and Fault Management Considerations

This subprofile specifies logical paths between elements (ports and logical units). The health and fault management information for these elements is specified in the profiles for those elements - for example, port subprofiles.

8.2.7.5.3 Cascading Considerations

None.

8.2.7.5.4 Supported Subprofiles and Packages

None

8.2.7.5.5 Methods of the Profile

All methods are part of SCSIPathConfigurationService and are optional.

SCSIPathConfigurationService.SetTPGAccess

This method allows a client to manually failover or failback. The parameters are:

- LogicalDevice - A reference to an instance of a subclass of LogicalDevice representing the SCSI logical unit where the command shall be sent.
- TargetPortGroups - Array of references to instances of SCSITargetPortGroup. All the referenced TargetPortGroup instances shall be part of the same target device
- AccessStates[] - An array of desired access states. Each access state in this array is the desired access state for the SCSITargetPortGroup in the corresponding entry in the TargetPortGroups parameter. The Active value is not part of SPC-3; it is a convenience for clients that are not sure whether to specify Active/Optimized of Active/Non-optimized. The instrumentation selects a value based on historic information, knowledge of the target configuration, or trial and error. Note that SCSITargetPortGroup.AccessState includes the value 'Transitioning' that is excluded here - a caller cannot request transitioning, though it may be reported by a target device.

SCSIPathConfigurationService.SetLoadBalanceAlgorithm

This method requests that the target change the load balance algorithm for the referenced LogicalDevice instance. The parameters are

- LogicalDevice - a reference to an instance of a subclass of LogicalDevice representing a SCSI logical unit.
- LoadBalanceAlgorithm - The desired load balance algorithm - possible values are “Unknown”, “Other”, “No Load Balancing”, “Round Robin”, “Least Blocks”, “Least IO”, or “Product Specific”
- OtherLoadBalanceAlgorithm - When LoadBalanceAlgorithm is 'Other', this parameter specifies a description of the load balancing algorithm. When LoadBalanceAlgorithm is 'Product Specific', this property provides a string specifying the vendor/product/version of the ManagedElement.

SCSIPathConfigurationService.AssignLogicalUnitToPortGroup

This method allows an administrator to assign a logical unit to a target port group. Each LU is typically associated with two target port groups, one in active state and one in standby state. The result of this method is that the LU associations change to a pair of target port groups. Only valid if the target device supports asymmetric access state and SCSIMultipathConfigurationCapabilities SupportsLuAssignment is set. The parameters are:

- LogicalDevice - a reference to an instance of a subclass of LogicalDevice representing a SCSI logical unit.
- TargetPortGroup - A reference to a target port group. The Target Port Group should be in an active state.

SCSIPathConfigurationService.SetOverRidePath

This method allows an administrator to temporarily disable load balancing for a specific logical unit. The path specified as a parameter shall have its AdministrativeOverride property set to 'Overriding' and all I/O to the logical unit shall be directed to this path. All other paths to this logical unit shall have AdministrativeOverride set to 'Overridden'. There is one parameter:

- Path - A reference to a SCSIInitiatorTargetLogicalUnitPath.

SCSIPathConfigurationService.CencelOverRidePath

This method clears an override path as set in SetOverridePath and load balancing is enabled. All paths to the logical unit specified as a parameter shall have AdministrativeOverride property set to 'No override in effect'. There is one parameter:

- LogicalUnit -A reference to a SCSIInitiatorTargetLogicalUnitPath.

After an override is canceled, the previous load balance algorithm should be restored.

8.2.7.5.6 Client Considerations and Recipes

8.2.7.5.6.1 Discover All Paths to a Disk Volume

```
)// DESCRIPTION
//
// This recipe discovers the topology of HW resources attached to the
// current host system. Host controllers (HBAs) and attached volumes
// (disks) supporting SCSI proptocol are reported.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
//
// 1. A reference to the top-level ComputerSystem in the HDR Profile
//    is known as $Host->
//

// Step 1. Get name(s) of the SCSIProtocolEndpoints representing
// SCSI initiators on the host system.
//
$SPEs->[] = AssociatorNames($Host->,// ObjectName
     “CIM_HostedAccessPoint”,// AssocClass
     “CIM_SCSIProtocolEndpoint”,// ResultClass
     “Antecedent”,   // Role
     “Dependent”)    // ResultRole

$Initiators->[] = <get the subset of $SPEs with Role = “Initiator”>
```

```
if ($Initiators->[] == null || $Initiators->[].length == 0) {
     <EXIT: No SCSI Initiators on the host system!>
}

// Determine the topology of inititors, targets, and volumes.
//
for (#i in $Initiators->[]) {
     // Step 2. Find the paths attached to each iniitiator
        // SCSIProtocolEndpoint.  Each path includes a REF to
        // a target SCSIProtocolEndpoint and to a logical unit.
     $Paths[] = References(
        $Initiators->[#i],// ObjectName
        “CIM_SCSIInitiatorTargetLogicalUnitPath”, // ResultClass
        “Initiator”,            // Role
        false,       // IncludeQualifiers
        false,       // IncludeClassOrigin
        {“LogicalUnitNumber”})  // PropertyList
     // All members of Paths[] have the same Initiator REF.
        // Sort the paths so that all members with identical
        // Target REFs are consecutive.
        $SortedPaths->[] = <Paths[] sorted by Target property>
        // Step 3. Find all the logical units attached to an
        // initiator/target pair and verify that each has
        // a unique logical unit number.
        #l = 0;         // the index of LU numbers to test
        $CurrentTarget = <initialize to null>
        for  (#p in $SortedPaths->[]) {
                // Each time a new target REF is discovered, save it
                // in $CurrentTarget and empty the list if LU numbers.
                if ($CurrentTarget != $SortedPaths[#p]->Target) {
                        $CurrentTarget = $SortedPaths[#p]->Target
                        #LUNumbers[] = {};      // empty the list
                }
                if contains($SortedPath->LogicalUnitNumber, #LUNumbers[]) {
                        <ERROR: logical unit number already in use>
                } else {
                        LUNumbers[#l++] = $SortedPath->LogicalUnitNumber
                }
                // Other interesting bits of info available
                // $SortedPaths->State
                // $SortedPaths->LogicalUnit is a REF to the
                //  LogicalDevice subclass (LogicalDisk, TapeDrive)
                //  where Name is a logical unit correlatable ID
                // $CurrentTarget is a ref to the target
                //  SCSIProtocolController where ConnectionType
                //  is the transport and Name is the transport-
                //  specific correlatable ID (e.g. PortWWN)
                // $Initiators->[#i] is a ref to the initiator
                //  SCSIProtocolController
        }
}
```

8.2.7.5.6.2 Force Failover or change Load Balancing on a volume

```
//
// DESCRIPTION:
// Set the desired path for a multipath disk volume
//
// Preconditions:
//  $Host - Reference to the hosting system
//  $Path  - ref to SCSIInitiatorTargetLogicalUnitPath instance - desired path
//
/  Notes:
// If the volume is asymmetric, failover applies.  If symmetric, then
// we use a driver override to set the path.
//
// $Vol  = LU REF from $Path
$Vol = $Path->LogicalUnit
// Get SCSIPathConfigurationService instances associated
// to $Vol via ServiceAvailableToElement- ERROR if not exactly 1
$Services->[] = AssociatorNames($Vol,  // this is a ref
     “CIM_ServiceAvailableToElement”,
     “CIM_SCSIPathConfigurationService”,
     null,null)
if ($Services == null || $Services->[].size != 1) {
    <ERROR: must not be more than 1
      CIM_SCSIPathConfigurationService
         associated with an LogicalDevice/volume>
}

// Get SCSIMultipathConfigurationCapabilities instances
// associated to $Service - Error if not exactly one
$Capabilities->[] = AssociatorNames($Services->[0],// ObjectName
     “CIM_ElementCapabilities”,// AssocClass
     “CIM_SCSIMultipathConfigurationCapabilities”,// ResultClass
     “ManagedElement”,   // Role
     “Capabilities”)    // ResultRole
if ($Capabilities == null || $Capabilities[].length != 1) {
    <ERROR: must be 1 CIM_SCSIMultipathConfigurationCapabilities instance
        associated with each SCSIPathConfigurationService>
}

// Look at CIM_SCSIMultipathSettings.Asymmetric to determine
// whether the Volume is Asymmetric MP.  If no SCSIMultipathSettings
// is associated to the volume, or if Assymmetric property is
// not-present/null, then assume Symmetric
$SettingDatas-[] = AssociatorNames($Vol,
         “CIM_ElementSettingData”,
      “CIM_SCSIMultipathSettings”,
      “null,null)
If ($SettingDatas == null || $SettingDatas[].length != 1 ||
     $SettingDatas->[0].Asymmetric == null ||
     $SettingDatas->[0].Asymmetric == false) {
    // A Symmetric MP volume has multiple, active paths.
    // Use SetOverridePath to make just one path active
    if ($Capabilities->[0].CanOverridePaths == false) {
      <EXIT: Instrumentation does not support OverridePaths method>
    }
    // set up and invoke the method
    %InArguments[“Path”]=$Path
    #MethodReturn = InvokeMethod(
     $Services->[0],
     “SetOverridePath”,
     %InArguments,
     %OutArguments)
    if(#MethodReturn != 0) {
        <ERROR! SetOverridePath method Failed >
    }
} else {
    // The Volume has Assymmetric MP access
    if (Capabilities->[0].CanSetTPGAccess == false) {
      <EXIT: Instrumentation does not support SetTPGAccess method>
    }
    // Find the TargetPortGroups containing $Vol
    $TPGs->[] = AssociatorNames($Vol,
            “CIM_ConcreteDependency”,
            “CIM_SCSITargetPortGroup”,
            “Dependent”, “Antecedent”)
    // Some of these TPGs may not include the Target Port in $Path,
    // locate one the does.
    #foundTPG = false
    for i in $TPGs->[] {
        $TargetPorts->[] = AssociatorNames($TPGs->[#i],
                          “CIM_MemberOfCollection”,
                 “CIM_LogicalPort”,
                 “Collection”,”Member”)
        if contains($Path->Target, $TargetPorts) {
            $TheTPG = $TPGs->[#i]
         #foundTPG = false
         break
        }
    }
    %LogicalUnit[“LogicalUnit”] = $Vol
    %InArguments[“TargetPortGroups”] = {$TheTPG}
    %InArguments[“AccessStates”] = {“6”} // Active
    #MethodReturn = InvokeMethod(
     $Services->[0],
     “SetTPGAccess”,
     %InArguments,
     %OutArguments)
    if(#MethodReturn != 0) {
        <ERROR! SetSetTPGAccess method Failed >
    }
    // To be completely accurate, we should include SetOverridePath
    // method call here; in theory a TPG can support multiple ports.
    // But in practice, Asymmetric arrays have one port per TPG.
}
```

8.2.7.5.6.3 Change a LogicalDisk’s Load Balancing Algorithm

```
//
// DESCRIPTION:
// Set the load balance algorithm for a multipath disk volume
//
// Preconditions:
//  $Host - Reference to the hosting system
//  $Vol - Reference to the volume
//
/  Notes:
// The currentload balance type could be a driver-wide default (from
// SCSIMultipathConfigurationCapabilities), or a per-LU value from
// SCSIMultipathSettings associated with $Vol.
// Once we get the current value, we search the list of supported values for
// a different supported value, then use it to call SetLoadBalanceAlgorithm
//
// get SCSIPathConfigurationService instances associated to $Vol
// via ServiceAvailableToElement- ERROR if not exactly 1
$Services->[] = AssociatorNames($Vol,  // this is a ref
     “CIM_ServiceAvailableToElement”,
     “CIM_SCSIPathConfigurationService”,
     null,null)
if ($Services == null || $Services->[].size != 1) {
    <ERROR: must not be more than 1
      CIM_SCSIPathConfigurationService
         associated with an LogicalDevice/volume>
}

// 3. Set $Capabilities to the instance SCSIMultipathConfigurationCapabilities
// associated to $Service - Error if not exactly one
$Capabilities->[] = AssociatorNames($Services->[0],// ObjectName
     “CIM_ElementCapabilities”,// AssocClass
```

```
      “CIM_SCSIMultipathConfigurationCapabilities”,// ResultClass
      “ManagedElement”,   // Role
      “Capabilities”)    // ResultRole
if ($Capabilities == null || $Capabilities[].length != 1) {
    <ERROR: must be 1 CIM_SCSIMultipathConfigurationCapabilities instance
        associated with each SCSIPathConfigurationService>
}
// the next two tests are not required, but will help diagnostics
if $Capabilities->[0].OnlySupportsSpecifiedProducts == true {
    <EXIT: the multipath instrumentation only supports 1 devices-specific
       load balance algorithm which cannot be changed>
}
if sizeof($Capabilities->[0].SupportedLoadBalanceTypes) == 1 {
    <EXIT: the multipath instrumentation only supports 1
       load balance algorithm which cannot be changed>
}

// Get the CIM_SCSIMultipathSettings instance associated with $Vol
$SettingDatas->[] = AssociatorNames($Vol,
         “CIM_ElementSettingData”,
      “CIM_SCSIMultipathSettings”,
      “null,null)
if ($SettingDatas != null || $SettingDatas[].length > 1 ) {
    <ERROR: must be 0 or 1 CIM_SCSIMultipathSettings instance
        associated with each SCSI logical unit>
}

// Determine the current load balance type
// The default from Capabilities applies unless overridden
#MyLoadBalanceType = $Capabilities->[0].DefaultLoadBalanceType
// If SettingData is associated to $Vol, it may override load balance
if (($SettingDatas != null || $SettingDatas[].length == 1 )
          // SettingData load balance 7 means use Capabilities default
       && $SettingDatas->[0].CurrentLoadBalanceType != “7”) {
    // Override with value from settings
    #MyLoadBalanceType = $SettingDatas->[0].CurrentLoadBalanceType
    }

#newTypeFound = false
for i in $Capabilities->[0].SupportedLoadBalanceTypes {
  if $Capabilities->[0].SupportedLoadBalanceTypes[#i] != “0” // Unknown
     && $Capabilities->[0].SupportedLoadBalanceTypes[#i] != #MyLoadBalanceType {
     // We found a supported locad balance type other than the
     // current one.  We can live with “No Load Balance” (2),
     // but just for kicks, try to find another one
     #newTypeFound = true
     %InArguments[“LoadBalanceAlgorithm” =
         $Capabilities->[0].SupportedLoadBalanceTypes[#i]
     if $Capabilities->[0].SupportedLoadBalanceTypes[#i] == “1” { // Other
         %InArguments[“OtherLoadBalanceAlgorithmDescription”] =
             $Capabilities->[0].OtherSupportedLoadBalanceTypes[#i]
     }
     if $Capabilities->[0].SupportedLoadBalanceTypes[#i] != “2” { // no LB
        break
     }
   }
}
if #newTypeFound == false {
    <EXIT: no supported load balance types found other than the current type>
}

// invoke the SetLoadBalanceAlgorithm method
%InArguments[“LogicalDevice”] = $Vol
#MethodReturn = InvokeMethod(
    $Services->[0],
    “SetLoadBalanceAlgorithm”,
    %InArguments,
    %OutArguments)
if(#MethodReturn != 0) {
    <ERROR! SetLoadBalanceAlgorithm method Failed >
}
```

8.2.7.5.7 Registered Name and Version

Multipath Management version 1.1.0

8.2.7.5.8 CIM Server Requirements

Table 822: CIM Server Requirements for Multipath Management

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.7.5.9 CIM Elements

Table 823: CIM Elements for Multipath Management

Element Name	Description
Mandatory Classes	
CIM_SCSIInitiatorTargetLogicalUnitPath (8.2.7.5.9.1)	Associates initiator and target SCSIProtocolEndpoints to a SCSI logical unit (LogicalDevice)
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SCSIInitiatorTargetLogicalUnitPath	Path creation
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SCSIInitiatorTargetLogicalUnitPath	Path deletion
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_SCSIInitiatorTargetLogicalUnitPath AND SourceInstance.State <> PreviousInstance.State	Path State change
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_SCSIInitiatorTargetLogicalUnitPath AND SourceInstance.AdministrativeWeight <> PreviousInstance.AdministrativeWeight	Path AdminsitrativeWeight change
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_SCSIInitiatorTargetLogicalUnitPath AND SourceInstance.AdministrativeOverride<> PreviousInstance.AdministrativeOverride	Path AdministrativeOverride change

8.2.7.5.9.1 CIM_SCSIInitiatorTargetLogicalUnitPath

Associates initiator and target SCSIProtocolEndpoints to a SCSI logical unit (LogicalDevice)
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 824: SMI Referenced Properties/Methods for CIM_SCSIInitiatorTargetLogicalUnitPath

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
LogicalUnit		CIM_LogicalDevice	A reference to a LogicalDevice
Initiator		CIM_SCSIProtocolEndpoint	A reference to the initiator CIM_SCSIProtocolEndpoint
Target		CIM_SCSIProtocolEndpoint	A reference to the target CIM_SCSIProtocolEndpoint
AdministrativeWeight	M	uint32	
State		uint32	
AdministrativeOverride		uint16	
Optional Properties/Methods			
OSDeviceName		string	

8.2.7.5.10 Related Standards

Table 825: Related Standards for Multipath Management

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF
SPC-3	21a November 2004	T10 (to be published as ISO/IEC 14776-313)

EXPERIMENTAL

8.2.8 Storage Profiles

8.2.8.1 Array Profile

8.2.8.1.1 Description

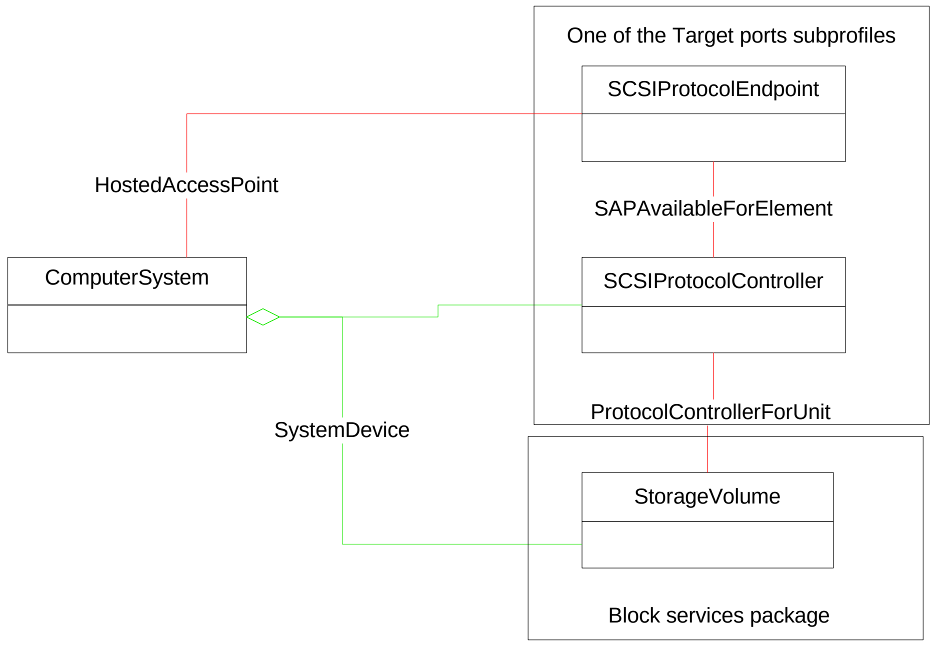
The Array model profile describes external RAID arrays and disk storage systems. The key classes are:

- Computer Systems that represent the array as a whole;
- Storage Volumes that are equivalent to SCSI Logical Units visible to consumers;
- StoragePools that are the allocatable/available space on the array;
- A SCSI transport (e.g., Fibre Channel or iSCSI) through which the LUNs are made available.

The basic array profile provides a high level read-only ‘view’ of an array. The Block Services Package (8.2.8.10) includes the basic description of how storage is managed. This profile also includes the mandatory Physical Package Package (8.2.1.10)) that describes the physical layout of the array and includes product identification information

The various subprofiles indicated in Figure 120: "Array Profile Instance Diagram" cover other areas of functionality. Refer to 8.2.8.1.7, "Registered Name and Version" for more information on these optional extensions.

Figure 120: Array Profile Instance Diagram



8.2.8.1.2 Health and Fault Management

Not defined in this standard.

8.2.8.1.3 Cascading Considerations

Not defined in this standard.

8.2.8.1.4 Supported Subprofiles and Packages

Table 826: Supported Subprofiles for Array

Registered Subprofile Names	Mandatory	Version
Access Points	No	1.1.0
Block Server Performance	No	1.1.0
Cluster	No	1.0.2
Extra Capacity Set	No	1.0.2
Disk Drive	No	1.0.2
Disk Drive Lite	No	1.1.0
Extent Mapping	No	1.0.2
Extent Composition	No	1.1.0
Location	No	1.1.0
Software	No	1.1.0
Copy Services	No	1.1.0
Job Control	No	1.1.0
Pool Manipulation Capabilities and Settings	No	1.0.2
LUN Creation	No	1.0.2
Device Credentials	No	1.1.0
LUN Mapping and Masking	No	1.0.2
Masking and Mapping	No	1.1.0
SPI Target Ports	No	1.1.0
FC Target Ports	No	1.1.0
iSCSI Target Ports	No	1.1.0
Backend Ports	No	1.0.2
Disk Sparing	No	1.1.0
FC Initiator Ports	No	1.1.0
SPI Initiator Ports	No	1.1.0

Table 827: Supported Packages for Array

Registered Package Names	Version
Block Services	1.1.0
Physical Package	1.1.0
Health	1.1.0

8.2.8.1.5 Methods of the Profile

None.

8.2.8.1.6 Client Considerations and Recipes

None.

8.2.8.1.7 Registered Name and Version

Array version 1.1.0

8.2.8.1.8 CIM Server Requirements

Table 828: CIM Server Requirements for Array

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.1.9 CIM Elements

Table 829: CIM Elements for Array

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.8.1.9.1)	'Top level' system that represents the whole array.
CIM_HostedAccessPoint (8.2.8.1.9.2)	This association that links ProtocolEndpoints to the hosting computer system.
CIM_LogicalPort (8.2.8.1.9.3)	Target (front end) ports for the array. Most requirements are defined in the referenced target port subprofiles. In this profile, this class represents a requirement for some target port - regardless of the transport.
CIM_SCSIProtocolController (8.2.8.1.9.5)	
CIM_SystemDevice (8.2.8.1.9.6)	This association links all **LogicalDevices** to the scoping system.
Optional Classes	
CIM_SCSIArbitraryLogicalUnit (8.2.8.1.9.4)	A SCSI Logical Unit that exists only for management of the array.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Addition of a new array instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Deletion of an array instance

8.2.8.1.9.1 CIM_ComputerSystem

'Top level' system that represents the whole array.
Created By : External
Modified By : External
Deleted By : External
Standard Names: The Name and NameFormat properties shall follow the requirements in 6.2.4.5.4

Class Mandatory: true

Table 830: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	Unique identifier for the array. Eg IP address
ElementName		string	User friendly name
OtherIdentifyingInfo	C	string[]	
IdentifyingDescriptions	C	string[]	
OperationalStatus		uint16[]	Overall status of the array
NameFormat		string	Format for Name property.
Dedicated		uint16[]	Indicates that this computer system is dedicated to operation as a storage array
Optional Properties/Methods			
PrimaryOwnerContact	M	string	Contact a details for owner
PrimaryOwnerName	M	string	Owner of the array

8.2.8.1.9.2 CIM_HostedAccessPoint

This association that links ProtocolEndpoints to the hosting computer system.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 831: SMI Referenced Properties/Methods for CIM_HostedAccessPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	
Antecedent		CIM_System	

8.2.8.1.9.3 CIM_LogicalPort

Target (front end) ports for the array. Most requirements are defined in the referenced target port subprofiles. In this profile, this class represents a requirement for some target port - regardless of the transport.
Class Mandatory: true

Table 832: SMI Referenced Properties/Methods for CIM_LogicalPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UsageRestriction		uint16	Shall be 2 to indicate this is a front end port only or 4 to indicate usage is not restricted.

8.2.8.1.9.4 CIM_SCSIArbitraryLogicalUnit

A SCSI Logical Unit that exists only for management of the array.
Class Mandatory: false

Table 833: SMI Referenced Properties/Methods for CIM_SCSIArbitraryLogicalUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifer
ElementName		string	User-friendly name
Name		string	
OperationalStatus		uint16[]	

8.2.8.1.9.5 CIM_SCSIProtocolController

Class Mandatory: true

Table 834: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	

8.2.8.1.9.6 CIM_SystemDevice

This association links all **LogicalDevices** to the scoping system.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 835: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.8.1.10 Related Standards

Table 836: Related Standards for Array

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.2 Storage Virtualizer Profile

8.2.8.2.1 Description

These products act like Raid arrays but do not include any local storage. A Storage Virtualizer system uses storage provided by array controllers 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 121:, "Storage Virtualizer Package Diagram" extend this description and also enable configuration. Refer to 8.2.8.2.4, "Supported Subprofiles and Packages" for more information on these optional extensions. This profile also includes the mandatory 8.2.1.10, "Physical Package Package" 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. 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 *CIM Schema* for information on class inheritance information and full information on the properties and methods used.

Instance Diagrams

Figure 121: "Storage Virtualizer Package Diagram" illustrates the relationship between the packages related to the Storage Virtualizer Profile.

Figure 121: Storage Virtualizer Package Diagram

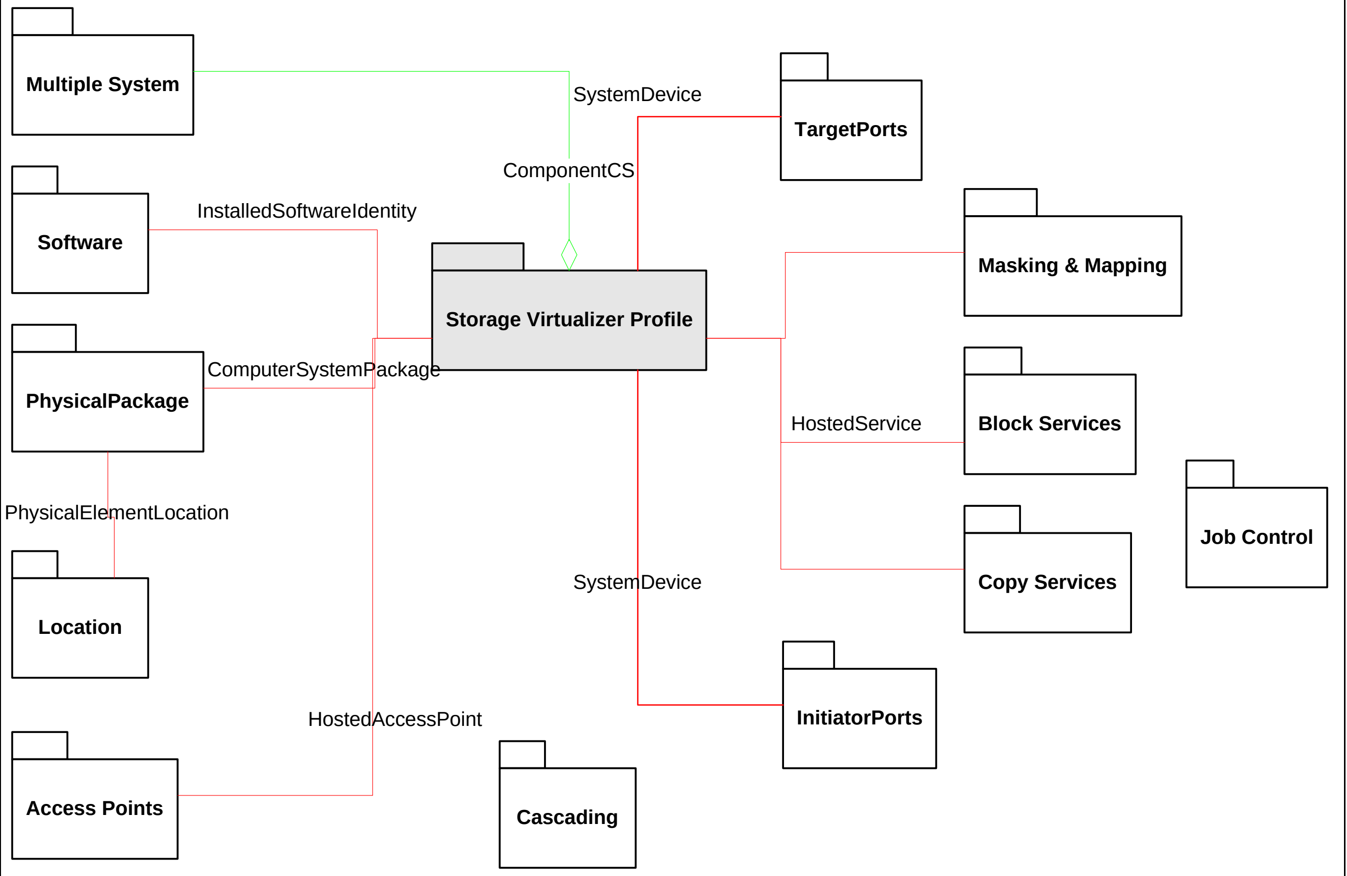
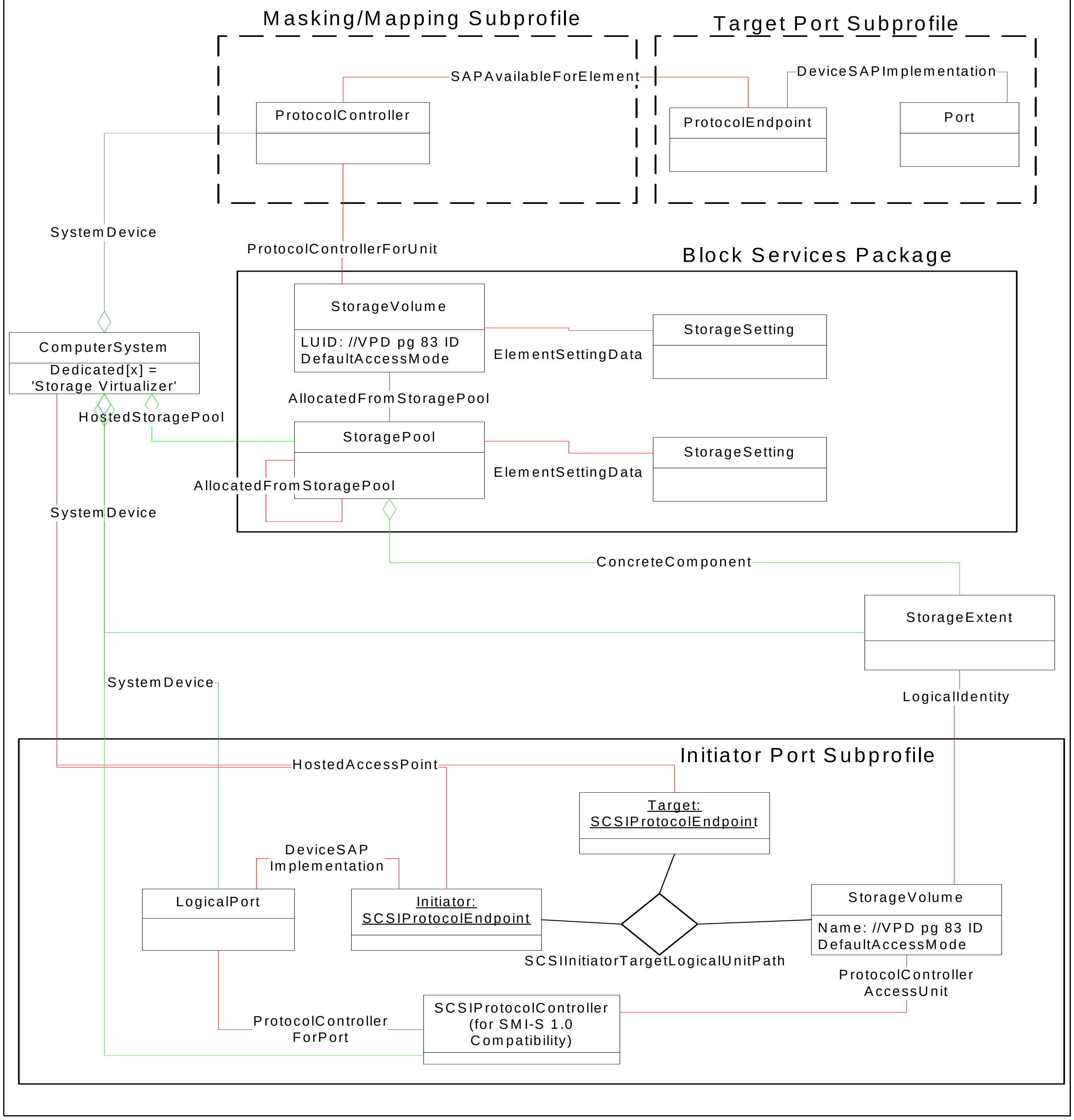


Figure 122: "Storage Virtualizer System Instance" illustrates a typical instance diagram.

Figure 122: Storage Virtualizer System Instance



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 and are connected to the imported StorageVolume from the array using LogicalIdentity. The SCSIProtocolController with the ProtocolControllerAccessesUnit association to the StorageVolume are provided for clients convenience and compatibility with IS24775-2006, *Storage Managemen*t *(*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.

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.

Device Management Access

Most devices now have a web GUI to allow device specific configuration. This is modeled using the common subprofile "Access Point".

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.

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.

Ports

An implementation of the Storage Virtualizer shall implement at least one Target Ports Subprofile and at least one Initiator Ports Subprofile. 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.

8.2.8.2.2 Health and Fault Management

Not defined in this standard.

8.2.8.2.3 Cascading Considerations

Not defined in this standard.

8.2.8.2.4 Supported Subprofiles and Packages

Table 837, “Supported Subprofiles for Storage Virtualizer” lists the supported subprofiles for this Profile.

Table 837: Supported Subprofiles for Storage Virtualizer

Registered Subprofile Names	Mandatory	Version
Access Points	No	1.1.0
Copy Services	No	1.1.0
Job Control	No	1.1.0
Location	No	1.1.0
Masking and Mapping	No	1.1.0
Software	No	1.1.0
Multiple Computer System	No	1.1.0
FC Initiator Ports	No	1.1.0
iSCSI Target Ports	No	1.1.0
FC Target Ports	No	1.1.0
iSCSI Initiator Ports	No	1.1.0
Extent Composition	No	1.1.0
Cascading	No	1.1.0

Table 838: Supported Packages for Storage Virtualizer

Registered Package Names	Version
Block Services	1.1.0
Physical Package	1.1.0

8.2.8.2.5 Methods of the Profile

None.

8.2.8.2.6 Client Considerations and Recipes

None.

8.2.8.2.7 Registered Name and Version

Storage Virtualizer version 1.1.0

8.2.8.2.8 CIM Server Requirements

Table 839: CIM Server Requirements for Storage Virtualizer

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.2.9 CIM Elements

Table 840: CIM Elements for Storage Virtualizer

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.8.2.9.1)	'Top-level' system that represents the whole virtualizer.
CIM_ConcreteComponent (8.2.8.2.9.2)	Used to associate StorageExtents that are playing the Pool Component role to a Primordial StoragePool.
CIM_LogicalIdentity (8.2.8.2.9.3)	Used to associate StorageExtents of a Primordial StoragePool to the imported StorageVolumes (of the InitiatorPort Subprofile.
CIM_LogicalPort (8.2.8.2.9.4)	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_StorageExtent (8.2.8.2.9.6)	Used to represent the storage imported from external arrays and used as ConcreteComponents of Primordial StoragePools.
Optional Classes	
CIM_LogicalPort (8.2.8.2.9.5)	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.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Creation of a ComputerSystem instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance CIM_ComputerSystem	Deletion of a ComputerSystem instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.CIM_StorageVolume::OperationalStatus <> PreviousInstance.CIM_StorageVolume::OperationalStatus	CQL - Modification of OperationalStatus of a Storage Volume instance

Table 840: CIM Elements for Storage Virtualizer

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalPort AND SourceInstance.CIM_LogicalPort::OperationalStatus <> PreviousInstance.CIM_LogicalPort::OperationalStatus	CQL - Modification of OperationalStatus of a Logical (FC or Ethernet) port instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus <> PreviousInstance.CIM_ComputerSystem::OperationalStatus	CQL - Modification of OperationalStatus of a ComputerSystem instance
Optional Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	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_FCPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Modification of OperationalStatus of a FC port instance, provided for backward compatibility with In-band Virtualization.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Modification of OperationalStatus of a ComputerSystem instance, provided for backward compatibility with In-band Virtualization.

8.2.8.2.9.1 CIM_ComputerSystem

'Top-level' system that represents the whole virtualizer.
Created By : External
Standard Names: The Name and NameFormat properties shall follow the requirements in 6.2.4.5.4
Class Mandatory: true

Table 841: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name	C	string	Unique identifier for the array, e.g., IP address or a FC WWN.
ElementName		string	User-friendly name
OperationalStatus		uint16[]	Overall status of the array
NameFormat		string	Format for Name property.
Dedicated		uint16[]	Indicates that this computer system is dedicated to operation as a storage virtualizer
Optional Properties/Methods			
PrimaryOwnerContact	M	string	Contact a details for owner
PrimaryOwnerName	M	string	Owner of the array

8.2.8.2.9.2 CIM_ConcreteComponent

Used to associate StorageExtents that are playing the Pool Component role to a Primordial StoragePool.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 842: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ManagedElement	A Primordial StoragePool
PartComponent		CIM_ManagedElement	The StorageExtent

8.2.8.2.9.3 CIM_LogicalIdentity

Used to associate StorageExtents of a Primordial StoragePool to the imported StorageVolumes (of the InitiatorPort Subprofile.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 843: SMI Referenced Properties/Methods for CIM_LogicalIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SameElement		CIM_ManagedElement	The imported StorageVolume
SystemElement		CIM_ManagedElement	The StorageExtent

8.2.8.2.9.4 CIM_LogicalPort

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.
Class Mandatory: true

Table 844: SMI Referenced Properties/Methods for CIM_LogicalPort (Target Port)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UsageRestriction		uint16	Shall be 3 to indicate this is a back end port only or 4 to indicate usage is not restricted.

8.2.8.2.9.5 CIM_LogicalPort

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.

Class Mandatory: false

Table 845: SMI Referenced Properties/Methods for CIM_LogicalPort (Initiator Port)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
UsageRestriction		uint16	Shall be 2 to indicate this is a front end port only or 4 to indicate usage is not restricted.

8.2.8.2.9.6 CIM_StorageExtent

Used to represent the storage imported from external arrays and used as ConcreteComponents of Primordial StoragePools.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 846: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
BlockSize		uint64	
NumberOfBlocks		uint64	
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	

8.2.8.2.10 Related Standards

Table 847: Related Standards for Storage Virtualizer

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.3 Volume Management Profile

8.2.8.3.1 Description

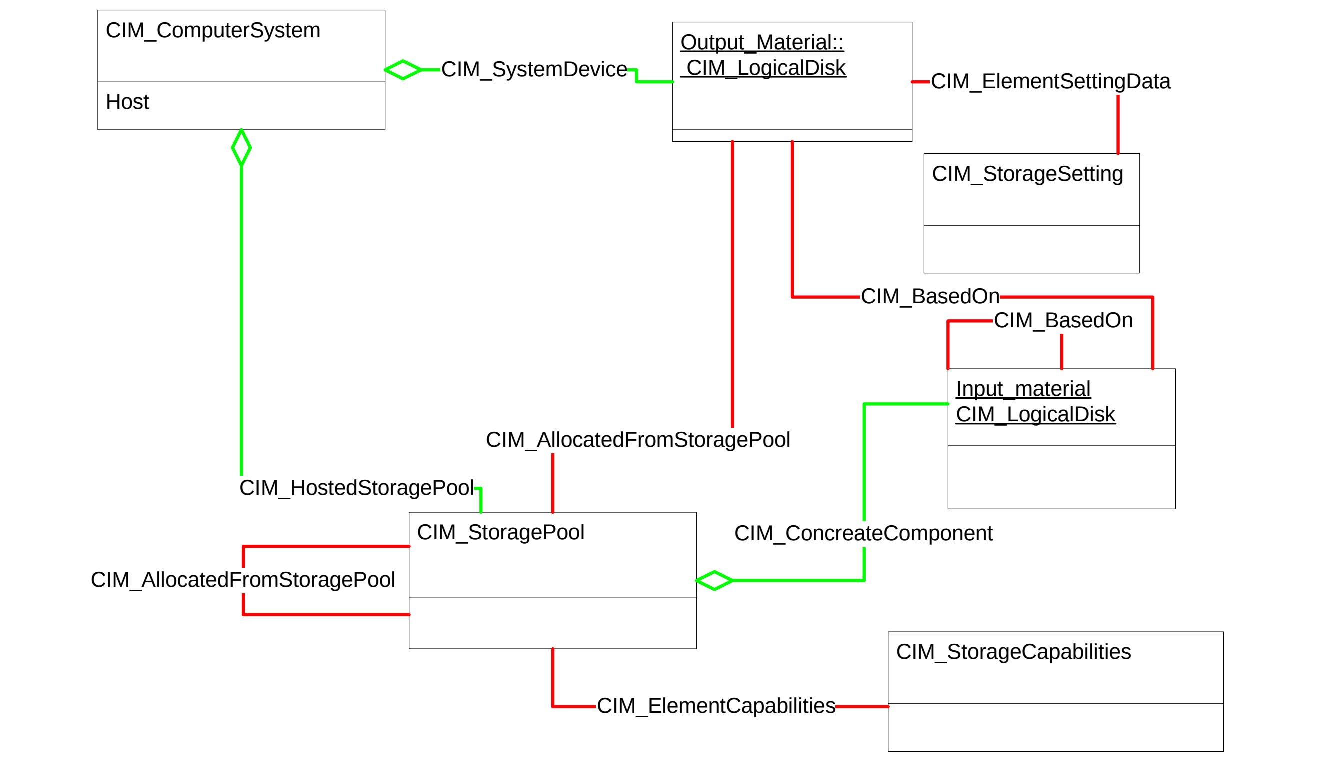
The host Volume Management (VM) profile addresses block storage virtualization and presents virtual block devices to clients. The model represents virtualization for host volume management where LogicalDisks are exported.

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.

The Volume Management profile uses existing classes from the Array profile and Block Services subprofile, and optionally uses the Host Discovered Resources subprofile to bind with the disks in the host operating system.

Instance Diagram

Figure 123: Volume Management Instance Diagram



Input Class of the Volume Manager

The host operating system provides a unique name for each disk via a special file name. Typically, these are device file names: drive letters on Windows systems or /dev/dsk/device1 on UNIX systems. A LogicalDisk can be based on a disk partition or created by the operating system to represent a discovered volume and would have an 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.

<u>Export Class of the Volume Manager</u>

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.

8.2.8.3.1.1 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.

8.2.8.3.1.2 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 8.2.8.10.1, "Description" for more details on methods for creating pools and logical volumes.

8.2.8.3.1.3 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 8.2.8.10, "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.

8.2.8.3.1.4 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.

8.2.8.3.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.3.3 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.

8.2.8.3.4 Supported Subprofiles and Packages

Table 848: Supported Subprofiles for Volume Management

Registered Subprofile Names	Mandatory	Version
Access Points	No	1.1.0
Extent Composition	No	1.1.0
Location	No	1.1.0
Software	No	1.1.0
Copy Services	No	1.1.0
Disk Sparing	No	1.1.0
Multi System	No	1.1.0
Job Control	No	1.1.0
Cascading	No	1.1.0
Block Storage Resource Ownership	No	1.1.0
Block Server Performance	No	1.1.0

Table 849: Supported Packages for Volume Management

Registered Package Names	Version
Block Services	1.1.0
Health	1.1.0

8.2.8.3.5 Methods of the Profile

None

8.2.8.3.6 Client Considerations and Recipes

Use the Block Services subprofile to create and modify volumes.

See recipes for creating volumes in 8.2.8.10, "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.

8.2.8.3.6.1 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.

8.2.8.3.7 Registered Name and Version

Volume Management version 1.1.0

8.2.8.3.8 CIM Server Requirements

Table 850: CIM Server Requirements for Volume Management

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.3.9 CIM Elements

Table 851: CIM Elements for Volume Management

Element Name	Description
Mandatory Classes	
CIM_AllocatedFromStoragePool (8.2.8.3.9.1)	
CIM_ComputerSystem (8.2.8.3.9.2)	
CIM_ElementCapabilities (8.2.8.3.9.3)	
CIM_ElementSettingData (8.2.8.3.9.4)	
CIM_HostedStoragePool (8.2.8.3.9.5)	
CIM_LogicalDisk (8.2.8.3.9.6)	
CIM_StorageCapabilities (8.2.8.3.9.7)	
CIM_StoragePool (8.2.8.3.9.8)	Logical Disks are allocated from 'concrete' pools.
CIM_StoragePool (8.2.8.3.9.9)	At least one primordial pool must exist for a host. This is the 'unallocated storage' of the host, and contains unused disks.
CIM_StorageSetting (8.2.8.3.9.10)	
CIM_SystemDevice (8.2.8.3.9.11)	
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Addition of a new logical disk instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Deletion of a logical disk instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	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	CQL - Change of status of a Logical Disk

Table 851: CIM Elements for Volume Management

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StoragePool AND SourceInstance.CIM_StoragePool::OperationalStatus <> PreviousInstance.CIM_StoragePool::OperationalStatus	CQL - Change of status of a storage pool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Addition of a storage pool instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Deletion of a storage pool instance

8.2.8.3.9.1 CIM_AllocatedFromStoragePool

Class Mandatory: true

Table 852: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StoragePool	
Dependent		CIM_LogicalElement	
SpaceConsumed		uint64	

8.2.8.3.9.2 CIM_ComputerSystem

Class Mandatory: true

Table 853: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Name		string	Unique identifier for the Host. IP address
ElementName		string	User-friendly name
OperationalStatus		uint16[]	Overall status of the Host
NameFormat		string	Format for Name property.
Dedicated		uint16[]	Indicates that this computer system is not dedicated to volume management.
Optional Properties/Methods			
PrimaryOwnerContact		string	
PrimaryOwnerName		string	

8.2.8.3.9.3 CIM_ElementCapabilities

Class Mandatory: true

Table 854: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Capabilities		CIM_Capabilities	

8.2.8.3.9.4 CIM_ElementSettingData

Created By : Extrinsic(s):
Class Mandatory: true

Table 855: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
SettingData		CIM_SettingData	

8.2.8.3.9.5 CIM_HostedStoragePool

Class Mandatory: true

Table 856: SMI Referenced Properties/Methods for CIM_HostedStoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_StoragePool	

8.2.8.3.9.6 CIM_LogicalDisk

Class Mandatory: true

Table 857: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
ElementName		string	User-friendly name
Name		string	Should be a durable name. As yet any name
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
BlockSize		uint64	
NumberOfBlocks		uint64	

Table 857: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
IsBasedOnUnderlyingRedun-dancy		boolean	
NoSinglePointOfFailure		boolean	
DataRedundancy		uint16	
PackageRedundancy		uint16	
DeltaReservation		uint8	
ConsumableBlocks		uint64	

8.2.8.3.9.7 CIM_StorageCapabilities

Class Mandatory: true

Table 858: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
NoSinglePointOfFailure		boolean	
NoSinglePointOfFailureDefault		boolean	
DataRedundancyMin		uint16	
DataRedundancyMax		uint16	
DataRedundancyDefault		uint16	
PackageRedundancyMin		uint16	
PackageRedundancyMax		uint16	
PackageRedundancyDefault		uint16	
DeltaReservationDefault		uint16	
DeltaReservationMax		uint16	
DeltaReservationMin		uint16	

8.2.8.3.9.8 CIM_StoragePool

Logical Disks are allocated from 'concrete' pools.
Created By : External
Class Mandatory: true

Table 859: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial = False)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
PoolID		string	
TotalManagedSpace		uint64	
RemainingManagedSpace		uint64	

Table 859: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial = False)

Property	Flags	Type	Description & Notes
Primordial		boolean	

8.2.8.3.9.9 CIM_StoragePool

At least one primordial pool must exist for a host. This is the 'unallocated storage' of the host, and contains unused disks.
Class Mandatory: true

Table 860: SMI Referenced Properties/Methods for CIM_StoragePool (Primordial = True)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
PoolID		string	
TotalManagedSpace		uint64	
RemainingManagedSpace		uint64	
Primordial		boolean	

8.2.8.3.9.10 CIM_StorageSetting

Class Mandatory: true

Table 861: SMI Referenced Properties/Methods for CIM_StorageSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque identifier
ElementName		string	User-friendly name, can be used for 'potted' settings for specific RAID lev-els.
DataRedundancyMin		uint16	
DataRedundancyMax		uint16	
DataRedundancyGoal		uint16	
PackageRedundancyMin		uint16	
PackageRedundancyMax		uint16	
PackageRedundancyGoal		uint16	
DeltaReservationGoal		uint8	
DeltaReservationMax		uint8	
DeltaReservationMin		uint8	

8.2.8.3.9.11 CIM_SystemDevice

Created By : External

Class Mandatory: true

Table 862: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.8.3.10 Related Standards

Table 863: Related Standards for Volume Management

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.4 NAS Head Profile

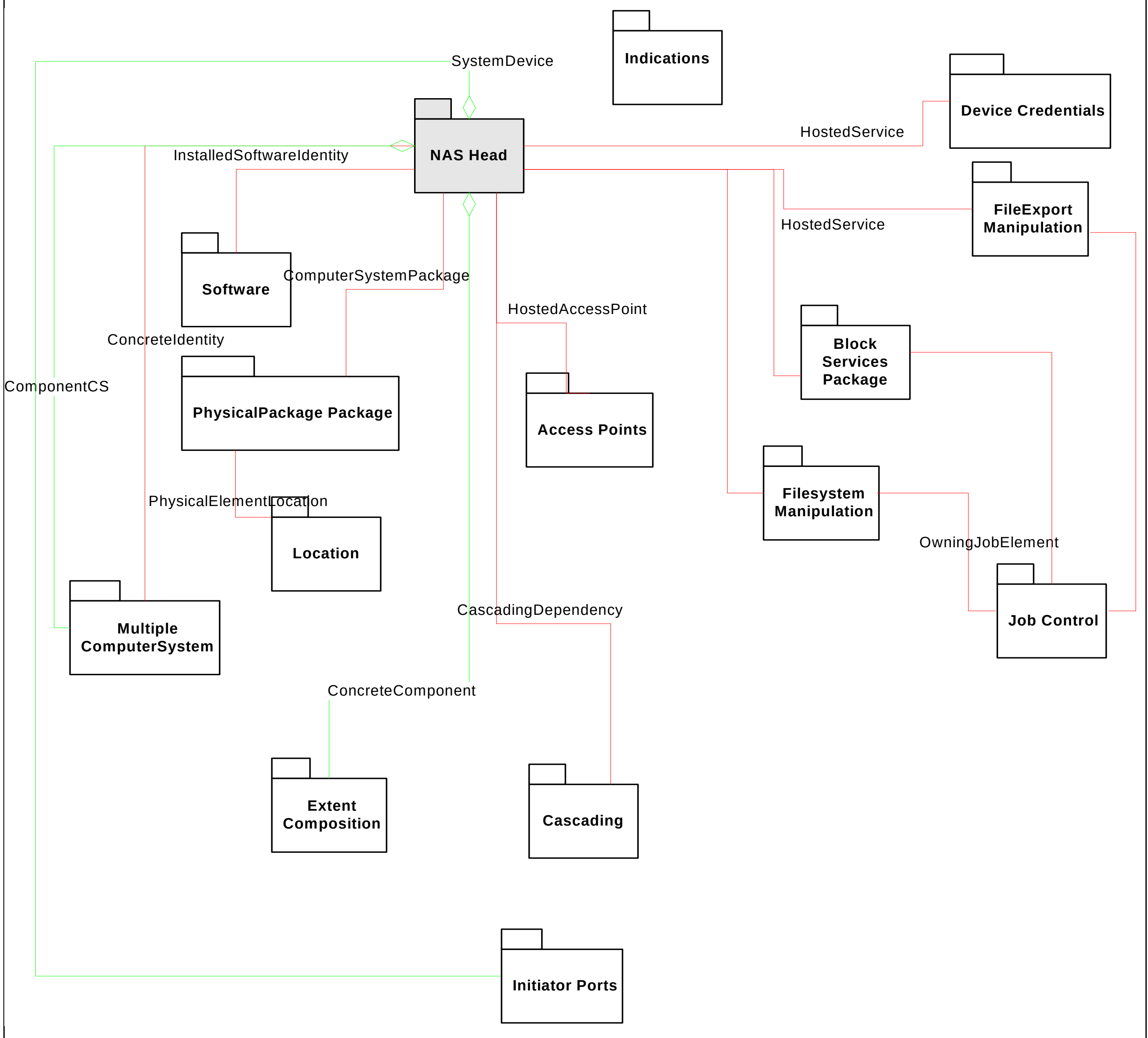
8.2.8.4.1 Description

A NAS Head is a NAS controller that gets its storage from external SAN storage. The storage obtained is visible to external management tools and they may share their storage with other hosts or devices. For example, an NAS Head might go to an array for its storage. The array provides storage to the NAS Head, but may also supply storage to Hosts (or other NAS heads).

This profile defines how to model the NAS head constructs, and how it reflects connections to and storage from the array(s) below it. The actual details of how the array exports storage to the NAS head, is not covered by the NAS head. This would be covered by the array profile.

The NAS Head profile reuses a significant portion of the Storage Virtualizer Profile. This is illustrated in Figure 124: "NAS Head Profiles and Subprofiles".

Figure 124: NAS Head Profiles and Subprofiles



The NAS Head Profile and its subprofiles provide the following capabilities to SMI-S:

- Device Level Configuration
    - The NAS Head Profile defines reporting of physical storage. This includes configuration of storage access below the NAS Head (on the SAN).
    - The NAS Head Profile supports indications on OperationalStatus of the NAS processors.
- Connectivity Level Configuration
    - The NAS Head Profile defines reporting on port connectivity to the NAS Head (and Port connectivity to underlying storage).
    - The NAS Head Profile supports indications on OperationalStatus of the NAS Ports and ProtocolEndpoints.
    - The File Export Manipulation Subprofile defines mechanism for establishing file access through port connectivity to the NAS offering.
- Block Level Configuration
    - The NAS Head Profile defines reporting on logical storage (StoragePools) and LogicalDisks on those pools.

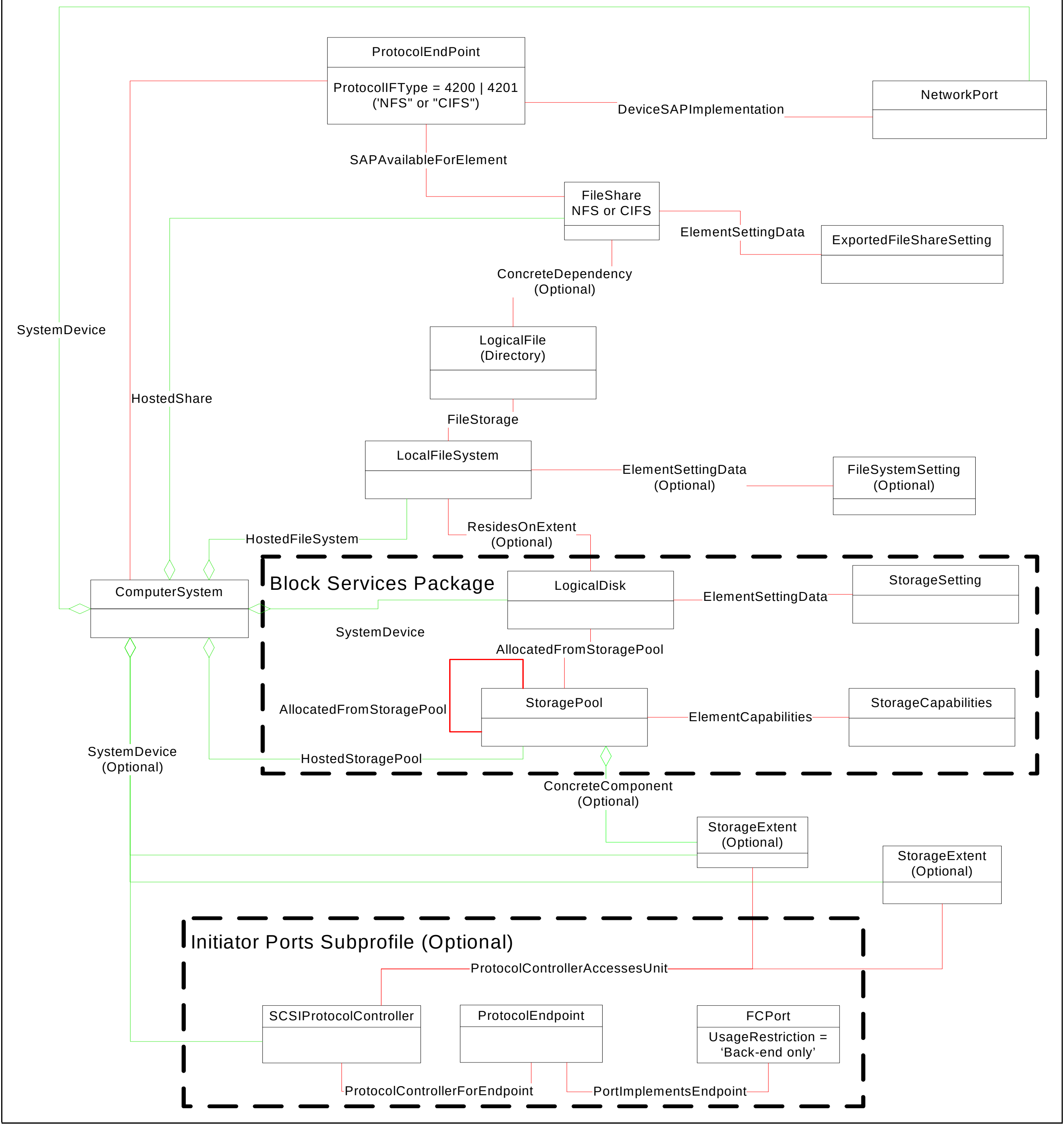
Note: Filesystems are built on the LogicalDisks.

-
    - The NAS Head Profile supports indications on OperationalStatus of the LogicalDisks.
- File/Record Configuration
    - The NAS Head Profile defines reporting on the file systems and file shares that are configured out of the storage of the NAS Head.
    - The NAS Head Profile supports indications on OperationalStatus of the FileSystems and FileShares.
    - The Filesystem Manipulation Subprofile defines the ability to configure file systems out of the storage of the NAS Profile it supports.
    - The File Export Manipulation Subprofile defines mechanism for establishing file shares on local file systems that can then be accessed by remote clients.

8.2.8.4.1.1 Summary Instance Diagram

Figure 125: "NAS Head Instance Diagram" illustrates the mandatory classes for the NAS Head Profile. This figure shows all the classes that are mandatory for the NAS Head Profile. Later diagrams will review specific sections of this diagram.

Figure 125: NAS Head Instance Diagram



The NAS Head Profile closely parallels the Storage Virtualizer Profile in how it models storage. Storage is assigned to StoragePools and LogicalDisks are allocated from those storage pools for the purpose of holding local file systems of the NAS.

As with the Storage Virtualizer Profile, the NAS Head StoragePools have StorageCapabilities associated to the StoragePools via ElementCapabilities. Similarly, LogicalDisks that are allocated from those StoragePools have StorageSettings, which are associated to the LogicalDisk via ElementSettingData. StoragePools are hosted by a ComputerSystem that represents the NAS "top level" system, and the StorageExtents have a SystemDevice association to the "top level" ComputerSystem.

Note: As with Self-Contained NAS, the StoragePools may be hosted by a component ComputerSystem if the Profile has implemented the Multiple Computer System Subprofile.

As with the Storage Virtualizer Profile, the "top level" ComputerSystem of the NAS Head does not (and typically isn't) a real ComputerSystem. It is merely the ManagedElement upon which all aspects of the NAS offering are scoped.

As with Storage Virtualizer Profile, the NAS Head draws its storage from an open SAN. That is, the actual disk storage is addressable independent of the NAS Head. As a result, the NAS head shall model the Initiator ports and the StorageExtents that it acquires from the SAN. The NAS Head supports at least one of the Initiator Ports Subprofiles (the dashed box at the bottom of the diagram) to effect the support for backend ports. The NAS Head includes the Block Services Package to effect the logical storage management (the dashed box just above the Initiator Ports dashed box in the diagram).

Everything above the LogicalDisk is specific to NAS (and does not appear in the Storage Virtualizer Profile). LocalFileSystems are created on the LogicalDisks, LogicialFiles within those LocalFileSystens are shared (FileShare) through ProtocolEndpoints associated with NetworkPorts.

The ConcreteDependency association is optional. It is shown here to illustrate a relationship between a FileShare and some Directory. However, the Directory need not be part of the LocalFileSystem. Similarly, the ResidesOnExtent is optional, but is shown here to illustrate that a LocalFileSystem may map to a LogicalDisk. However, other mappings to storage are also possible.

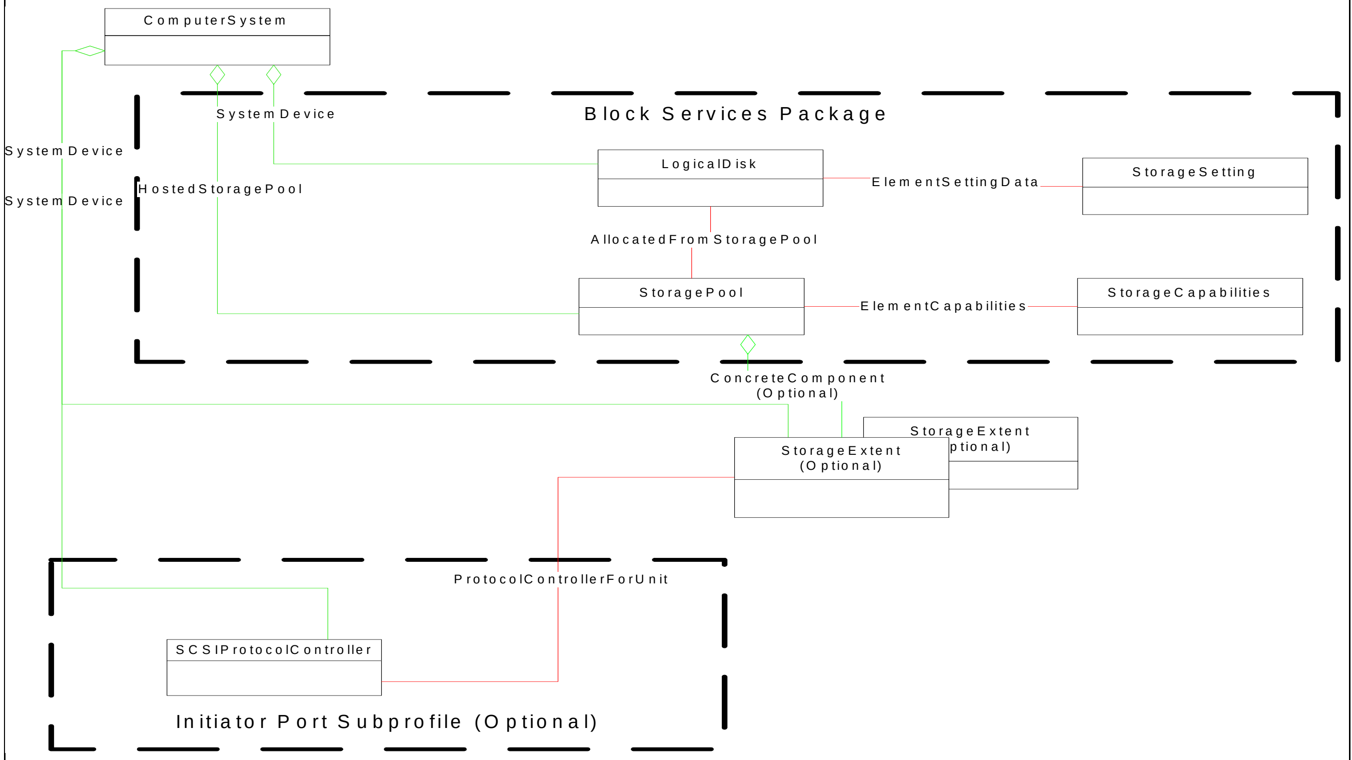
Also note that FileSystemSetting (and the corresponding ElementSettingData) are also optional. They are only shown here to illustrate where they would show up in the model should they be implemented.

In the base NAS Head profile, these are automatically populated based on how the NAS Head is configured. Client modification of the configuration (including configuring storage, creating extents, local file systems and file shares) are functions found in subprofiles of the NAS Head Profile.

8.2.8.4.1.2 NAS Storage Model

Figure 126: "NAS Storage Instance Diagram" illustrates the classes mandatory for modeling of storage for the NAS Head Profile.

Figure 126: NAS Storage Instance Diagram



The NAS Head Profile uses most of the classes and associations used by the Storage Virtualizer Profile (including those in the Block Services Package). In doing this, it leverages many of the subprofiles that are available for Storage Virtualizer Profiles. The classes and associations shown in Figure 126: "NAS Storage Instance Diagram" are the minimum mandatory for read only access in the base profile.

Storage for the NAS shall be modeled as logical storage. That is, StoragePools shall be modeled, including the HostedStoragePool and ElementCapabilities to the StorageCapabilities supported by the StoragePool. In addition, for NAS Heads, which get their storage from a SAN, the StorageExtents that compose the lowest StoragePools may also be modeled with ConcreteComponent associations to the StoragePool to which they belong. However, modeling of StorageExtents is optional for NAS Heads.

In addition, in order for storage to be used it shall be allocated to one or more LogicalDisks. A LogicalDisk shall have an AllocatedFromStoragePool association to the StoragePool from which it is allocated. The LogicalDisk shall have an ElementSettingData association to the settings that were used when the LogicalDisk was created.

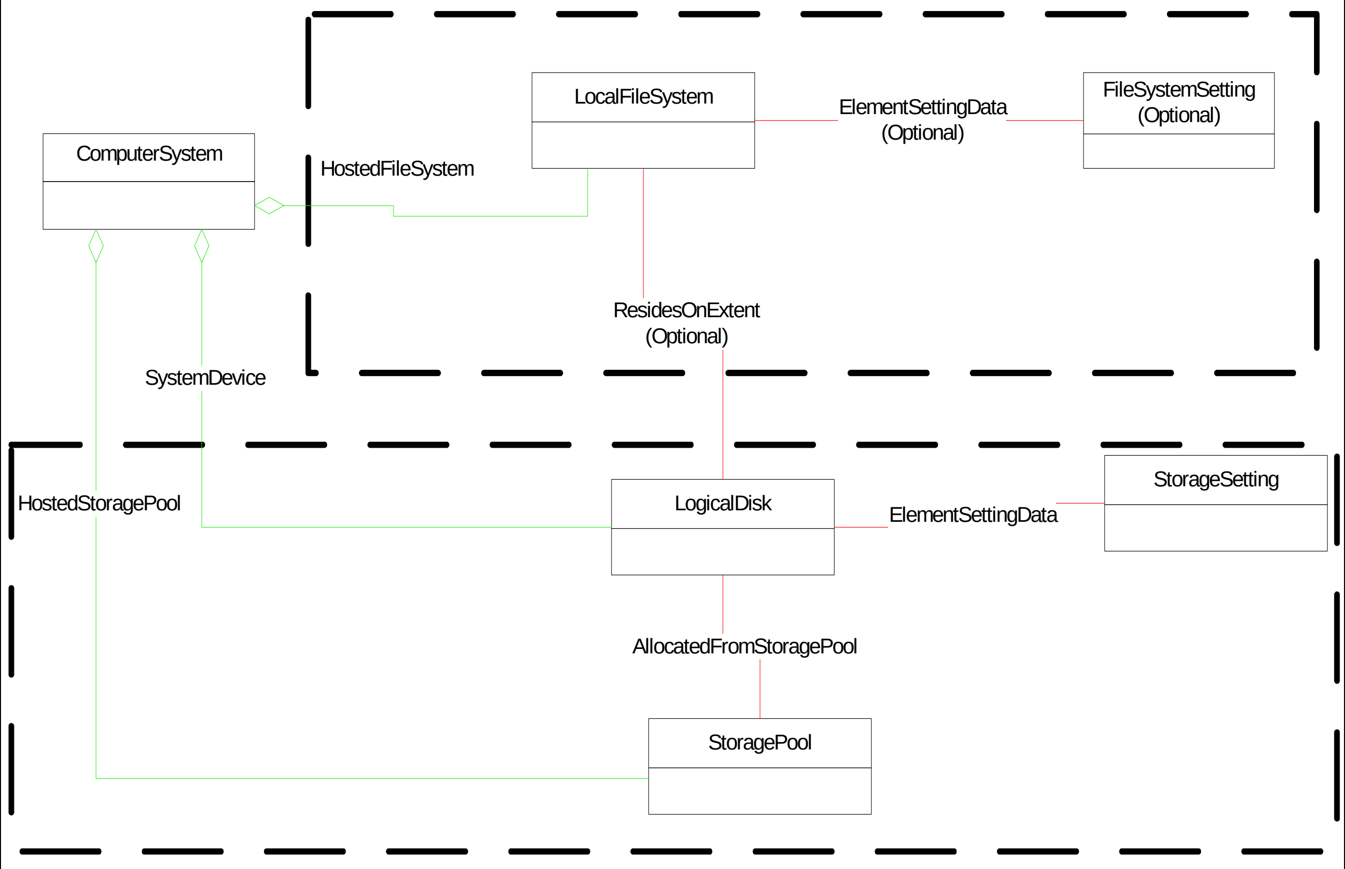
For manipulation of Storage, see 8.2.8.10, "Block Services Package". For NAS Heads, LogicalDisks are the ElementType that is supported for storage allocation functions (e.g., CreateOrModifyElementFromStoragePool and ReturnToStoragePool) and LogicalDisk creation is optional. NAS also supports (optionally) the Pool manipulation functions (e.g., CreateOrModifyStoragePool and DeleteStoragePool) of the Block Services Package.

8.2.8.4.1.3 NAS Filesystem Model

Figure 127: "NAS Filesystem Instance Diagram" illustrates the classes mandatory for the modeling of file systems for the NAS Profiles.

Note: This part of the model is the same for both the NAS Head and the Self-contained NASt

Figure 127: NAS Filesystem Instance Diagram



The NAS Profile builds on the storage with Filesystems which are established on LogicalDisks. In the case of NAS Profiles, one Filesystem is established on one LogicalDisk.

Note: One Filesystem may also span multiple LogicalDisks or a Filesystem is may be allocated directly from a StoragePool, but these methods of storing a FileSystem are not covered by this version of SMI-S.

A Filesystem shall be represented in the model as instance of LocalFileSystem. A LocalFileSystem instance may have exactly one ResidesOnExtent association to one exactly one LogicalDisk. In this case, a client would determine the size (in bytes) of a Filesystem by inspecting the size of the LogicalDisk on which the filesystem resides. FileSystemSize can also be found as a property of LocalFileSystem. For other methods of FileSystem storage, the client should use the FileSystemSize property of the LocalFileSystem.

The FileSystem shall have a HostedFileSystem association to a NAS ComputerSystem. Normally this will be the top level ComputerSystem of the NAS profile. However, if the Multiple Computer System Subprofile is implemented, the HostedFileSystem may be associated to a component ComputerSystem (See t8.2.1.9, "Multiple Computer System Subprofile").

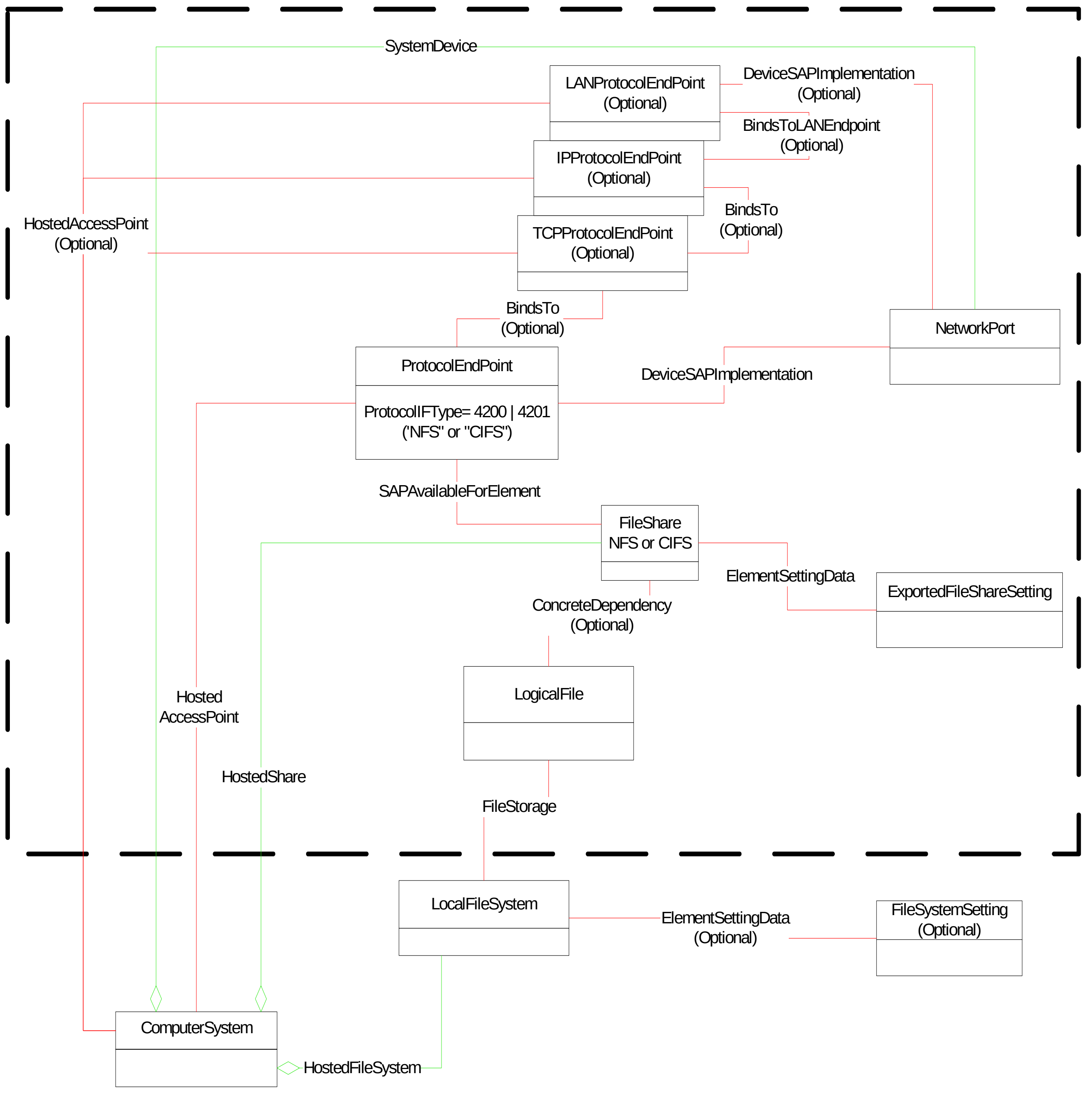
The LocalFileSystem instance may also have an ElementSettingData association to the FileSystemSetting for the Filesystem. However, the FileSystemSetting is optional and may not be present.

8.2.8.4.1.4 NAS File Share Model

Figure 128: "NAS File Share Instance Diagram" illustrates the classes mandatory to model File Shares for the NAS Profile.

Note: This part of the model is the same for both the NAS Head and the Self-contained NAS.

Figure 128: NAS File Share Instance Diagram


SystemDevice
LANProtocolEndPoint (Optional)
DeviceSAPImplementation (Optional)
BindsToLANEndpoint (Optional)
IPProtocolEndPoint (Optional)
HostedAccessPoint (Optional)
BindsTo (Optional)
TCPProtocolEndPoint (Optional)
BindsTo (Optional)
NetworkPort
ProtocolEndPoint
ProtocolIFType= 4200 | 4201 ('NFS" or "CIFS")
DeviceSAPImplementation
SAPAvailableForElement
FileShare NFS or CIFS
ElementSettingData
ExportedFileShareSetting
ConcreteDependency (Optional)
LogicalFile
Hosted AccessPoint
HostedShare
FileStorage
LocalFileSystem
ElementSettingData (Optional)
FileSystemSetting (Optional)
ComputerSystem
HostedFileSystem

The NAS Profile shall model any File Shares that have been exported to the network. A File Share shall be represented as a FileShare instance with associations to the ComputerSystem that hosts the share (via HostedShare), to the ExportedFileShareSetting (via ElementSettingData) and to the ProtocolEndpoint through which the Share can be accessed (via SAPAvailableForElement). Optionally, there may also be an association between the FileShare and the LogicalFile that the share represents (via ConcreteDependency).

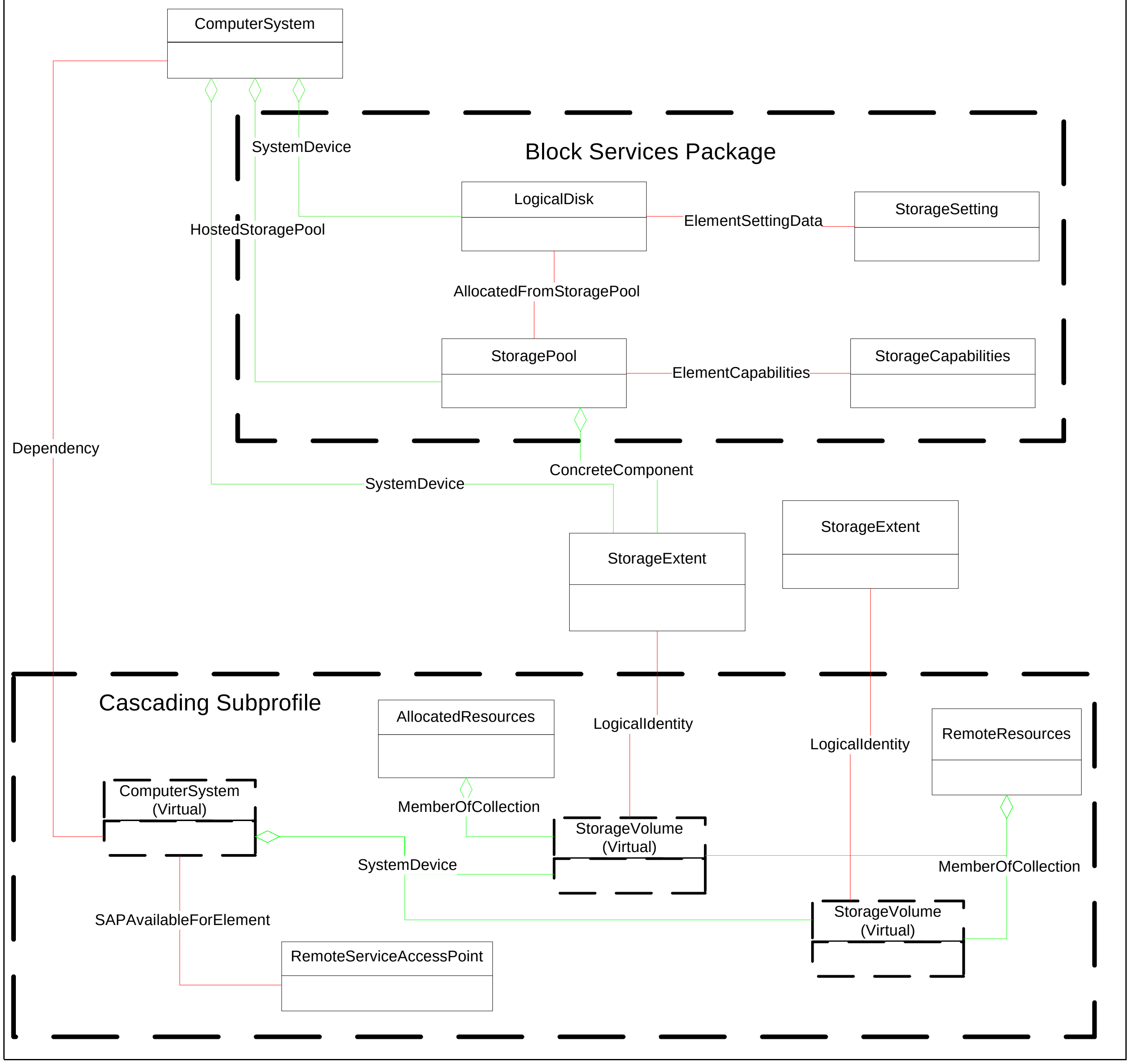
The LogicalFile on which the FileShare is based shall have a FileStorage association to the LocalFileSystem in which it resides.

EXPERIMENTAL

8.2.8.4.1.5 NAS Head Support of Cascading

Figure 129: "NAS Head Cascading Support Instance Diagram"illustrates the NAS Head support for cascading. Support for the Cascading Subprofile is optional (and the Cascading Subprofile is experimental). It is provided here to illustrate stitching between the NAS Head and Array or Storage Virtualizer Profiles.

Figure 129: NAS Head Cascading Support Instance Diagram



The lower dashed box in the figure illustrates the classes and associations of the Cascading Subprofile. The dashed classes are virtual instances (copies cached from the Array or Storage Virtualizer Profile). The other classes of the Cascading Subprofile represent NAS Head usage of those classes. For example, the collection AllocatedResources collects all the Array volumes that are used in StoragePools of the NAS Head. The RemoteResources collection collects all volumes that the NAS Head has discovered (whether used or not).

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

EXPERIMENTAL

8.2.8.4.2 Health and Fault Management Considerations

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

- Network Ports (See 8.2.8.4.2.1, "OperationalStatus for Network Ports")
- Back-end Ports (See 8.2.3.2, "Fibre Channel Initiator Port Subprofile", 8.2.3.1, "Parallel SCSI (SPI) Initiator Port Subprofile" and 8.2.3.3, "iSCSI Initiator Port Subprofile")
- ComputerSystems (See 6.3, "Health and Fault Management")
- FileShares that are exported (See 8.2.8.4.9.11, "CIM_ExportedFileShareSetting")
- Local File Systems (See 8.2.8.4.2.2, "OperationalStatus for FileShares")
- ProtocolEndpoints (See 8.2.8.4.2.4, "OperationalStatus for ProtocolEndpoints")

8.2.8.4.2.1 OperationalStatus for Network Ports

Table 864: NetworkPort OperationalStatus

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

8.2.8.4.2.2 OperationalStatus for FileShares

Table 865: FileShare OperationalStatus

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

8.2.8.4.2.3 OperationalStatus for Filesystems

Table 866: Filesystem OperationalStatus

OperationalStatus	Description
OK	The Filesystem has good status
Stressed	Filesystem resources are stressed
Degraded	The Filesystem is operating in a degraded mode. This could be due to the OperationalStatus of the underlying storage.
Predictive Failure	Filesystem might fail
Lost Communications	Filesystem cannot be accessed - if this happens in real-time, the opStatus is Lost Communication, otherwise it is Stopped.
Error	The Filesystem is not functioning
Non-recoverable Error	The Filesystem is not functioning and no SMI-S action will fix the problem.
Supporting Entity in Error	FileSystem is in an error state because a supporting entity is not accessible
Starting	The Filesystem is in process of initialization
Stopping	The Filesystem is in process of stopping
Stopped	The Filesystem is stopped
Dormant	The Filesystem is offline

8.2.8.4.2.4 OperationalStatus for ProtocolEndpoints

Table 867: ProtocolEndpoint OperationalStatus

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

EXPERIMENTAL

8.2.8.4.3 Cascading Considerations

The NAS Head is a cascading Profile, but the Cascading Subprofile is Experimental in this version of SMI-S. As such, the Cascading Subprofile is defined as an optional subprofile. A NAS Head may cascade storage. The cascading considerations for this are discussed in the following sections.

8.2.8.4.3.1 Cascading Resources for the NAS Head Profile

By definition, a NAS Head gets its storage from the network. As such, there is a cascading relationship between the NAS Head Profile and the Profiles (e.g., Array Profiles) that provide the storage for the NAS Head. Figure 129: "NAS Head Cascading Support Instance Diagram" illustrates the constructs to be used to model this cascading relationship.

- The NAS Head Cascaded Resources are Primordial StorageExtents (used to populate Primordial StoragePools)
  - The NAS Head obtains the storage for these from Array or Storage Virtualizer Profiles
  - Each Primordial StorageExtent maps (via LogicalIdentity) to a StorageVolume (from the Array or Storage Virtualizer Profile).

8.2.8.4.3.2 Ownership Privileges Asserted by NAS Heads

In support of the Cascading NAS Heads may assert ownership over the StorageVolumes that they import. If the Array or Storage Virtualizer implementation supports Ownership, NAS Heads would assert ownership using the following Privileges:

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

8.2.8.4.3.3 NAS Head Limitations on use of the Cascading Subprofile

The NAS Head support for Cascading places the following limitations and restrictions on the Cascading Subprofile:

- The AllocationService is not supported. - Allocation is done as a side effect of assigning the extents to the Primordial pool.
- Dependency - The Dependency between the NAS Head top level ComputerSystem and the Array or Virtualizer top level ComputerSystem may exist, even when there are no resources that are imported. This signifies that the NAS Head has discovered the Array or Virtualizer, but has no access to any of their volumes.

EXPERIMENTAL

8.2.8.4.4 Supported Subprofiles and Packages

Table 868: Supported Subprofiles for NAS Head

Registered Subprofile Names	Mandatory	Version
Indication	Yes	1.1.0

Table 868: Supported Subprofiles for NAS Head

Registered Subprofile Names	Mandatory	Version
Cascading	No	1.1.0
Access Points	No	1.1.0
Multiple Computer System	No	1.1.0
Software	No	1.1.0
Location	No	1.1.0
Extent Composition	No	1.1.0
File System Manipulation	No	1.1.0
File Export Manipulation	No	1.1.0
Job Control	No	1.1.0
SPI Initiator Ports	No	1.1.0
FC Initiator Ports	No	1.1.0
Device Credentials	No	1.1.0

Table 869: Supported Packages for NAS Head

Registered Package Names	Version
Physical Package	1.1.0
Block Services	1.1.0
Health	1.1.0

8.2.8.4.5 Methods of the Profile

8.2.8.4.5.1 Extrinsic Methods of the Profile

None.

8.2.8.4.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Manipulation functions are only supported for managing Indications (IndicationFilters, IndicationSubscriptions and ListenerDestinations).

8.2.8.4.6 Client Considerations and Recipes

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

```
sub GetFSServer(IN REF CIM_FileSystem $fs,
                 OUT CIM_ComputerSystem $system);

sub GetFSCapabilityFromServer(IN REF CIM_System $server,
                              OUT CIM_FileSystemConfigurationServiceCapabilities
$capability,
                              OUT CIM_FileSystemConfigurationService
$fsconfigurator,
                              IN Optional String $filesystemtype = "",
                              IN Optional String $otherpropertyname = NULL,
                              IN Optional String $otherpropertyvalue = NULL);

sub GetFSCapabilityFromFileSystem(IN REF CIM_FileSystem $fs,
                  OUT CIM_FileSystemConfigurationServiceCapabilities $capability,
                  OUT CIM_FileSystemConfigurationService $fsconfigurator);

sub GetExportServiceAndCapabilities(IN REF CIM_FileSystem $fs,
                                    IN String $sharetype,
                                    OUT CIM_FileExportService $feservice,
                                    OUT CIM_ExportedFileShareCapabilities
$efscapability);
```

Conventions used in the NAS recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is used without further validation. Real code will need to be more robust.
- We use Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.

8.2.8.4.6.1 List Existing Filesystems on the NAS

```
// DESCRIPTION
// The goal of this recipe is to locate all file systems hosted on the NAS.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1. A reference to the top-level ComputerSystem was previously discovered
// and is defined in the $NAS-> variable.

// Function ListFileSystems
// This function takes a given top-level ComputerSystem and locates the file
// systems which it hosts.
// Input:
//   A reference to the top-level ComputerSystem representing the NAS.
// Return:
//   An array of instance(s) to the file systems hosted on the NAS or null
//   if there are no hosted file systems.
sub CIMInstance[] ListFileSystems(REF $System->) {

    // Step 1. Locate the file systems hosted directly by the top-level
    // ComputerSystem representing the NAS.
    #FSProps[] = {“CSCreationClassName”, “CSName”, “CreationClassName”,
         “Name”, “OperationalStatus”, “CaseSensitive”, “CasePreserved”,
         “MaxFileNameLength”, “FileSystemType”, “IsFixedSize”}
    $FileSystems[] = Associators($System->,
         “CIM_HostedFileSystem”,
         “CIM_LocalFileSystem”,
         “GroupComponent”,
         “PartComponent”,
         false,
         false,
         #FSProps[])

    // Step 2. Locate any non-top-level ComputerSystems that may be present in
    // a NAS device that supports the Multiple Computer System Subprofile.
    try {
     $ComponentSystems->[] = AssociatorNames($System->,
         “CIM_ComponentCS,
         “CIM_ComputerSystem”,
         “GroupComponent”,
         “PartComponent”)

     // Step 3. Locate the file systems hosted by each non-top-level
     // ComputerSystems and add them to the list of known file systems.
     if ($ComponentSystems->[] != null && $ComponentSystems->[].length > 0) {
         $ComponentFS[]
         for (#i in $ComponentSystems->[]) {
         #fsCounter = $FileSystems[].length
         $ComponentFS[] = Associators($ComponentSystems->[#i],
             “CIM_HostedFileSystem”,
             “CIM_LocalFileSystem”,
             “GroupComponent”,
             “PartComponent”,
             false,
             false,
             #FSProps[])
         if ($ComponentFS[] != null && $ComponentFS[].length > 0) {
              for (#j in $ComponentFS[]) {
             $FileSystems[#fsCounter] = $ComponentFS[#j]
             #fsCounter++
              }
         }
         }
     }
    } catch (CIMException $Exception) {
     // ComponentCS may not be included in the model implemented at all if
     // the Multiple Computer System Subprofile is not supported.
     if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
         return $FileSystems[]
     }
     <ERROR! An unexpected failure occured>
    }
    return $FileSystems[]
}

// MAIN
$FS[] = ListFileSystems($NAS->)
```

8.2.8.4.6.2 Get the ComputerSystem that hosts a FileSystem

```
//
// Get the ComputerSystem that hosts a FileSystem
//
sub GetFSServer(IN REF CIM_FileSystem $fs,
        OUT CIM_ComputerSystem $system)
{
  $system = Associators($fs,
             “CIM_HostedFileSystem”,
             “CIM_ComputerSystem”,
             “PartComponent”,
             “GroupComponent”)->[0];
}
```

8.2.8.4.6.3 List Existing FileShares on the NAS

```
//
// List the shares on a Server
//

sub ListSystemShares(IN CIM_System $server,
                    OUT CIM_FileSystem $shares[])
{
    //
    // Use the HostedShare Association
    //
    $shares[] = Associators($server,
                            “CIM_HostedShare”,
                            “CIM_Share”,
                            “Antecedent”,
                            “Dependent”);
}
```

8.2.8.4.7 Registered Name and Version

NAS Head version 1.1.0

8.2.8.4.8 CIM Server Requirements

Table 870: CIM Server Requirements for NAS Head

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.4.9 CIM Elements

Table 871: CIM Elements for NAS Head

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.8.4.9.3)	This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the NAS Head.
CIM_ComputerSystem (8.2.8.4.9.4)	This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top level system. The File Server(s) must be manageable as a computer system and so could be exposed through other profiles and so there must be a way to correlate it with other management clients.
CIM_ConcreteComponent (8.2.8.4.9.5)	Represents the association between a Primordial StoragePool and the underlying StorageExtents that compose it.
CIM_DeviceSAPImplementation (8.2.8.4.9.7)	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
CIM_ElementSettingData (8.2.8.4.9.9)	(FileShare) Associates a configuration setting to the configured element. It is used in this Profile with FileShare and ExportedFileShareSetting elements.
CIM_ExportedFileShareSetting (8.2.8.4.9.11)	The configuration settings for an Exported FileShare that is a setting for a FileShare available for exporting.
CIM_FileShare (8.2.8.4.9.12)	Represents the sharing characteristics of a particular file element.
CIM_HostedAccessPoint (8.2.8.4.9.14)	(CIFS or NFS) Represents the association between a CIFS or NFS front end ProtocolEndpoint and the Computer System that hosts it.
CIM_HostedFileSystem (8.2.8.4.9.16)	Represents the association between a LocalFileSystem and the NAS Head (or FileServer) that hosts it.

Table 871: CIM Elements for NAS Head

Element Name	Description
CIM_HostedShare (8.2.8.4.9.17)	Represents that a shared element is hosted by a NAS Head Computer System.
CIM_LocalFileSystem (8.2.8.4.9.20)	Represents a LocalFileSystem of a NAS Head.
CIM_LogicalDisk (8.2.8.4.9.21)	Represents the single Storage Extent on which the NAS Head will build a LocalFileSystem.
CIM_LogicalFile (8.2.8.4.9.22)	The NAS Head Profile only makes a limited set of LogicalFiles (or Directory subclass) instances visible. These are any file or directory that is exported as a share.
CIM_NetworkPort (8.2.8.4.9.23)	Represents the front end logical port that supports access to a local area network.
CIM_ProtocolEndPoint (8.2.8.4.9.24)	(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
CIM_SAPAvailableForElement (8.2.8.4.9.26)	Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
CIM_StorageExtent (8.2.8.4.9.27)	This StorageExtent represents the LUNs (StorageVolumes) imported from a storage device to the NAS Head.
CIM_SystemDevice (8.2.8.4.9.28)	This association links all **LogicalDevices** to the scoping system. This is used to represent both front end and back end devices.
Optional Classes	
CIM_BindsTo (8.2.8.4.9.1)	Associates a higher level ProtocolEndpoint to an underlying ProtocolEndpoint. This is used in the NAS Head to support the TCP/IP Network protocol stack.
CIM_BindsToLANEndpoint (8.2.8.4.9.2)	Associates an IPProtocolEndpoint to an underlying LANEndpoint in the NAS Head (to support the TCP/IP Network protocol stack).
CIM_ConcreteDependency (8.2.8.4.9.6)	Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based.
CIM_DeviceSAPImplementation (8.2.8.4.9.8)	(LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
CIM_ElementSettingData (8.2.8.4.9.10)	(FileSystem) Associates a configuration setting to the configured element. It is used in this Profile with LocalFileSystem and FileSystemSetting elements.
CIM_FileSystemSetting (8.2.8.4.9.13)	This element represents the configuration settings of a File System.
CIM_HostedAccessPoint (8.2.8.4.9.15)	(TCP, IP or LAN) Represents the association between a front end TCP, IP or LAN ProtocolEndpoint and the Computer System that hosts it.
CIM_IPProtocolEndpoint (8.2.8.4.9.18)	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
CIM_LANEndpoint (8.2.8.4.9.19)	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.

Table 871: CIM Elements for NAS Head

Element Name	Description
CIM_ResidesOnExtent (8.2.8.4.9.25)	Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.
CIM_TCPProtocolEndpoint (8.2.8.4.9.29)	Represents the front-end ProtocolEndpoint used to support TCP services.
Mandatory Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus[*] <> PreviousInstance.CIM_ComputerSystem::OperationalStatus[*]	CQL - Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.CIM_NetworkPort::OperationalStatus[*]<> PreviousInstance.CIM_NetworkPort::OperationalStatus[*]	CQL - Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.CIM_ProtocolEndpoint::OperationalStatus[*] <> PreviousInstance.CIM_ProtocolEndpoint::OperationalStatus[*]	CQL - Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LocalFileSystem AND SourceInstance.CIM_LocalFileSystem::OperationalStatus[*] <> PreviousInstance.CIM_LocalFileSystem::OperationalStatus[*]	CQL - Change of Status of a Filesystem. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.CIM_FileShare::OperationalStatus[*] <> PreviousInstance.CIM_FileShare::OperationalStatus[*]	CQL - Change of Status of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus[*] <> PreviousInstance.CIM_LogicalDisk::OperationalStatus[*]	CQL - Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

8.2.8.4.9.1 CIM_BindsTo

Associates a higher level ProtocolEndpoint to an underlying ProtocolEndpoint. This is used in the NAS Head to support the TCP/IP Network protocol stack.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 872: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ProtocolEndpoint that uses a lower level ProtocolEndpoint for connectivity.

Table 872: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Antecedent		CIM_ProtocolEndpoint	The ProtocolEndpoint that supports a higher-level ProtocolEndpoint.

8.2.8.4.9.2 CIM_BindsToLANEndpoint

Associates an IPProtocolEndpoint to an underlying LANEndpoint in the NAS Head (to support the TCP/IP Network protocol stack).
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 873: SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	A TCPProtocolEndpoint.
Antecedent		CIM_LANEndpoint	A LANEndpoint.
FrameType		uint16	Only supports 1="Ethernet" at this point.

8.2.8.4.9.3 CIM_ComputerSystem

This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the NAS Head.
Created By : Static
Modified By : External
Deleted By : Static
Class Mandatory: true

Table 874: SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	The actual class of this object, e.g., Vendor_NASComputerSystem.
ElementName		string	User-friendly name
Name		string	Unique identifier for the NAS Head in a format specified by NameFormat. For example, IP address or Vendor/Model/SerialNo.
OperationalStatus		uint16[]	Overall status of the NAS Head
NameFormat		string	Format for Name property.
Dedicated		uint16[]	This shall be a NAS Head (24).
OtherIdentifyingInfo	C	string[]	An array of know identifiers for the NAS Head.

Table 874: SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Property	Flags	Type	Description & Notes
IdentifyingDescriptions	C	string[]	An array of descriptions of the OtherIdentifyingInfo.Some of the descriptions would be Ipv4 Address, Ipv6 Address or Fully Qualified Domain Name.
Optional Properties/Methods			
PrimaryOwnerContact	M	string	Owner of the NAS Head
PrimaryOwnerName	M	string	Contact details for owner

8.2.8.4.9.4 CIM_ComputerSystem

This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top level system. The File Server(s) must be manageable as a computer system and so could be exposed through other profiles and so there must be a way to correlate it with other management clients.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 875: SMI Referenced Properties/Methods for CIM_ComputerSystem (File Server)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dedicated		uint16[]	This is a File Server (Dedicated=16).It could also support other capabilities, so we do not restrict the values that can be in the Dedicated array.
NameFormat		string	Format for Name property. This shall be "Other".
Name	C	string	Unique identifier for the NAS Head's File Servers. e.g. Vendor/Model/SerialNo+FS+Number. The Fileserver can have any number of IP addresses, so an IP address does not constitute a single unique id. Also, under various load-balancing or redundancy regimens, the IP address could move around, so it may not even be correlatable. For that reason, the vendor must support a format that will provide a unique ID for the file server.
OperationalStatus		uint16[]	Overall status of the File Server.

8.2.8.4.9.5 CIM_ConcreteComponent

Represents the association between a Primordial StoragePool and the underlying StorageExtents that compose it.
Created By : External
Modified By : Static
Deleted By : External

Class Mandatory: true

Table 876: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ManagedElement	The Primordial StoragePool that is built from the StorageExtent.
PartComponent		CIM_ManagedElement	A StorageExtent that is part of a Primordial StoragePool.

8.2.8.4.9.6 CIM_ConcreteDependency

Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 877: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The LogicalFile that is being shared.
Dependent		CIM_ManagedElement	The Share that represents the LogicalFile being shared.

8.2.8.4.9.7 CIM_DeviceSAPImplementation

(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 878: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ProtocolEndpont that supports on the NetworkPort. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		CIM_LogicalDevice	The NetworkPort supported by the Access Point.

8.2.8.4.9.8 CIM_DeviceSAPImplementation

(LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
Created By : External
Modified By : Static
Deleted By : External

Class Mandatory: false

Table 879: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	A LANEndpoint that depends on a NetworkPort for connecting to its LAN segment.
Antecedent		CIM_LogicalDevice	The Logical network adapter device that connects to a LAN.

8.2.8.4.9.9 CIM_ElementSettingData

(FileShare) Associates a configuration setting to the configured element. It is used in this Profile with FileShare and ExportedFileShareSetting elements.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 880: SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The FileShare.
SettingData		CIM_SettingData	The current configuration of the FileShare.

8.2.8.4.9.10 CIM_ElementSettingData

(FileSystem) Associates a configuration setting to the configured element. It is used in this Profile with LocalFileSystem and FileSystemSetting elements.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 881: SMI Referenced Properties/Methods for CIM_ElementSettingData (FileSystem)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The LocalFileSystem.
SettingData		CIM_SettingData	The current configuration of the LocalFileSystem.

8.2.8.4.9.11 CIM_ExportedFileShareSetting

The configuration settings for an Exported FileShare that is a setting for a FileShare available for exporting.
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 882: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Setting)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	A unique ID for the setting.
ElementName		string	A user-friendly name for a Setting.
FileSharingProtocol		uint16	The file sharing protocol supported by this share. NFS (2) and CIFS (3) are the supported values.
ProtocolVersions		string[]	An array of the versions of the supported file sharing protocol. A share may support multiple versions of the same protocol.

8.2.8.4.9.12 CIM_FileShare

Represents the sharing characteristics of a particular file element.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 883: SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	A unique id for the FileShare element.
SharingDirectory		boolean	Indicates if the shared element is a file or a directory. This is useful when importing but less so when exporting.

8.2.8.4.9.13 CIM_FileSystemSetting

This element represents the configuration settings of a File System.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 884: SMI Referenced Properties/Methods for CIM_FileSystemSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a File System Setting.
ElementName		string	A User-Friendly Name for this Setting element.
ActualFileSystemType		uint16	This identifies the type of filesystem that this Setting represents.

Table 884: SMI Referenced Properties/Methods for CIM_FileSystemSetting

Property	Flags	Type	Description & Notes
FilenameCaseAttributes		uint16	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		uint16[]	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
NumberOfObjectsMin		uint64[]	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjectsMax		uint64[]	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjects		uint64[]	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		uint64[]	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMax		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
Optional Properties/Methods			
FilenameReservedCharacterSet		String[]	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames.
DataExtentsSharing		uint16	This allows the creation of data blocks (or storage extents) that are shared between files.
CopyTarget		uint16	This specifies if support should be provided for using the created file system as a target of a Copy operation.
FileNameStreamFormats		uint16[]	This is an array that specifies the stream formats supported for filenames by the created array (e.g., UTF-8).
FilenameFormats		uint16[]	This is an array that specifies the formats supported for filenames by the created array (e.g., DOS 8.3 names).

Table 884: SMI Referenced Properties/Methods for CIM_FileSystemSetting

Property	Flags	Type	Description & Notes
FilenameLengthMax		uint16[]	This specifies the maximum length of a filename supported by this capabilities.
SupportedLockingSemantics		uint16[]	This array specifies the kind of file access/locking semantics supported by this capabilities.
SupportedAuthorizationProtocols		uint16[]	This array specifies the kind of file authorization protocols supported by this capabilities.
SupportedAuthenticationProtocols		uint16[]	This array specifies the kind of file authentication protocols supported by this capabilities.

8.2.8.4.9.14 CIM_HostedAccessPoint

(CIFS or NFS) Represents the association between a CIFS or NFS front end ProtocolEndpoint and the Computer System that hosts it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 885: SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ServiceAccessPoint hosted on the FileServer. These include ProtocolEndpoints for NFS or CIFS.
Antecedent		CIM_System	The Computer System hosting this Access Point. In the context of the NAS Head, these are always FileServers (Dedicated=16).

8.2.8.4.9.15 CIM_HostedAccessPoint

(TCP, IP or LAN) Represents the association between a front end TCP, IP or LAN ProtocolEndpoint and the Computer System that hosts it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 886: SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP, IP or LAN)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ServiceAccessPoint hosted on the FileServer. These include ProtocolEndpoints for TCPProtocolEndpoints, IPProtocolEndpoints, and LANEndpoints.

Table 886: SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP, IP or LAN)

Property	Flags	Type	Description & Notes
Antecedent		CIM_System	The Computer System hosting this Access Point. In the context of the NAS Head, these are always FileServers (Dedicated=16).

8.2.8.4.9.16 CIM_HostedFileSystem

Represents the association between a LocalFileSystem and the NAS Head (or FileServer) that hosts it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 887: SMI Referenced Properties/Methods for CIM_HostedFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that hosts a LocalFileSystem. The NAS head hosts all the file systems made available to operational users, while the FileServer hosts a local "root" filesystem to support a filepath naming mechanism.
PartComponent		CIM_FileSystem	The hosted filesystem.

8.2.8.4.9.17 CIM_HostedShare

Represents that a shared element is hosted by a NAS Head Computer System.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 888: SMI Referenced Properties/Methods for CIM_HostedShare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_Share	The Share that is hosted by a File Server Computer System
Antecedent		CIM_System	The File Server Computer System that hosts a FileShare.

8.2.8.4.9.18 CIM_IPProtocolEndpoint

LAN endpoints supported are: 1="Other",6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".

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

Class Mandatory: false

Table 889: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		string	The CIM Class name of the Protocol Endpoint.
Name		string	The unique name of the Protocol End-point.
NameFormat		string	The Format of the Name.
OperationalStatus		uint16[]	The operational status of the PEP.
Description		string	This shall be the IP protocol endpoints supported by the NAS Head.
ProtocolIFType		uint16	4096="IP v4", 4097="IP v6", and 4098 is both. (Note that 1="Other" is not sup-ported)
IPv4Address		string	An IP v4 address in the format "A.B.C.D".
IPv6Address		string	
SubnetMask		string	An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		uint8	For an IPv6 address.
Optional Properties/Methods			
RequestedState		uint16	
EnabledState		uint16	
OtherEnabledState		string	
TimeOfLastStateChange		datetime	

8.2.8.4.9.19 CIM_LANEndpoint

Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 890: SMI Referenced Properties/Methods for CIM_LANEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the Protocol Endpoint.

Table 890: SMI Referenced Properties/Methods for CIM_LANEndpoint

Property	Flags	Type	Description & Notes
CreationClassName		string	The CIM Class name of the Protocol Endpoint.
Name		string	The unique name of the Protocol Endpoint.
NameFormat		string	The Format of the Name.
OperationalStatus		uint16[]	The operational status of the PEP.
Description		string	This shall be the LAN protocol endpoints supported by the NAS Head.
ProtocolIFType		uint16	LAN endpoints supported are: 1="Other",6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".
MACAddress		string	Primary Unicast address for this LAN device.
AliasAddresses		string[]	Other unicast addresses supported by this device.
GroupAddresses		string[]	Multicast addresses supported by this device.
MaxDataSize		uint32	The max size of packet supported by this LAN device.
Optional Properties/Methods			
RequestedState		uint16	
EnabledState		uint16	
OtherEnabledState		string	
TimeOfLastStateChange		datetime	
OtherTypeDescription		string	If the LAN endpoint is a vendor-extension specified by "Other" and a description.
LANID	N	string	A unique id for the LAN segment to which this device is connected. The value will be NULL if the LAN is not connected.

8.2.8.4.9.20 CIM_LocalFileSystem

Represents a LocalFileSystem of a NAS Head.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 891: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	The CIM class of the hosting NAS Head Computer System.

Table 891: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
CSName		string	The Name of the hosting NAS Head Computer System.
CreationClassName		string	The CIM class of this instance.
Name		string	A unique name for this Filesystem in the context of the hosting NAS Head.
OperationalStatus		uint16[]	The current operational status of the LocalFileSystem.
CaseSensitive		boolean	Whether this filesystem is sensitive to the case of characters in filenames.
CasePreserved		boolean	Whether this filesystem preserves the case of characters in filenames when saving and restoring.
MaxFileNameLength		uint32	The length of the longest filename.
FileSystemType		string	This matches ActualFileSystemType
Optional Properties/Methods			
Root		string	A path that specifies the root of the file-system in an unitary Computer Sys-tems acting as a FileServer.
BlockSize		uint64	The size of a block in bytes for certain filesystems that use a fixed block size when creating filesystems.
FileSystemSize		uint64	The total current size of the file system in blocks.
AvailableSpace		uint64	The space available currently in the file system in blocks. NOTE: This value is an approximation.
ReadOnly		boolean	Indicates that this is a read-only filesys-tem that does not allow modifications.
EncryptionMethod		string	Indicates if files are encrypted and the method of encryption.
CompressionMethod		string	Indicates if files are compressed before being stored, and the methods of com-pression.
CodeSet		uint16[]	The codeset used in filenames.
NumberOfFiles		uint64	The actual current number of files in the filesystem. NOTE: This value is an approximation.

8.2.8.4.9.21 CIM_LogicalDisk

Represents the single Storage Extent on which the NAS Head will build a LocalFileSystem.
Created By : ExternalExtrinsic(s):
Modified By : ExternalExtrinsic(s):
Deleted By : ExternalExtrinsic(s):

Class Mandatory: true

Table 892: SMI Referenced Properties/Methods for CIM_LogicalDisk (LD for FS)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	CIM Class of the NAS Head Computer System that is the host of this LogicalDisk.
SystemName		string	Name of the NAS Head Computer System that hosts this LogicalDisk.
CreationClassName		string	CIM Class of this instance of LogicalDisk.
DeviceID		string	Opaque identifier for the LogicalDisk.
OperationalStatus		uint16[]	A subset of operational status that is applicable for LogicalDisks in a NAS Head.
ExtentStatus		uint16[]	This LogicalDisk is neither imported (16) nor exported (17).
Primordial		boolean	This represents a Concrete Logical Disk that is not primordial.
Name		string	Identifier for a local LogicalDisk that will be used for a filesystem; since this logical disk will be referenced by a client, it must have a unique name. We cannot constrain the format here, but the OS-specific format described in the Block Services specification is not appropriate, so "Other" is used.
NameFormat		uint16	The format of the Name appropriate for LogicalDisks in the NAS Head. This shall be coded as 1 (other).

8.2.8.4.9.22 CIM_LogicalFile

The NAS Head Profile only makes a limited set of LogicalFiles (or Directory subclass) instances visible. These are any file or directory that is exported as a share.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 893: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	CIM Class of the Fileserver Computer System that hosts the Filesystem of this File.
CSName		string	Name of the Fileserver Computer System that hosts the Filesystem of this File.

Table 893: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
FSCreationClassName		string	CIM Class of the LocalFileSystem on the Fileserver Computer System that contains this File.
FSName		string	Name of the LocalFileSystem on the Fileserver Computer System that contains this File.
CreationClassName		string	CIM Class of this instance of LogicalFile.
Name		string	Name of this LogicalFile.

8.2.8.4.9.23 CIM_NetworkPort

LAN endpoints supported are: 1="Other", 6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 894: SMI Referenced Properties/Methods for CIM_NetworkPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A user-friendly name for this Network adapter that provides a network port.
OperationalStatus		uint16[]	The operational status of the adapter.
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Network Port.
SystemName		string	The name of the Computer System hosting the Network Port.
CreationClassName		string	The CIM Class name of the Network Port.
DeviceID		string	A unique ID for the device (in the context of the hosting System).
PermanentAddress	C	string	The hard-coded address of this port.
Optional Properties/Methods			
Speed		uint64	
MaxSpeed		uint64	
RequestedSpeed		uint64	
UsageRestriction		uint16	
PortType		uint16	
PortNumber		uint16	A unique number for the adapter in the context of the hosting System.
NetworkAddresses		string[]	An array of network addresses for this port.

Table 894: SMI Referenced Properties/Methods for CIM_NetworkPort

Property	Flags	Type	Description & Notes
LinkTechnology		uint16	1="Other", 2="Ethernet", 3="IB", 4="FC", 5="FDDI", 6="ATM", 7="Token Ring", 8="Frame Relay", 9="Infrared", 10="BlueTooth", 11="Wireless LAN.The link technology supported by this adapter.
OtherLinkTechnology		string	The vendor-specific "Other" link technology supported by this adapter.
FullDuplex		boolean	
AutoSense		boolean	
SupportedMaximumTransmission-Unit		uint64	
ActiveMaximumTransmissionUnit		uint64	

8.2.8.4.9.24 CIM_ProtocolEndPoint

(CIFS or NFS) Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 895: SMI Referenced Properties/Methods for CIM_ProtocolEndPoint (CIFS or NFS)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		string	The CIM Class name of the Protocol Endpoint.
Name		string	The unique name of the Protocol Endpoint.
NameFormat		string	The Format of the Name.
OperationalStatus		uint16[]	The operational status of the PEP.
Description		string	This shall be one of the NFS or CIFS protocol endpoints supported by the NAS Head.
ProtocolIFType		uint16	This represents either NFS=4200 or CIFS=4201. Other protocol types are specified in subclasses of ProtocolEndpoint.
Optional Properties/Methods			
RequestedState		uint16	
EnabledState		uint16	
OtherEnabledState		string	

Table 895: SMI Referenced Properties/Methods for CIM_ProtocolEndPoint (CIFS or NFS)

Property	Flags	Type	Description & Notes
TimeOfLastStateChange		datetime	

8.2.8.4.9.25 CIM_ResidesOnExtent

Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 896: SMI Referenced Properties/Methods for CIM_ResidesOnExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalElement	The local file system that is built on top of a LogicalDIsk.
Antecedent		CIM_StorageExtent	The LogicalDIsk that underlies a Local-FileSystem.

8.2.8.4.9.26 CIM_SAPAvailableForElement

Represents the association between a ServiceAccessPoint to the shared element that is being accessed through that SAP.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 897: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The element that is made available through a SAP. In the NAS Head, these are FileShares configured for either export or import.
AvailableSAP		CIM_ServiceAccessPoint	The Service Access Point that is available to this element.

8.2.8.4.9.27 CIM_StorageExtent

This StorageExtent represents the LUNs (StorageVolumes) imported from a storage device to the NAS Head.
Created By : Static or External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 898: SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	

Table 898: SMI Referenced Properties/Methods for CIM_StorageExtent (Primordial)

Property	Flags	Type	Description & Notes
SystemName		string	
CreationClassName		string	
DeviceID		string	
BlockSize		uint64	
NumberOfBlocks		uint64	
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
Name		string	Identifier for a remote LUN on a storage array; possibly, the array ID plus LUN Node WWN. This LUN is imported from a remote storage device, so the NameFormat identifies the remote LUN by identifying the remote array and the unique LUN ID at that array. As an example below, we have specified a 16-character hex format for the Name taken from the Node WWN format.
Primordial		boolean	The StorageExtent imported from an Array is considered primordial in the NAS Head.

8.2.8.4.9.28 CIM_SystemDevice

This association links all **LogicalDevices** to the scoping system. This is used to represent both front end and back end devices.
Created By : External or StaticExtrinsic(s):
Modified By : ExternalExtrinsic(s):
Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 899: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that contains this device.
PartComponent		CIM_LogicalDevice	The logical device that is a part of a computer system. These include StorageVolumes, NetworkPorts, 'back end' LogicalPorts for accessing storage, StorageExtents, protocol controllers, and so on.

8.2.8.4.9.29 CIM_TCPProtocolEndpoint

Represents the front-end ProtocolEndpoint used to support TCP services.
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: false

Table 900: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		string	The CIM Class name of the Protocol Endpoint.
Name		string	The unique name of the Protocol Endpoint.
NameFormat		string	The Format of the Name.
OperationalStatus		uint16[]	The operational status of the PEP.
Description		string	This shall be the TCP protocol endpoints supported by the NAS Head.
ProtocolIFType		uint16	4111="TCP". Note that no other protocol type is supported by this endpoint.
PortNumber		uint32	The number of the TCP Port that this element represents.
Optional Properties/Methods			
RequestedState		uint16	
EnabledState		uint16	
OtherEnabledState		string	
TimeOfLastStateChange		datetime	

8.2.8.4.10 Related Standards

Table 901: Related Standards for NAS Head

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

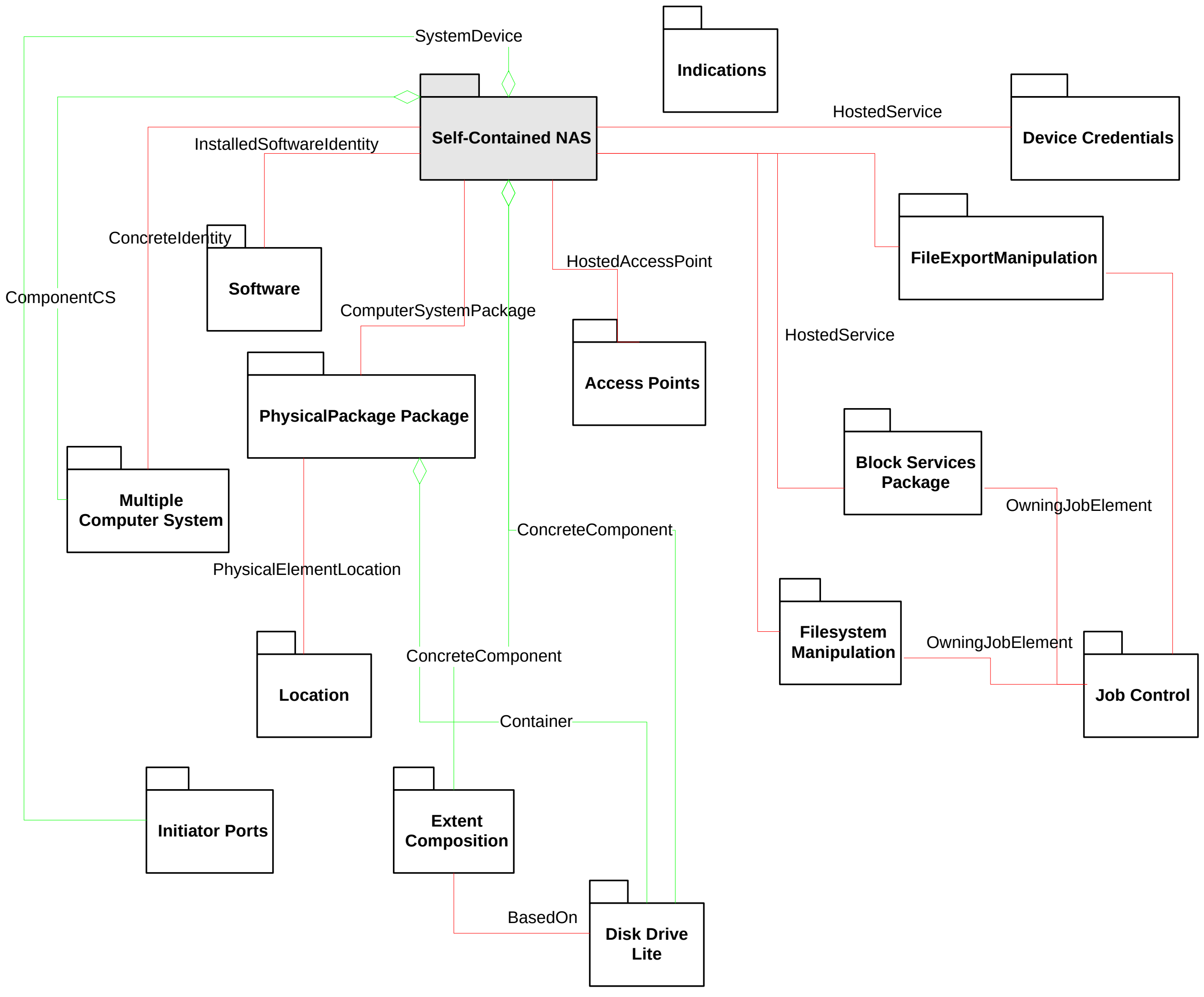
8.2.8.5 Self-Contained NAS Profile

8.2.8.5.1 Description

The Self-contained NAS profile defines NAS systems that are self-contained in that all the storage they use to store the NAS data is part of the NAS System (and not exposed). As a result, the Self-contained NAS profile needs to be able to address aspects of physical storage.

The Self-contained NAS profile reuses a significant portion of the Array Profile. This is illustrated in Figure 130: "Self-Contained NAS Profile and Subprofiles".

Figure 130: Self-Contained NAS Profile and Subprofiles



The Self-Contained NAS Profile and its subprofiles provide the following capabilities to SMI-S:

Device Level Configuration

- The Self-Contained NAS Profile defines reporting of physical storage. This includes configuration of storage at the Disk Drive level.

- The Self-Contained NAS Profile supports indications on OperationalStatus of the NAS processors.

Connectivity Level Configuration

- The Self-Contained NAS Profile defines reporting on port connectivity to the Self-Contained NAS.
- The Self-Contained NAS Profile supports indications on OperationalStatus of the NAS Ports and ProtocolEndpoints.
- The File Export Manipulation Subprofile defines mechanism for establishing file access through port connectivity to the NAS offering.

Block Level Configuration

- The Self-Contained NAS Profile defines reporting on logical storage (StoragePools) and LogicalDisks on those pools.

Note: Filesystems are built on the LogicalDisks.

- The Self-Contained NAS Profile supports indications on OperationalStatus of the LogicalDisks.

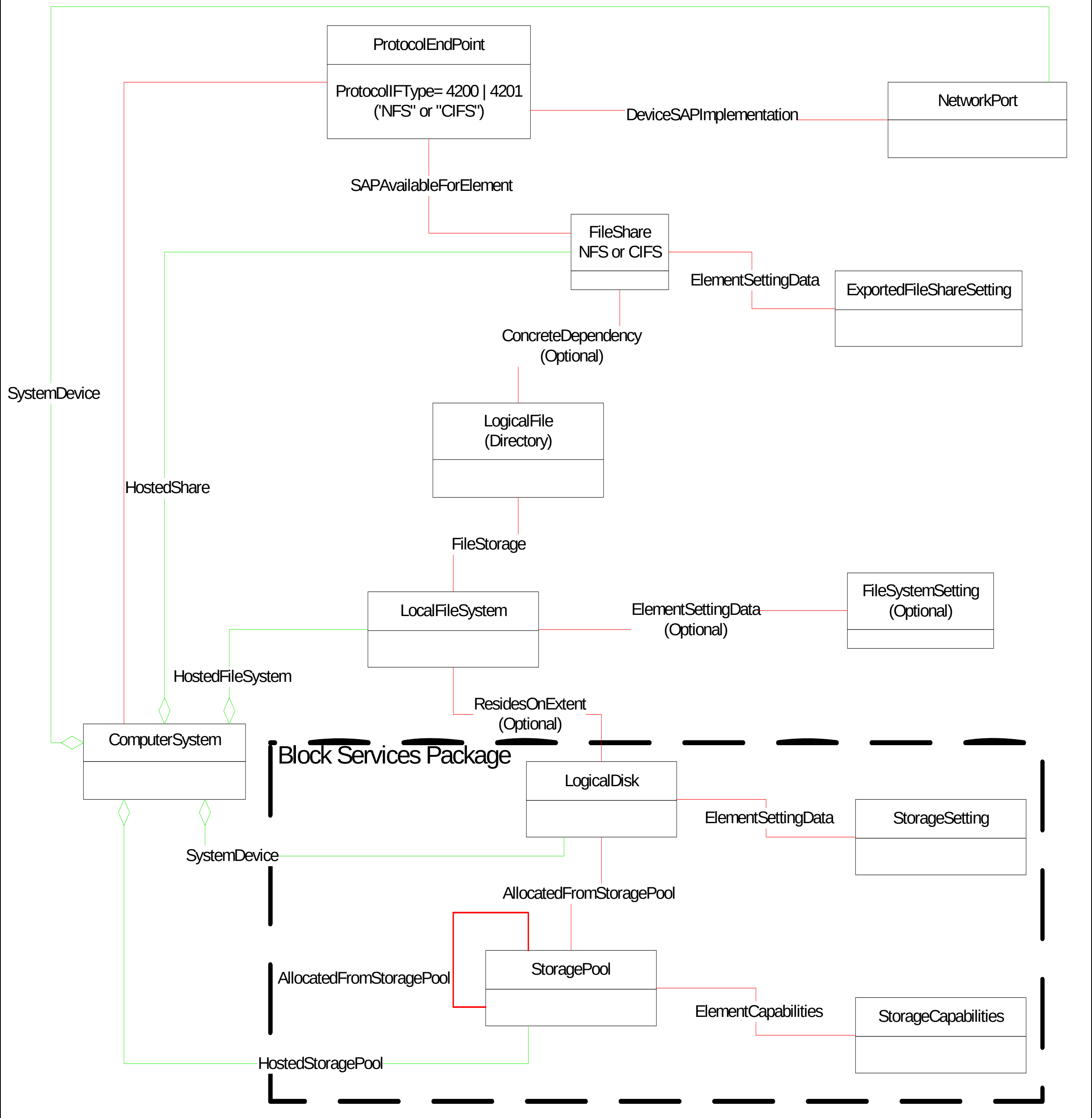
File/Record Configuration

- The Self-Contained NAS Profile defines reporting on the file systems and file shares that are configured out of the storage of the Self-Contained NAS.
- The Self-Contained NAS Profile supports indications on OperationalStatus of the FileSystems and FileShares.
- The Filesystem Manipulation Subprofile defines the ability to configure file systems out of the storage of the NAS Profile it supports.
- The File Export Manipulation Subprofile defines mechanism for establishing file shares on local file systems that can then be accessed by remote clients.

8.2.8.5.1.1 Summary Instance Diagram

Figure 131: "Self-Contained NAS Instance Diagram" illustrates the mandatory classes of the Self-Contained NAS Profile. This figure shows all the classes that are mandatory for the Self-contained NAS Profile. Later diagrams will review specific sections of this diagram.

Figure 131: Self-Contained NAS Instance Diagram



The Self-Contained NAS Profile closely parallels the Array Profile in how it models storage. Storage is assigned to StoragePools and LogicalDisks are allocated from those storage pools for the purpose of holding local file systems of the NAS.

As with the Array profile, the Self-contained NAS StoragePools have StorageCapabilities associated to the StoragePools via ElementCapabilities. Similarly, LogicalDisks have StorageSettings, which are associated to the LogicalDisk via ElementSettingData. StoragePools are hosted by a ComputerSystem that represents the NAS “top level” system, and the LogicalDisks have a SystemDevice association to the “top level” ComputerSystem.

Note: As with Arrays, the StoragePools may be hosted by a component ComputerSystem if the profile has implemented the Multiple Computer System Subprofile.

As with Arrays, the “top level” ComputerSystem of the Self-Contained NAS does not (and typically isn’t) a real ComputerSystem. It is merely the ManagedElement upon which all aspects of the NAS offering are scoped.

Everything above the LogicalDisk is specific to NAS (and does not appear in the Array Profile). LocalFileSystems are created on the Logicaldisks, LogicialFiles within those LocalFileSystens are shared (FileShare) through ProtocolEndpoints associated with NetworkPorts.

Note: The classes and associations in the dashed box are from the Block Services Package.

The ConcreteDependency association is optional. It is shown here to illustrate a relationship between a FileShare and some Directory. However, the Directory need not be part of the LocalFileSystem. Similarly, the ResidesOnExtent is optional, but is shown here to illustrate that a LocalFileSystem may map to a LogicalDisk. However, other mappings to storage are also possible.

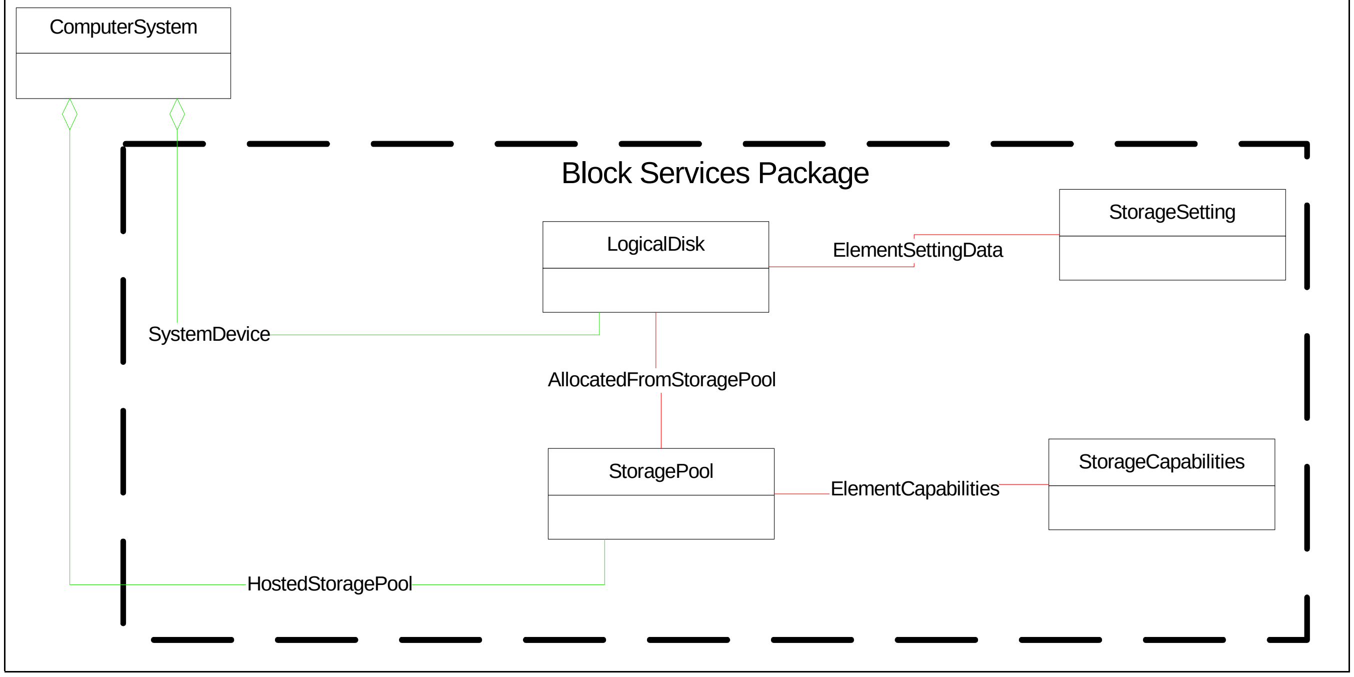
Also note that FileSystemSetting (and the corresponding ElementSettingData) are also optional. They are only shown here to illustrate where they would show up in the model should they be implemented.

In the base Self-Contained NAS profile, these are automatically populated based on how the Self-Contained NAS is configured. Client modification of the configuration (including configuring storage, creating extents, local file systems and file shares) are functions found in subprofiles of the profile.

8.2.8.5.1.2 NAS Storage Model

Figure 132: "NAS Storage Instance Diagram" illustrates the classes mandatory for modeling of storage for the Self-Contained NAS Profile.

Figure 132: NAS Storage Instance Diagram



The Self-Contained NAS Profile uses most of the classes and associations defined in the Array Profile (including those in the Block Services Package). In doing this, it leverages many of the subprofiles that are available for Array Profiles. The classes and associations shown in Figure 132: "NAS Storage Instance Diagram" are the minimum mandatory classes and associations of the Block Services Package for read only access in the base profile.

Storage for the NAS shall be modeled as logical storage. That is, StoragePools shall be modeled, including the HostedStoragePool and ElementCapabilities to the StorageCapabilities supported by the StoragePool. In addition, in order for storage to be used it shall be allocated to one or more LogicalDisks. A LogicalDisk shall have an AllocatedFromStoragePool association to the StoragePool from which it is allocated. And the LogicalDisk shall have an ElementSettingData association to the settings that were used when the LogicalDisk was created.

Note: At this level, the model for storage is the same for both the Self-contained NAS Profile and the NAS Head Profile. In the case of the Self-contained NAS, storage for the StoragePools is drawn from Disk Drives. Modeling of Disk Drives is Optional (See 8.2.8.14 "Disk Drive Lite Subprofile").

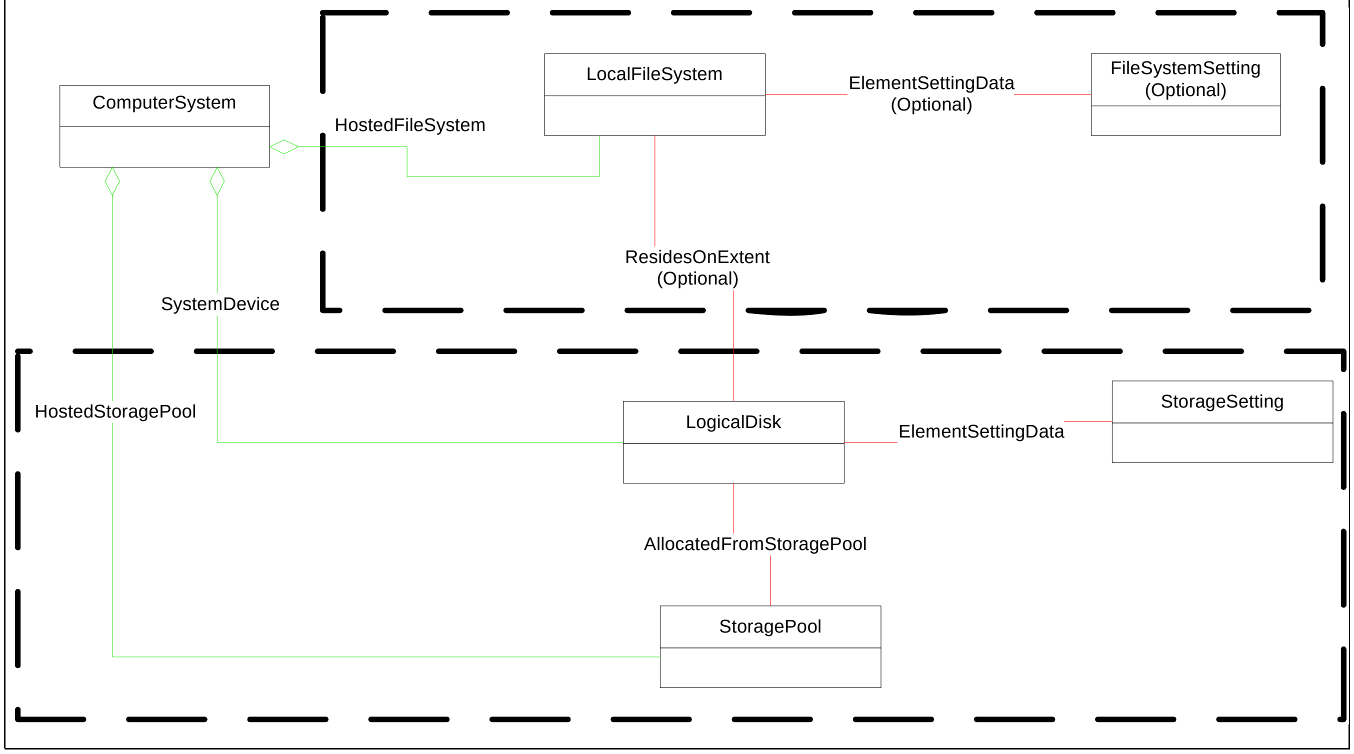
For manipulation of Storage, see8.2.8.10 "Block Services Package". For Self-Contained NAS, LogicalDisks are the ElementType that is supported for storage allocation functions (e.g., CreateOrModifyElementFromStoragePool and ReturnToStoragePool) and LogicalDisk creation is optional. NAS also supports (optionally) the Pool manipulation functions (e.g., CreateOrModifyStoragePool and DeleteStoragePool) of the Block Services Package.

8.2.8.5.1.3 NAS Filesystem Model

Figure 133: "NAS Filesystem Instance Diagram" illustrates the classes mandatory for the modeling of file systems for the NAS Profiles.

Note: This part of the model is the same for both the Self-contained NAS and the NAS Head.

Figure 133: NAS Filesystem Instance Diagram



The NAS Profile builds on the storage with Filesystems which are established on LogicalDisks. In the case of NAS Profiles, one Filesystem is established on one LogicalDisk.

Note: One Filesystem may also span multiple LogicalDisks or a Filesystem is may be allocated directly from a StoragePool, but these methods of storing a FileSystem are not covered by this version of SMI-S.

A Filesystem shall be represented in the model as instance of LocalFileSystem. A LocalFileSystem instance may have exactly one ResidesOnExtent association to one exactly one LogicalDisk. In this case, a client would determine the size (in bytes) of a Filesystem by inspecting the size of the LogicalDisk on which the filesystem resides. FileSystemSize can also be found as a property of LocalFileSystem. For other methods of FileSystem storage, the client should use the FileSystemSize property of the LocalFileSystem.

The FileSystem shall have a HostedFileSystem association to a NAS ComputerSystem. Normally this will be the top level ComputerSystem of the NAS profile. However, if the Multiple Computer System Subprofile is implemented, the HostedFileSystem may be associated to a component ComputerSystem (See 8.2.7.5 "SCSI Multipath Management Subprofile").

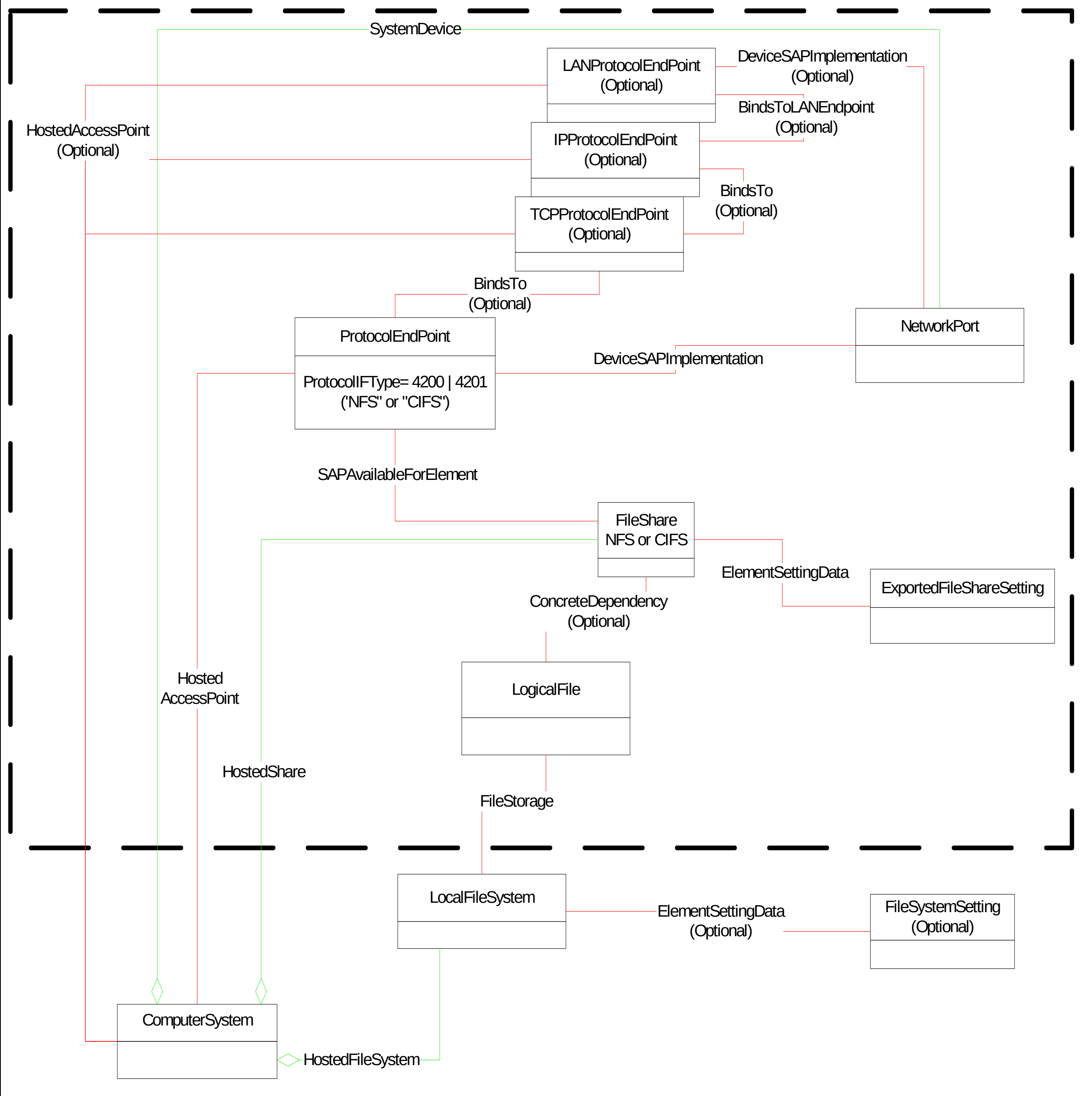
The LocalFileSystem instance may also have an ElementSettingData association to the FileSystemSetting for the Filesystem. However, the FileSystemSetting is optional and may not be present.

8.2.8.5.1.4 NAS File Share Model

Figure 134: "NAS File Share Instance Diagram" illustrates the classes mandatory for model File Shares for the NAS Profile.

Note: This part of the model is the same for both the Self-contained NAS and the NAS Head.

Figure 134: NAS File Share Instance Diagram


SystemDevice
LANProtocolEndPoint (Optional)
DeviceSAPImplementation (Optional)
BindsToLANEndpoint (Optional)
HostedAccessPoint (Optional)
IPProtocolEndPoint (Optional)
BindsTo (Optional)
TCPProtocolEndPoint (Optional)
BindsTo (Optional)
ProtocolEndPoint
ProtocolIFType= 4200 | 4201 ('NFS" or "CIFS")
NetworkPort
DeviceSAPImplementation
SAPAvailableForElement
FileShare NFS or CIFS
ElementSettingData
ExportedFileShareSetting
ConcreteDependency (Optional)
LogicalFile
Hosted AccessPoint
HostedShare
FileStorage
LocalFileSystem
ElementSettingData (Optional)
FileSystemSetting (Optional)
ComputerSystem
HostedFileSystem

The NAS Profile shall model any File Shares that have been exported to the network. A File Share shall be represented as a FileShare instance with associations to the ComputerSystem that hosts the share (via HostedShare), to the ExportedFileShareSetting (via ElementSettingData) and to the ProtocolEndpoint through which the Share can be accessed (via SAPAvailableForElement). Optionally, there may also be an association between the FileShare and the LogicalFile that the share represents (via ConcreteDependency).

The LogicalFile on which the FileShare is based shall have a FileStorage association to the Filesystem in which it resides.

EXPERIMENTAL

8.2.8.5.2 Health and Fault Management Considerations

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

- Network Ports (See Table 902)
- Back-end Ports (See 8.2.3.2 "Fibre Channel Initiator Port Subprofile", 8.2.3.1 "Parallel SCSI (SPI) Initiator Port Subprofile" and 8.2.3.3 "iSCSI Initiator Port Subprofile")
- ComputerSystems (See 8.2.1.6 "Health Package")
- FileShares that are exported (See Table 919: "SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting")
- Local File Systems (See Table 906, “Supported Subprofiles for Self-contained NAS System”)
- ProtocolEndpoints (See Table 905: "ProtocolEndpoint OperationalStatus")
- DiskDrive (See 8.2.8.14 "Disk Drive Lite Subprofile")

8.2.8.5.2.1 OperationalStatus for Network Ports

Table 902: NetworkPort OperationalStatus

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

8.2.8.5.2.2 OperationalStatus for FileShares

Table 903: FileShare OperationalStatus

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

8.2.8.5.2.3 OperationalStatus for Filesystems

Table 904: Filesystem OperationalStatus

OperationalStatus	Description
OK	The Filesystem has good status
Stressed	Filesystem resources are stressed
Degraded	The Filesystem is operating in a degraded mode. This could be due to the OperationalStatus of the underlying storage.
Predictive Failure	Filesystem might fail
Lost Communications	Filesystem cannot be accessed - if this happens in real-time, the opStatus is Lost Communication, otherwise it is Stopped.
Error	The Filesystem is not functioning
Non-recoverable Error	The Filesystem is not functioning and no SMI-S action will fix the problem.
Supporting Entity in Error	FileSystem is in an error state because a supporting entity is not accessible
Starting	The Filesystem is in process of initialization
Stopping	The Filesystem is in process of stopping
Stopped	The Filesystem is stopped
Dormant	The Filesystem is offline

8.2.8.5.2.4 OperationalStatus for ProtocolEndpoints

Table 905: ProtocolEndpoint OperationalStatus

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

EXPERIMENTAL

8.2.8.5.3 Cascading Considerations

Not Applicable.

8.2.8.5.4 Supported Subprofiles and Packages

Table 906: Supported Subprofiles for Self-contained NAS System

Registered Subprofile Names	Mandatory	Version
Indication	Yes	1.1.0
Access Points	No	1.1.0
Multiple Computer System	No	1.1.0
Software	No	1.1.0
Location	No	1.1.0
Extent Composition	No	1.1.0
File System Manipulation	No	1.1.0
File Export Manipulation	No	1.1.0
Job Control	No	1.1.0
Disk Drive Lite	No	1.1.0
SPI Initiator Ports	No	1.1.0
FC Initiator Ports	No	1.1.0
iSCSI Initiator Ports	No	1.1.0
Device Credentials	No	1.1.0

Table 907: Supported Packages for Self-contained NAS System

Registered Package Names	Version
Physical Package	1.1.0
Block Services	1.1.0
Health	1.1.0

8.2.8.5.5 Methods of the Profile

8.2.8.5.5.1 Extrinsic Methods of the Profile

None.

8.2.8.5.5.2 Intrinsic Methods of the Profile

The profile supports read methods and association traversal. Manipulation functions are only supported for managing Indications (IndicationFilters, IndicationSubscriptions and ListenerDestinations).

8.2.8.5.6 Client Considerations and Recipes

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

```
sub GetFSServer(IN REF CIM_FileSystem $fs,
                 OUT CIM_ComputerSystem $system);

sub GetFSCapabilityFromServer(IN REF CIM_System $server,
                              OUT CIM_FileSystemConfigurationServiceCapabilities
$capability,
                              OUT CIM_FileSystemConfigurationService
$fsconfigurator,
                              IN Optional String $filesystemtype = "",
                              IN Optional String $otherpropertyname = NULL,
                              IN Optional String $otherpropertyvalue = NULL);

sub GetFSCapabilityFromFileSystem(IN REF CIM_FileSystem $fs,
                   OUT CIM_FileSystemConfigurationServiceCapabilities $capability,
                   OUT CIM_FileSystemConfigurationService $fsconfigurator);

sub GetExportServiceAndCapabilities(IN REF CIM_FileSystem $fs,
                                    IN String $sharetype,
                                    OUT CIM_FileExportService $feservice,
                                    OUT CIM_ExportedFileShareCapabilities
$efscapability);
```

Conventions used in the NAS recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is used without further validation. Real code will need to be more robust.
- We use Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.

8.2.8.5.6.1 List Existing Filesystems on the NAS

```
// DESCRIPTION
// The goal of this recipe is to locate all file systems hosted on the
// Self-contained NAS.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1. A reference to the top-level ComputerSystem was previously discovered
// and is defined in the $SCNAS-> variable.

// Function ListFileSystems
// This function takes a given top-level ComputerSystem and locates the file
// systems which it hosts.
// Input:
//   A reference to the top-level ComputerSystem representing the
//   Self-contained NAS.
// Return:
//   An array of instance(s) to the file systems hosted on the Self-contained
//   NAS or null if there are no hosted file systems.
```

```
sub CIMInstance[] ListFileSystems(REF $System->) {

    // Step 1. Locate the file systems hosted directly by the top-level
    // ComputerSystem representing the Self-contained NAS.
    #FSProps[] = {“CSCreationClassName”, “CSName”, “CreationClassName”,
         “Name”}
    $FileSystems[] = Associators($System->,
         “CIM_HostedFileSystem”,
         “CIM_LocalFileSystem”,
         “GroupComponent”,
         “PartComponent”,
         false,
         false,
         #FSProps[])

    // Step 2. Locate any non-top-level ComputerSystems that may be present in
    // a Self-contained NAS device that supports the Multiple Computer System
    // Subprofile.
    try {
     $ComponentSystems->[] = AssociatorNames($System->,
        “CIM_ComponentCS,
        “CIM_ComputerSystem”,
        “GroupComponent”,
        “PartComponent”)

     // Step 3. Locate the file systems hosted by each non-top-level
     // ComputerSystems and add them to the list of known file systems.
     if ($ComponentSystems->[] != null && $ComponentSystems->[].length > 0) {
         $ComponentFS[]
         for (#i in $ComponentSystems->[]) {
        #fsCounter = $FileSystems[].length
        $ComponentFS[] = Associators($ComponentSystems->[#i],
            “CIM_HostedFileSystem”,
            “CIM_LocalFileSystem”,
            “GroupComponent”,
            “PartComponent”,
            false,
            false,
            #FSProps[])
        if ($ComponentFS[] != null && $ComponentFS[].length > 0) {
            for (#j in $ComponentFS[]) {
            $FileSystems[#fsCounter] = $ComponentFS[#j]
            #fsCounter++
            }
        }
         }
     }
    } catch (CIMException $Exception) {
     // ComponentCS may not be included in the model implemented at all if
     // the Multiple Computer System Subprofile is not supported.
     if ($Exception.CIMStatusCode == CIM_ERR_INVALID_PARAMETER) {
         return $FileSystems[]
     }
     <ERROR! An unexpected failure occured>
    }
    return $FileSystems[]
}

// MAIN
$FS[] = ListFileSystems($SCNAS->)
```

8.2.8.5.6.2 Get the ComputerSystem that hosts a FileSystem

```
//
// Get the ComputerSystem that hosts a FileSystem
//
sub GetFSServer(IN REF CIM_FileSystem $fs,
                OUT CIM_ComputerSystem $system)
{
    $system = Associators($fs,
                          “CIM_HostedFileSystem”,
                          “CIM_ComputerSystem”,
                          “PartComponent”,
                          “GroupComponent”)->[0];
}
```

8.2.8.5.6.3 List Existing FileShares on the NAS

```
//
// List the shares on a Server
//

sub ListSystemShares(IN CIM_System $server,
                    OUT CIM_FileSystem $shares[])
{
    //
    // Use the HostedShare Association
    //
    $shares[] = Associators($server,
                            “CIM_HostedShare”,
                            “CIM_Share”,
                            “Antecedent”,
                            “Dependent”);
}
```

8.2.8.5.7 Registered Name and Version

Self-contained NAS System version 1.1.0

8.2.8.5.8 CIM Server Requirements

Table 908: CIM Server Requirements for Self-contained NAS System

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.5.9 CIM Elements

Table 909: CIM Elements for Self-contained NAS System

Element Name	Description
Mandatory Classes	
CIM_ComputerSystem (8.2.8.5.9.3)	This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the Self-contained NAS System.
CIM_ComputerSystem (8.2.8.5.9.4)	This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top-level system. The File Server(s) shall be manageable as a computer system and so could be exposed through other profiles and so there must be a way to correlate it with other management clients.
CIM_DeviceSAPImplementation (8.2.8.5.9.6)	(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
CIM_ElementSettingData (8.2.8.5.9.8)	(FileShare) Associates a setting to the FileShare that the SC NAS System manages actively.
CIM_ExportedFileShareSetting (8.2.8.5.9.10)	The configuration settings for a FileShare that is available for exporting.
CIM_FileShare (8.2.8.5.9.11)	Represents the sharing characteristics of a particular file element.
CIM_FileStorage (8.2.8.5.9.12)	Associates a Logical File or Directory to the LocalFileSystem that contains it.
CIM_HostedAccessPoint (8.2.8.5.9.14)	(CIFS or NFS) Represents the association between a front end ProtocolEndpoint and the Computer System that hosts it.
CIM_HostedFileSystem (8.2.8.5.9.16)	Represents the association between a LocalFileSystem and the SC NAS System (or FileServer) that hosts it.
CIM_HostedShare (8.2.8.5.9.17)	Represents that a shared element is hosted by a SC NAS System Computer System.
CIM_LocalFileSystem (8.2.8.5.9.20)	Represents a LocalFileSystem of a SC NAS System Computer System.
CIM_LogicalDisk (8.2.8.5.9.21)	Represents LogicalDisks used for building LocalFileSystems.
CIM_LogicalFile (8.2.8.5.9.22)	The Self-Contained NAS Profile only makes a limited set of LogicalFiles (or Directory subclass) instances visible. These are any file or directory that is exported as a share.
CIM_NetworkPort (8.2.8.5.9.23)	Represents the front end logical port that supports access to a local area network.
CIM_ProtocolEndPoint (8.2.8.5.9.24)	Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.

Table 909: CIM Elements for Self-contained NAS System

Element Name	Description
CIM_SAPAvailableForElement (8.2.8.5.9.26)	Represents the association between a ProtocolEndpoint to the shared element that is being accessed through that SAP.
CIM_SystemDevice (8.2.8.5.9.27)	This association links all **LogicalDevices** to the scoping system. This is used to represent both front end and back end devices.
Optional Classes	
CIM_BindsTo (8.2.8.5.9.1)	Associates a higher level ProtocolEndpoint to an underlying ProtocolEndpoint. This is used in the SC NAS System to support the TCP/IP Network protocol stack.
CIM_BindsToLANEndpoint (8.2.8.5.9.2)	Associates an IPProtocolEndpoint to an underlying LANEndpoint in the SC NAS System (to support the TCP/IP Network protocol stack).
CIM_ConcreteDependency (8.2.8.5.9.5)	Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based.
CIM_DeviceSAPImplementation (8.2.8.5.9.7)	(LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
CIM_ElementSettingData (8.2.8.5.9.9)	(FileSystem) Associates a setting to the LocalFileSystem that the SC NAS System manages actively.
CIM_FileSystemSetting (8.2.8.5.9.13)	This element represents the configuration settings of a File System.
CIM_HostedAccessPoint (8.2.8.5.9.15)	(TCP, IP or LAN) Represents the association between a front end TCP, IP or LAN ProtocolEndpoint and the Computer System that hosts it.
CIM_IPProtocolEndpoint (8.2.8.5.9.18)	Represents the front-end ProtocolEndpoint used to support the IP protocol services.
CIM_LANEndpoint (8.2.8.5.9.19)	Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
CIM_ResidesOnExtent (8.2.8.5.9.25)	Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.
CIM_TCPProtocolEndpoint (8.2.8.5.9.28)	Represents the front-end ProtocolEndpoint used to support TCP services.
Mandatory Indications	
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND SourceInstance.CIM_ComputerSystem::OperationalStatus[*] <> PreviousInstance.CIM_ComputerSystem::OperationalStatus[*]	CQL - Change of Status of a NAS ComputerSystem (controller). PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPort AND SourceInstance.CIM_NetworkPort::OperationalStatus[*] <> PreviousInstance.CIM_NetworkPort::OperationalStatus[*]	CQL - Change of Status of a Port. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

Table 909: CIM Elements for Self-contained NAS System

Element Name	Description
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolEndpoint AND SourceInstance.CIM_ProtocolEndpoint::OperationalStatus[*] <> PreviousInstance.CIM_ProtocolEndpoint::OperationalStatus[*]	CQL - Change of Status of a ProtocolEndpoint PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LocalFileSystem AND SourceInstance.CIM_LocalFileSystem::OperationalStatus[*] <> PreviousInstance.CIM_LocalFileSystem::OperationalStatus[*]	CQL - Change of Status of a Filesystem. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.CIM_FileShare::OperationalStatus[*] <> PreviousInstance.CIM_FileShare::OperationalStatus[*]	CQL - Change of Status of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the Profile.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.CIM_LogicalDisk::OperationalStatus[*] <> PreviousInstance.CIM_LogicalDisk::OperationalStatus[*]	CQL - Change of status of a LogicalDisk. PreviousInstance is optional, but may be supplied by an implementation of the Profile.

8.2.8.5.9.1 CIM_BindsTo

Associates a higher level ProtocolEndpoint to an underlying ProtocolEndpoint. This is used in the SC NAS System to support the TCP/IP Network protocol stack.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 910: SMI Referenced Properties/Methods for CIM_BindsTo

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ProtocolEndpoint that uses a lower level ProtocolEndpoint for connectivity.
Antecedent		CIM_ProtocolEndpoint	The ProtocolEndpoint that supports a higher-level ProtocolEndpoint.

8.2.8.5.9.2 CIM_BindsToLANEndpoint

Associates an IPProtocolEndpoint to an underlying LANEndpoint in the SC NAS System (to support the TCP/IP Network protocol stack).
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: false

Table 911: SMI Referenced Properties/Methods for CIM_BindsToLANEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	A TCPProtocolEndpoint.
Antecedent		CIM_LANEndpoint	A LANEndpoint.
FrameType		uint16	Only supports 1="Ethernet" at this point.

8.2.8.5.9.3 CIM_ComputerSystem

This declares that at least one computer system entry will pre-exist. The Name property should be the Unique identifier for the Self-contained NAS System.
Created By : Static
Modified By : External
Deleted By : Static
Class Mandatory: true

Table 912: SMI Referenced Properties/Methods for CIM_ComputerSystem (Top Level)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	The actual class of this object, e.g., Vendor_NASComputerSystem.
ElementName		string	User-friendly name
Name		string	Unique identifier for the Self-contained NAS System in a format specified by NameFormat. For example, IP address or Vendor/Model/SerialNo.
OperationalStatus		uint16[]	Overall status of the Self-contained NAS System
NameFormat		string	Format for Name property.
Dedicated		uint16[]	This shall indicate that this computer system is dedicated to operation as a Self-contained NAS (25).
OtherIdentifyingInfo	C	string[]	An array of know identifiers for the NAS Head.
IdentifyingDescriptions	C	string[]	An array of descriptions of the OtherIdentifyingInfo. Some of the descriptions would be "Ipv4 Address", "Ipv6 Address" or "Fully Qualified Domain Name".
Optional Properties/Methods			
PrimaryOwnerContact	M	string	Owner of the Self-contained NAS System
PrimaryOwnerName	M	string	Contact details for owner

8.2.8.5.9.4 CIM_ComputerSystem

This declares that at least one computer system that provides File Server capabilities will pre-exist. This could be the same as the top-level ComputerSystem but this would not be true in a cluster, so this has a separate entry that is not tagged as a top-level system. The File Server(s) shall be manageable as a computer system and so could be exposed through other profiles and so there must be a way to correlate it with other management clients.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 913: SMI Referenced Properties/Methods for CIM_ComputerSystem (File Server)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dedicated		uint16[]	This is a File Server (16).
NameFormat		string	Format for Name property. This shall be "Other".
Name	C	string	Unique identifier for the Self-contained NAS System's File Servers. E.g., Vendor/Model/SerialNo+FS+Number. The Fileserver can have any number of IP addresses, so an IP address does not constitute a single unique id. Also, under various load-balancing or redundancy regimens, the IP address could move around, so it may not even be correlatable. For that reason, the vendor shall support a format that will provide a unique id for the file server.
OperationalStatus		uint16[]	Overall status of the File Server.

8.2.8.5.9.5 CIM_ConcreteDependency

Represents an association between a FileShare element and the actual shared LogicalFile or Directory on which it is based.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 914: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The LogicalFile that is being shared.
Dependent		CIM_ManagedElement	The Share that represents the LogicalFile being shared.

8.2.8.5.9.6 CIM_DeviceSAPImplementation

(CIFS or NFS to NetworkPort) Represents the association between a CIFS or NFS ProtocolEndpoint and the NetworkPort that it supports.
Created By : External
Modified By : Static

Deleted By : External
Class Mandatory: true

Table 915: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (CIFS or NFS to NetworkPort)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ProtocolEndpont that supports on the NetworkPort. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		CIM_LogicalDevice	The NetworkPort supported by the Access Point.

8.2.8.5.9.7 CIM_DeviceSAPImplementation

(LANEndpoint to NetworkPort) Associates a logical front end Port (a NetworkPort) to the LANEndpoint that uses that device to connect to a LAN.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 916: SMI Referenced Properties/Methods for CIM_DeviceSAPImplementation (LANEndpoint to NetworkPort)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	A LANEndpoint that depends on a NetworkPort for connecting to its LAN segment.
Antecedent		CIM_LogicalDevice	The Logical network adapter device that connects to a LAN.

8.2.8.5.9.8 CIM_ElementSettingData

(FileShare) Associates a setting to the FileShare that the SC NAS System manages actively.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 917: SMI Referenced Properties/Methods for CIM_ElementSettingData (FileShare)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The FileShare whose configuration settings are specified by the ExportedFileShareSetting.
SettingData		CIM_SettingData	The ExportedFileShareSetting that specifies a configuration setting for the FileShare.

8.2.8.5.9.9 CIM_ElementSettingData

(FileSystem) Associates a setting to the LocalFileSystem that the SC NAS System manages actively.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 918: SMI Referenced Properties/Methods for CIM_ElementSettingData (FileSystem)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The LocalFileSystem whose configuration settings are specified by the FileSystemSetting.
SettingData		CIM_SettingData	The FileSystemSetting that specifies a configuration setting for the FileSystem.

8.2.8.5.9.10 CIM_ExportedFileShareSetting

The configuration settings for a FileShare that is available for exporting.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 919: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	A unique ID for the setting.
ElementName		string	A user-friendly name for a Setting.
FileSharingProtocol		uint16	The file sharing protocol supported by this share. NFS is 2 and CIFS is 3 are the ones supported for SMI-S 1.1.0
ProtocolVersions		string[]	An array of the versions of the supported file sharing protocol. A share may support multiple versions of the same protocol.

8.2.8.5.9.11 CIM_FileShare

Represents the sharing characteristics of a particular file element.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 920: SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	A unique id for the FileShare element.

Table 920: SMI Referenced Properties/Methods for CIM_FileShare (Exported File Share)

Property	Flags	Type	Description & Notes
SharingDirectory		boolean	Indicates if the shared element is a file or a directory. This is useful when importing but less so when exporting.

8.2.8.5.9.12 CIM_FileStorage

Associates a Logical File or Directory to the LocalFileSystem that contains it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 921: SMI Referenced Properties/Methods for CIM_FileStorage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_FileSystem	The LocalFileSystem that contains the LogicalFile.
PartComponent		CIM_LogicalFile	The LogicalFile contained in the Local-FileSystem.

8.2.8.5.9.13 CIM_FileSystemSetting

This element represents the configuration settings of a File System.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 922: SMI Referenced Properties/Methods for CIM_FileSystemSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a File System Setting.
ElementName		string	A User-Friendly Name for this Setting element.
ActualFileSystemType		uint16	This identifies the type of filesystem that this Setting represents.
FilenameCaseAttributes		uint16	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		uint16[]	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
NumberOfObjectsMin		uint64[]	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[].

Table 922: SMI Referenced Properties/Methods for CIM_FileSystemSetting

Property	Flags	Type	Description & Notes
NumberOfObjectsMax		uint64[]	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjects		uint64[]	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		uint64[]	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMax		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
Optional Properties/Methods			
FilenameReservedCharacterSet		String[]	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames.
DataExtentsSharing		uint16	This allows the creation of data blocks (or storage extents) that are shared between files.
CopyTarget		uint16	This specifies if support should be provided for using the created file system as a target of a Copy operation.
FileNameStreamFormats		uint16[]	This is an array that specifies the stream formats supported for filenames by the created array (e.g., UTF-8).
FilenameFormats		uint16[]	This is an array that specifies the formats supported for filenames by the created array (e.g. DOS 8.3 names).
FilenameLengthMax		uint16[]	This specifies the maximum length of a filename supported by this capabilities.
SupportedLockingSemantics		uint16[]	This array specifies the kind of file access/locking semantics supported by this capabilities.
SupportedAuthorizationProtocols		uint16[]	This array specifies the kind of file authorization protocols supported by this capabilities.
SupportedAuthenticationProtocols		uint16[]	This array specifies the kind of file authentication protocols supported by this capabilities.

8.2.8.5.9.14 CIM_HostedAccessPoint

(CIFS or NFS) Represents the association between a front end ProtocolEndpoint and the Computer System that hosts it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 923: SMI Referenced Properties/Methods for CIM_HostedAccessPoint (CIFS or NFS)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ServiceAccessPoint hosted on the FileServer. These include ProtocolEndpoints for NFS and CIFS.
Antecedent		CIM_System	The Computer System hosting this Access Point. In the context of the SC NAS System, these are always FileServers (Dedicated=16).

8.2.8.5.9.15 CIM_HostedAccessPoint

(TCP, IP or LAN) Represents the association between a front end TCP, IP or LAN ProtocolEndpoint and the Computer System that hosts it.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 924: SMI Referenced Properties/Methods for CIM_HostedAccessPoint (TCP, IP or LAN)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ServiceAccessPoint	The ServiceAccessPoint hosted on the FileServer. These include ProtocolEndpoints for TCPProtocolEndpoints, IPProtocolEndpoints, and LANEndpoints among others.
Antecedent		CIM_System	The Computer System hosting this Access Point. In the context of the SC NAS System, these are always FileServers (Dedicated=16).

8.2.8.5.9.16 CIM_HostedFileSystem

Represents the association between a LocalFileSystem and the SC NAS System (or FileServer) that hosts it.
Created By : External
Modified By : Static
Deleted By : External

Class Mandatory: true

Table 925: SMI Referenced Properties/Methods for CIM_HostedFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that hosts a LocalFileSystem. The SC NAS System hosts all the file systems made available to operational users, while the FileServer hosts a local "root" filesystem to support a filepath naming mechanism.
PartComponent		CIM_FileSystem	The hosted filesystem.

8.2.8.5.9.17 CIM_HostedShare

Represents that a shared element is hosted by a SC NAS System Computer System.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 926: SMI Referenced Properties/Methods for CIM_HostedShare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_Share	The Share that is hosted by a File Server Computer System
Antecedent		CIM_System	The File Server Computer System that hosts a FileShare.

8.2.8.5.9.18 CIM_IPProtocolEndpoint

Represents the front-end ProtocolEndpoint used to support the IP protocol services.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 927: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the IP Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the IP Protocol Endpoint.
CreationClassName		string	The CIM Class name of the IP Protocol Endpoint.
Name		string	The unique name of the IP Protocol Endpoint.

Table 927: SMI Referenced Properties/Methods for CIM_IPProtocolEndpoint

Property	Flags	Type	Description & Notes
NameFormat		string	The Format of the Name of the IP Protocol Endpoint.
ProtocolIFType		uint16	4096="IP v4", 4097="IP v6", and 4098 is both. (Note that 1="Other" is not supported)
IPv4Address		string	An IP v4 address in the format "A.B.C.D".
IPv6Address		string	
SubnetMask		string	An IP v4 subnet mask in the format "A.B.C.D".
PrefixLength		uint8	For an IPv6 address.

8.2.8.5.9.19 CIM_LANEndpoint

Represents the front-end ProtocolEndpoint used to support a Local Area Network and its services.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 928: SMI Referenced Properties/Methods for CIM_LANEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the LAN Endpoint.
SystemName		string	The name of the Computer System hosting the LAN Endpoint.
CreationClassName		string	The CIM Class name of the LAN Endpoint.
Name		string	The unique name of the LAN Endpoint.
NameFormat		string	The Format of the Name for the LAN Endpoint.
ProtocolIFType		uint16	LAN endpoints supported are: 1="Other",6="Ethernet CSMA/CD", 9="ISO 802.5 Token Ring", 15="FDDI".
MACAddress		string	Primary Unicast address for this LAN device.
AliasAddresses		string[]	Other Unicast addresses supported by this device.
GroupAddresses		string[]	Multicast addresses supported by this device.
MaxDataSize		uint32	The max size of packet supported by this LAN device. (If there were a Network subprofile, this would not be exposed in a SC NAS System Profile).

Table 928: SMI Referenced Properties/Methods for CIM_LANEndpoint

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
OtherTypeDescription		string	If the LAN endpoint is a vendor-extension specified by "Other" and a description.
LANID		string	A unique id for the LAN segment that this device is connected to. Will be NULL if the LAN is not connected.

8.2.8.5.9.20 CIM_LocalFileSystem

Represents a LocalFileSystem of a SC NAS System Computer System.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 929: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	The CIM class of the hosting SC NAS System Computer System.
CSName		string	The Name of the hosting SC NAS System Computer System.
CreationClassName		string	The CIM class of this instance.
Name		string	A unique name for this Filesystem in the context of the hosting SC NAS System.
OperationalStatus		uint16[]	The current operational status of the LocalFileSystem.
CaseSensitive		boolean	Whether this filesystem is sensitive to the case of characters in filenames.
CasePreserved		boolean	Whether this filesystem preserves the case of characters in filenames when saving and restoring.
MaxFileNameLength		uint32	The length of the longest filename.
FileSystemType		string	This matches ActualFileSystemType
Optional Properties/Methods			
Root		string	A path that specifies the root of the filesystem in an unitary Computer Systems acting as a FileServer.
BlockSize		uint64	The size of a block in bytes for certain filesystems that use a fixed block size when creating filesystems.
FileSystemSize		uint64	The total current size of the file system in blocks.

Table 929: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
AvailableSpace		uint64	The space available currently in the file system in blocks. NOTE: This value is an approximation.
ReadOnly		boolean	Indicates that this is a read-only filesys-tem that does not allow modifications.
EncryptionMethod		string	Indicates if files are encrypted and the method of encryption.
CompressionMethod		string	Indicates if files are compressed before being stored, and the methods of com-pression.
CodeSet		uint16[]	The codeset used in filenames.
NumberOfFiles		uint64	The actual current number of files in the filesystem. NOTE: This value is an approximation.

8.2.8.5.9.21 CIM_LogicalDisk

Represents LogicalDisks used for building LocalFileSystems.
Created By : ExternalExtrinsic(s):
Modified By : ExternalExtrinsic(s):
Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 930: SMI Referenced Properties/Methods for CIM_LogicalDisk (Disk for FS)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	CIM Class of the SC NAS System Computer System that is the host of this LogicalDisk.
SystemName		string	Name of the SC NAS System Com-puter System that hosts this LogicalD-isk.
CreationClassName		string	CIM Class of this instance of LogicalD-isk.
DeviceID		string	Opaque identifier for the LogicalDisk.
OperationalStatus		uint16[]	A subset of operational status that is applicable for LogicalDisks in a SC NAS System.
ExtentStatus		uint16[]	This LogicalDisk is neither imported (16) nor exported (17).
Primordial		boolean	This represents a Concrete Logical Disk that is not primordial.
NameFormat		uint16	The format of the Name appropriate for LogicalDisks in the Self-contained NAS System. This should be coded as 1 (other).

Table 930: SMI Referenced Properties/Methods for CIM_LogicalDisk (Disk for FS)

Property	Flags	Type	Description & Notes
Name		string	Identifier for a local LogicalDisk that will be used for a filesystem; since this storage extent will be referenced by a client, it needs to have a unique name. We cannot constrain the format here, but the OS-specific format described in the Block Services specification is not appropriate, so "Other" is used.

8.2.8.5.9.22 CIM_LogicalFile

The Self-Contained NAS Profile only makes a limited set of LogicalFiles (or Directory subclass) instances visible. These are any file or directory that is exported as a share.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 931: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	CIM Class of the Fileserver Computer System that hosts the Filesystem of this File.
CSName		string	Name of the Fileserver Computer System that hosts the Filesystem of this File.
FSCreationClassName		string	CIM Class of the LocalFileSystem on the Fileserver Computer System that contains this File.
FSName		string	Name of the LocalFileSystem on the Fileserver Computer System that contains this File.
CreationClassName		string	CIM Class of this instance of LogicalFile.
Name		string	Name of this LogicalFile.

8.2.8.5.9.23 CIM_NetworkPort

Represents the front end logical port that supports access to a local area network.
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 932: SMI Referenced Properties/Methods for CIM_NetworkPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A user-friendly name for this Network adapter that provides a network port.
OperationalStatus		uint16[]	The operational status of the adapter.
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Network Port.
SystemName		string	The name of the Computer System hosting the Network Port.
CreationClassName		string	The CIM Class name of the Network Port.
DeviceID		string	A unique ID for the device (in the context of the hosting System).
PermanentAddress		string	The hard-coded address of this port.
Optional Properties/Methods			
Speed		uint64	
MaxSpeed		uint64	
RequestedSpeed		uint64	
UsageRestriction		uint16	
PortType		uint16	
PortNumber		uint16	A unique number for the adapter in the context of the hosting System).
NetworkAddresses		string[]	An array of network addresses for this port.
LinkTechnology		uint16	1="Other", 2="Ethernet", 3="IB", 4="FC", 5="FDDI", 6="ATM", 7="Token Ring", 8="Frame Relay", 9="Infrared", 10="BlueTooth", 11="Wireless LAN. The link technology supported by this adapter.
OtherLinkTechnology		string	The vendor-specific "Other" link technology supported by this adapter.
FullDuplex		boolean	
AutoSense		boolean	
SupportedMaximumTransmissionUnit		uint64	
ActiveMaximumTransmissionUnit		uint64	

8.2.8.5.9.24 CIM_ProtocolEndPoint

Represents the front-end ProtocolEndpoint used to support NFS and CIFS services.
Created By : External
Modified By : External
Deleted By : External

Class Mandatory: true

Table 933: SMI Referenced Properties/Methods for CIM_ProtocolEndPoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the Protocol Endpoint.
CreationClassName		string	The CIM Class name of the Protocol Endpoint.
Name		string	The unique name of the Protocol End-point.
NameFormat		string	The Format of the Name
OperationalStatus		uint16[]	The operational status of the PEP.
Description		string	This shall be one of the NFS or CIFS protocol endpoints supported by the SC NAS System.
ProtocolIFType		uint16	This represents either NFS=4200 or CIFS=4201. Other protocol types are specified in sub-classes of Protoco-lEndpoint.
Optional Properties/Methods			
RequestedState		uint16	
EnabledState		uint16	
OtherEnabledState		string	
TimeOfLastStateChange		datetime	

8.2.8.5.9.25 CIM_ResidesOnExtent

Represents the association between a local FileSystem and the underlying LogicalDisk that it is built on.
Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: false

Table 934: SMI Referenced Properties/Methods for CIM_ResidesOnExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalElement	The local file system that is built on top of a LogicalDIsk.
Antecedent		CIM_StorageExtent	The LogicalDIsk that underlies a Local-FileSystem.

8.2.8.5.9.26 CIM_SAPAvailableForElement

Represents the association between a ProtocolEndpoint to the shared element that is being accessed through that SAP.
Created By : External
Modified By : Static

Deleted By : External
Class Mandatory: true

Table 935: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The element that is made available through the ProtocolEndpoint. In the SC NAS System, these are FileShares configured for either export or import.
AvailableSAP		CIM_ServiceAccessPoint	The ProtocolEndpoint that is available to the FileShare.

8.2.8.5.9.27 CIM_SystemDevice

This association links all **LogicalDevices**to the scoping system. This is used to represent both front end and back end devices.
Created By : External or StaticExtrinsic(s):
Modified By : ExternalExtrinsic(s):
Deleted By : ExternalExtrinsic(s):
Class Mandatory: true

Table 936: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that contains this device.
PartComponent		CIM_LogicalDevice	The logical device that is a part of a computer system. These include StorageVolumes, NetworkPorts, 'back end' ports for accessing storage, StorageExtents, protocol controllers, and so on.

8.2.8.5.9.28 CIM_TCPProtocolEndpoint

Represents the front-end ProtocolEndpoint used to support TCP services.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: false

Table 937: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the TCP Protocol Endpoint.
SystemName		string	The name of the Computer System hosting the TCP Protocol Endpoint.
CreationClassName		string	The CIM Class name of the TCP Protocol Endpoint.

Table 937: SMI Referenced Properties/Methods for CIM_TCPProtocolEndpoint

Property	Flags	Type	Description & Notes
Name		string	The unique name of the TCP Protocol Endpoint.
NameFormat		string	The Format of the Name of the TCP Protocol Endpoint.
ProtocolIFType		uint16	4111="TCP". (Note that no other protocol type is supported by this endpoint.)
PortNumber		uint32	The number of the TCP Port that this element represents.

8.2.8.5.10 Related Standards

Table 938: Related Standards for Self-contained NAS System

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.8.6 Filesystem Manipulation Subprofile

8.2.8.6.1 Description

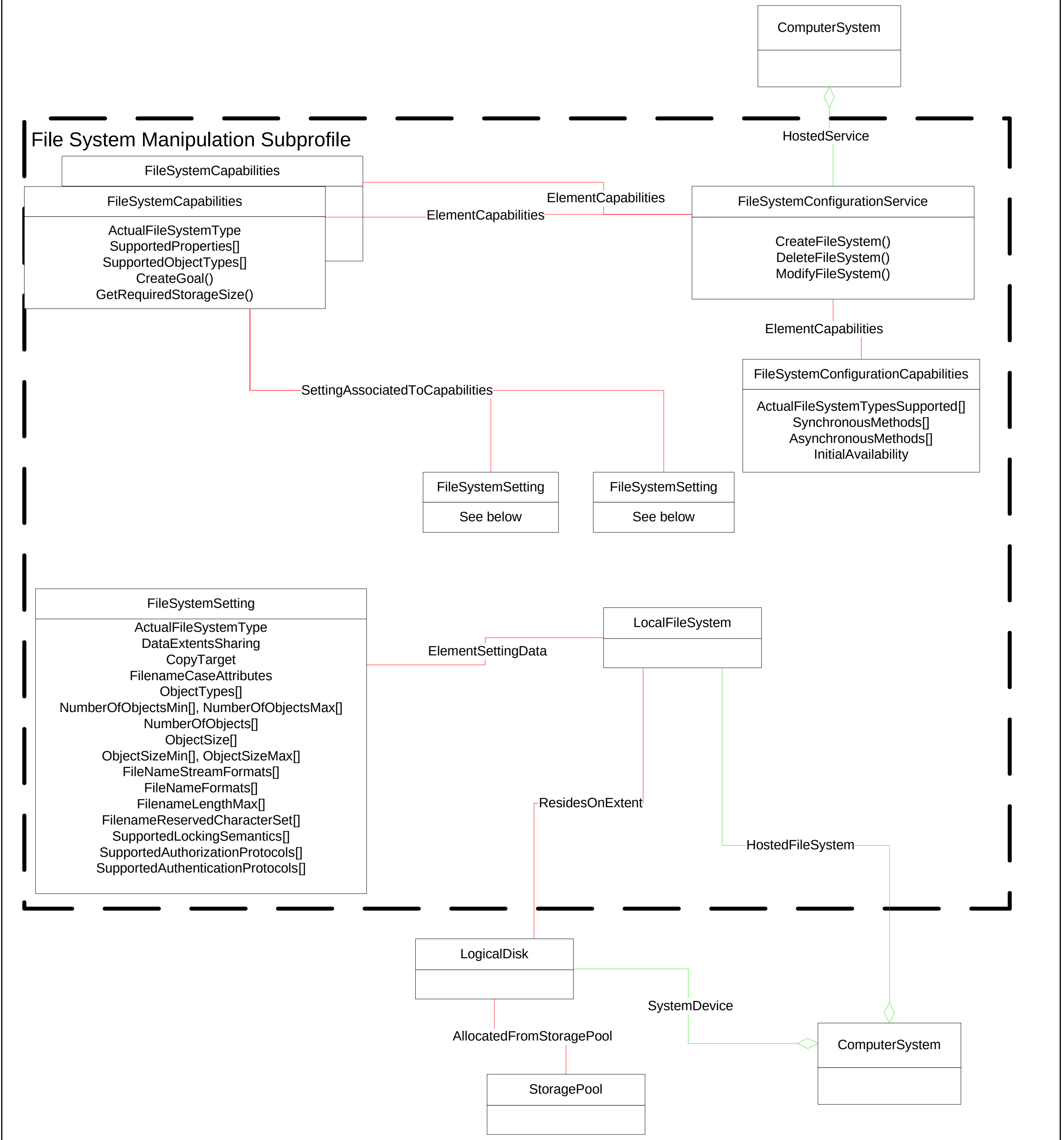
This subprofile provides support for configuring and manipulating filesystems in the context of a NAS profile.

8.2.8.6.1.1 Instance Diagrams

FileSystem Creation classes and associations

Figure 135: "LocalFileSystem Creation Instance Diagram" illustrate the constructs involved with creating a LocalFileSystem for NAS. This summarizes the mandatory classes and associations for this subprofile. Specific areas will be discussed in later sections.

Figure 135: LocalFileSystem Creation Instance Diagram



If a NAS Profile supports the Filesystem Manipulation Subprofile, it will have at least one instance of the FilesystemConfigurationService. This service will be hosted on either the top level ComputerSystem of the NAS or one of the component ComputerSystems. The services offered are CreateFileSystem, ModifyFileSystem and DeleteFileSystem.

Associated to the FilesystemConfigurationService (via ElementCapabilities) will be one instance of FilesystemConfigurationCapabilities. This instance describes the capabilities of the service. It will identify the methods supported, whether they support Job Control or not, the types of filesystems that can be created and whether or not the filesystem is made available after creation.

For each type of filesystem that can be created, there will be one FilesystemCapabilities instance that define the range of capabilities supported for that particular filesystem type. One of these instances will also be identified as a default capability (via DefaultElementCapabilities). This indicates the default filesystem type (if the client does not care).

For the convenience of clients a NAS profile may also populate a set of “pre-defined” FileSystemSettings for each of the FilesystemCapabilities. These will be associated to the FilesystemCapabilities via the SettingAssociatedToCapabilities association.

The FilesystemCapabilities instance also supports two methods: CreateGoal and GetRequiredStorageSize. These methods are described in detail in 8.2.8.6.5.2, "Intrinsic Methods of the Profile", but their basic function is establishing a client defined FilesystemSetting (goal) and determining the LogicalDisk size required to support the desired filesystem.

CreateGoal takes an embedded FilesystemSetting structure as input and generates a valid embedded FilesystemSetting structure. If a client supplies NULL input to this method, the returned FilesystemSetting structure will be a default setting for the ActualFilesystemType of the FilesystemCapabilities. If the input embedded FilesystemSetting is not null, the method will return a “best fit” with the requested setting. The client may iterate on this method until it acquires a setting that suits its needs. It will use this embedded settings structure when it invokes the CreateFileSystem method.

The next step is to determine the LogicalDisk size required to support the FilesystemSettting structure. This is done by invoking the GetRequiredStorageSize method. The inputs are the FilesystemSetting structure and a StorageSetting that describes the quality of service the client wants for the storage (e.g., data redundancy, package redundancy, etc.). The method returns three numbers: The expected size, the minimum size and a maximum usable size. The client would use these numbers in selecting the appropriate LogicalDisk on which to create the Filesystem.

Armed with the Filesystem goal (embedded FilesystemSetting structure) and a LogicalDisk, the client can now create the filesystem. It uses the CreateFileSystem method to do this. Creation of the Filesystem creates a LocalFileSystem and a FilesystemSetting instance. Once a FileSystem is created several associations are created as a side effect of the method. These associations are:

HostedFilesystem to associate the filesystem to the ComputerSystem that hosts it

ResidesOnExtent to associate the filesystem to the extent that holds the filesystem data

ElementSettingData to associate the filesystem to the settings defined for it

In addition to the CreateFileSystem method there are methods for deleting a filesystem and modifying a filesystem. Deleting a file straightforward. It deletes the LocalFileSystem and the FilesystemSetting and the associations to those instances (HostedFileSystem, ElementSettingData and ResidesOnExtent). It does not, however, delete the LogicalDisk. The LogicalDisk becomes available for use in another CreateFilesystem operation.

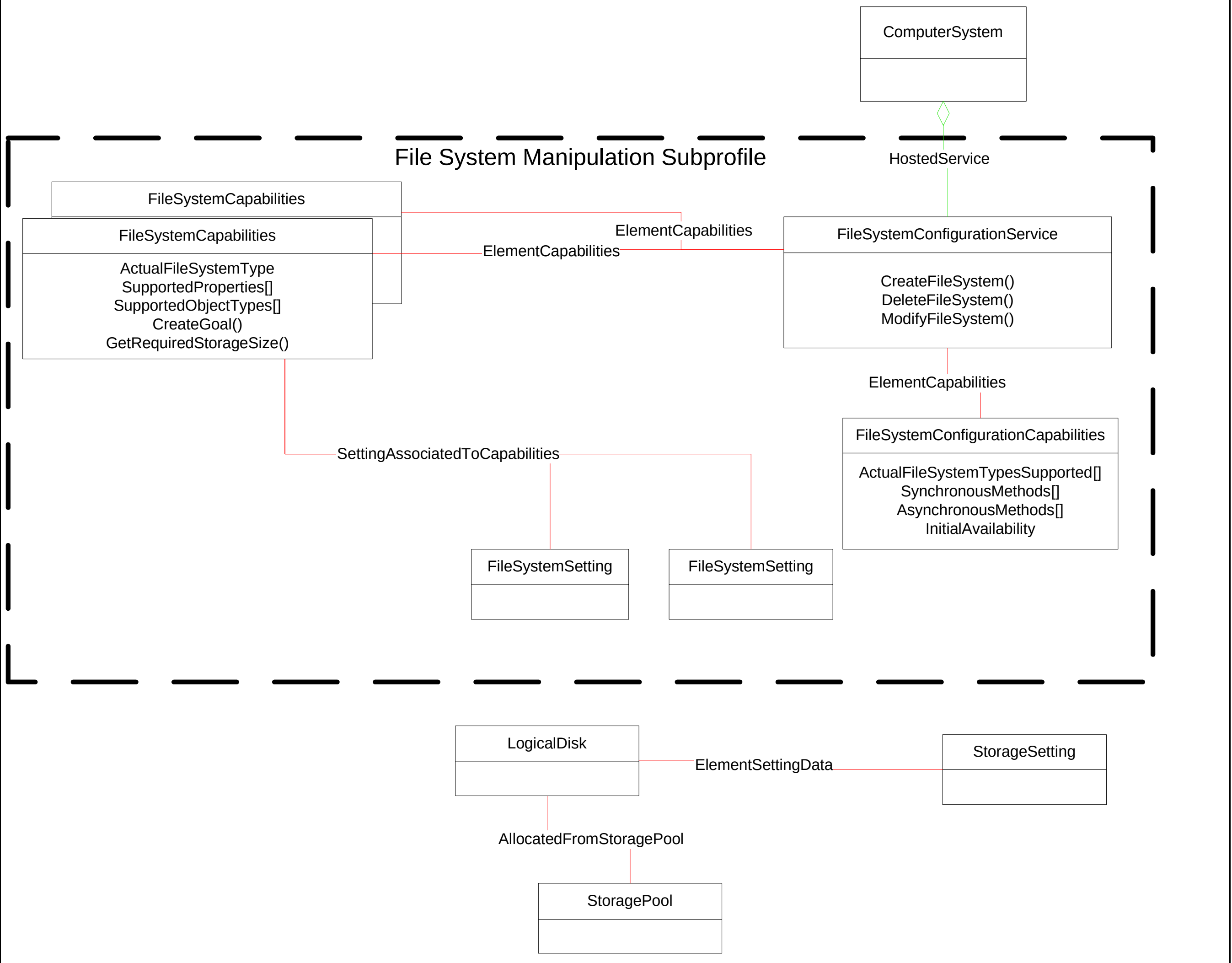
Modifying a Filesystem requires that the client return to the FilesystemCapabilities that was originally used to create the filesystem. Once the FilesystemCapabilities are found, the client would create the FilesystemGoal (embedded structure) desired and invoke the CreateGoal method. As with the original filesystem creation, it may be necessary to iterate on the CreateGoal. Once a desired goal is acquired, the client would issue the ModifyFileSystem.

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

Finding Filesystem Configuration Services, Capabilities and Pre-defined Settings

When creating a filesystem the first step is to determine what can be created. Figure 136:, "Capabilities and Settings for Filesystem Creation Diagram" illustrates an instance diagram showing the instances that will exist for supporting filesystem creation.

Figure 136: Capabilities and Settings for Filesystem Creation Diagram



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

Note: If no service is found from the Top Level ComputerSystem, the client should look for component computer systems that may be hosting the service.

Once the service is found an instance of the FilesystemConfigurationCapabilities shall be associated to the service via the ElementCapabilities association. A client should follow this association (filtering on FilesystemConfigurationCapabilities) to inspect the configuration capabilities that are supported. One property that should to be considered is the ActualFilesystemTypesSupported array. The client would decide which of these filesystem types it will want to create.

To determine the particular capabilities of the ActualFileSystemType, one FilesystemCapabilities for each ActualFilesystemType can be found associated to the FilesystemConfigurationService using the ElementCapabilities association (filtering on the result of FilesystemCapabilities). This capabilities instance will identify the range of properties values supported for the ActualFilesystemType in question.

In addition, an implementation may implement a set of pre-defined FilesystemSettings that may be used by clients to assist in establishing settings desired by the client. If any of these are established they can be found by traversing the SettingsAssociatedToCapabilities association.

8.2.8.6.2 Health and Fault Management Considerations

Under Consideration for a future standard.

8.2.8.6.3 Cascading Considerations

Under Consideration for a future standard.

8.2.8.6.4 Supported Subprofiles and Packages

Table 939: Supported Subprofiles for Filesystem Manipulation

Registered Subprofile Names	Mandatory	Version
Job Control	No	1.1.0

8.2.8.6.5 Methods of the Profile

8.2.8.6.5.1 Extrinsic Methods of the Profile

Table 940: Filesystem Manipulation Methods that cause Instance Creation, Deletion or Modification

Method	Created Instances	Deleted Instances	Modified Instances
CreateFileSystem	LocalFileSystem FileSystemSetting ElementSettingData ResidesOnExtent HostedFilesystem	N/A	N/A
DeleteFileSystem		LocalFileSystem FileSystemSetting ElementSettingData ResidesOnExtent HostedFilesystem	N/A
ModifyFileSystem	N/A	N/A	FileSystemSetting
CreateGoal	N/A	N/A	N/A
GetRequiredStorage-Size	N/A	N/A	N/A

8.2.8.6.5.1.1 CreateFileSystem

Start a job to create a filesystem on a LogicalDisk. If the operation completes successfully and did not require a long-running ConcreteJob, it will return 0. If 4096/0x1000 is returned, a ConcreteJob will be started to create the element. A Reference to the ConcreteJob will be returned in the output parameter Job. If any other value is returned, the job will not be started, and no action will be taken. This method shall return a Error representing that a single named property of a setting (or other) parameter (either reference or embedded object) has an invalid value or that an invalid combination of named properties of a setting (or other) parameter (either reference or embedded “object) has been requested.

The parameter TheElement will contain a Reference to the filesystem if this operation completed successfully.

The LogicalDisk to use is specified by the InExtent parameter. If this is NULL, a default LogicalDisk will be created in a vendor-specific way and used. One way to create the default LogicalDisk is to use one of the default settings supported by the StorageConfigurationService hosted by the host hosting the FileSystemConfigurationService.

The desired settings for the filesystem are specified by the Goal parameter. Goal is an embedded object of class FileSystemSetting or a derived class, encoded as a string-valued embedded object parameter; this allows the client to specify the properties desired for the filesystem. The Goal parameter includes information that can be used by the vendor to compute the size of the Filesystem. If the LogicalDisk specified here cannot support the goal size, an appropriate error value will be returned, and no action will be taken.

A ResidesOnExtent association is created between the Filesystem and the InExtent.

CreateFileSystem(

[IN, Description (A end user relevant name for the filesystem being created. If NULL, then a system-supplied default name can be used. The value will be stored in the 'ElementName' property for the created element.)]

string **ElementName**,

[OUT, IN (false), Description(Reference to the job (may be null if job completed).")]

CIM_ConcreteJob REF **Job**,

[IN, EmbeddedInstance `("CIM_FileSystemSetting")`, Description(The requirements for the Filesystem element to maintain. This is an element of class CIM_FileSystemSetting, or a derived class, encoded as a string-valued embedded instance parameter; this allows the client to specify the properties desired for the filesystem. If NULL or the empty string, the default configuration will be specified by the FileSystemConfigurationService.)]

string **Goal**,

[IN, Description(The LogicalDisk on which the created FileSystem will reside. If this is NULL, a default LogicalDisk will be created in a vendor-specific way and used. One way to create the default LogicalDisk is to use one of the default settings supported by the StorageConfigurationService hosted by the host hosting the FileSystemConfigurationService.)]

CIM_StorageExtent REF **InExtent**,

[IN, OUT, Description (The newly created FileSystem.)]

CIM_LogicalElement REF **TheElement**

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "StorageExtent is not big enough to satisfy the request.", "StorageExtent specified by default cannot be created.", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}

8.2.8.6.5.1.2 ModifyFileSystem

Start a job to modify a previously created FileSystem. If the operation completes successfully and did not require a long-running ConcreteJob, it will return 0. If 4096/0x1000 is returned, a ConcreteJob will be started to modify the element. A Reference to the ConcreteJob will be returned in the output parameter Job. If any other value is returned, either the job will not be started, or if started, no action will be taken.

This method shall return a Error representing that a single named property of a setting (or other) parameter (either reference or embedded object) has an invalid value or that an invalid combination of named properties of a setting (or other) parameter (either reference or embedded object) has been requested.

The parameter TheElement specifies the FileSystem to be modified. This element shall have been created by this FileSystemConfigurationService.

The desired settings for the FileSystem are specified by the Goal parameter. Goal is an element of class FileSystemSetting, or a derived class, encoded as a string-valued embedded instance parameter; this allows the client to specify the properties desired for the filesystem. The Goal parameter includes information that can be used by the vendor to compute the size of the FileSystem. If the operation would result in a change in the size of a filesystem, the ResidesOnExtent association will be used to determine how to implement the change. If the LogicalDisk specified cannot support the goal size, an appropriate error value will be returned, and no action will be taken. If the operation succeeds, the ResidesOnExtent association might reference a different LogicalDisk.

ModifyFileSystem(

[IN, Description (A end user relevant name for the FileSystem being modified. If NULL, then the name will not be changed. If not NULL, this parameter will supply a new name for the FileSystem element.)]

string **ElementName**,

[OUT, IN (false), Description(Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, EmbeddedInstance ("CIM_FileSystemSetting"), Description(The requirements for the FileSystem element to maintain. This is an element of class FileSystemSetting, or a derived class, encoded as a string-valued embedded instance parameter; this allows the client to specify the properties desired for the filesystem. If NULL or the empty string, the FileSystem service attributes will not be changed. If not NULL, this parameter will supply new settings that replace or are merged with the current settings of the FileSystem element.)]

string **Goal**,

[IN, Description (The FileSystem element to modify.)]

CIM_LogicalElement REF **TheElement**,

[IN, Description (An enumerated integer that specifies the action to take if the FileSystem is still in use when this request is made. This option is only relevant if the FileSystem needs to be made unavailable while the request is being executed.), ValueMap { "2", "3", "4", "..", "0x1000..0xFFFF" },

Values { "Do Not Execute Request", "Wait for specified time, then Execute Request

Immediately", "Try to Quiesce for specified time, then Execute Request Immediately",

"DMTF Reserved", "Vendor Defined" }]

uint16 **InUseOptions**,,

[IN, Description (An integer that indicates the time in seconds to wait before performing the request on this FileSystem. The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request.)]

uint16 **WaitTime**);

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "FileSystem In Use, cannot Modify", "Cannot satisfy new Goal.", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}

8.2.8.6.5.1.3 DeleteFileSystem

Start a job to delete a FileSystem. If the FileSystem cannot be deleted, no action will be taken, and the Return Value will be 4097/0x1001. If the method completed successfully and did not require a long-running ConcreteJob, it will return 0. If 4096/0x1000 is returned, a ConcreteJob will be started to delete the FileSystem. A Reference to the ConcreteJob will be returned in the output parameter Job.

The ClearStorage parameter, if 'true', directs that the underlying storage should be cleaned. Note that if this is not done, the filesystem may be recoverable.

DeleteFileSystem(

[OUT, IN (false), Description (Reference to the job (may be null if job completed).)]

    CIM_ConcreteJob REF **Job**,

[IN, Description ("An element or association that uniquely identifies the FileSystem to be deleted.)]

    CIM_ManagedElement REF **TheFileSystem**,

[IN, Description (An enumerated integer that specifies the action to take if the FileSystem is still in use when this request is made.),

ValueMap{ "2", "3", "4", "..","0x1000..0xFFFF" },

Values {"Do Not Delete", "Wait for specified time, then Delete Immediately", "Attempt Quiescence for specified time, then Delete Immediately", "DMTF Reserved", "Vendor Defined" }]

    uint16 **InUseOptions**,

[IN, Description (An integer that indicates the time in seconds to wait before deleting this FileSystem. The combination of InUseOptions = '3' and WaitTime ='0' "(the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request.)]

    uint32 **WaitTime**);

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed, Unspecified Reasons", "Invalid Parameter", "FileSystem in use, Failed", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}

8.2.8.6.5.1.4 CreateGoal

Start a job to create a supported FileSystemSetting from a FileSystemSetting provided by the caller. If the operation completes successfully and did not require a long-running ConcreteJob, it will return 0. If 4096/0x1000 is returned, a ConcreteJob will be started to create the element. A Reference to the ConcreteJob will be returned in the output parameter Job.

This method may return a Error representing that a single named property of a setting (or other) parameter (either reference or embedded object) has an invalid value or that an invalid combination of named properties of a setting (or other) parameter (either reference or embedded object) has been requested.

If the input TemplateGoal is NULL or the empty string, this method returns a default FileSystemSetting that is supported by this FileSystemCapabilities.

The output is returned as the SupportedGoal parameter. Both TemplateGoal and SupportedGoal are embedded objects and do not exist in the provider but are maintained by the client.

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

CreateGoal(

[OUT, IN (false), Description (Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, EmbeddedInstance("CIM_FileSystemSetting"), Description (TemplateGoal is an element of class FileSystemSetting, or a derived class, encoded as a string-valued embedded object parameter, that is used as the template to be matched.)]

string **TemplateGoal**,

[OUT, IN(false), EmbeddedInstance("CIM_FileSystemSetting"), Description (SupportedGoal is an element of class CIM_FileSystemSetting, or a derived class, encoded as a string-valued embedded object parameter, that is used to return the best supported match to the TemplateGoal.)]

string **SupportedGoal**);

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "TemplateGoal is not well-formed", "TemplateGoal cannot be satisfied exactly", "StorageSetting cannot be used with ActualFileSystemType", "StorageSetting cannot be used with CopyTarget", "StorageSetting cannot be used with ObjectType", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}]

8.2.8.6.5.1.5 GetRequiredStorageSize

This method returns the "expected" size of a LogicalDisk that would support a filesystemfilesystem specified by the input FileSystemGoal parameter assuming that the other settings for the LogicalDisk are specified by the ExtentSetting parameter.

If the input FileSystemGoal or the ExtentSetting are NULL, this method returns a value computed by using the default FileSystemSetting or some vendor-specific canned StorageSetting.

A value of 0 is returned if this method is not able to determine a reasonable size or does not restrict sizes based on setting information.

GetRequiredStorageSize(

[IN, EmbeddedInstance("CIM_FileSystemSetting"), Description (FileSystemGoal is an element of class CIM_FileSystemSetting, or a derived class, encoded as a string-valued embedded object parameter, that is used to specify the settings for the FileSystem to be created.)]

string **FileSystemGoal**,

[IN, EmbeddedInstance("CIM_FileSystemSetting"), Description (ExtentSetting is an element of class CIM_StorageSetting, or a derived class, encoded as a string-valued embedded object parameter, that is used to specify the settings of the LogicalDisk to be used for this FileSystem.)]

CIM_StorageSetting REF **ExtentSetting**,

[OUT, Description (A number that indicates the size of the storage extent that this FileSystem is expected to need. A value of 0 indicates that there is no expected size.)]

uint64 **ExpectedSize**,

[OUT, Description (A number that indicates the size of the smallest storage extent that would support the specified FileSystem. A value of 0 indicates that there is no minimum size.)]

uint64 **MinimumSizeAcceptable**,

[OUT, Description (A number that indicates the size of the largest storage extent that would be usable for the specified FileSystem. A value of 0 indicates that there is no maximum size.)]

uint64 **MaximumSizeUsable**);

8.2.8.6.5.2 Intrinsic Methods of the Profile

None.

8.2.8.6.6 Client Considerations and Recipes

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

```
sub GetFSSetting(IN REF CIM_FileSystem $fs,
                 OUT CIM_FileSystemSettingData $setting);

sub GetFSServer(IN REF CIM_FileSystem $fs,
                 OUT CIM_ComputerSystem $system);

sub GetFSCapabilityFromServer(IN REF CIM_System $server,
                              OUT CIM_FileSystemConfigurationServiceCapabilities
$capability,
                              OUT CIM_FileSystemConfigurationService
$fsconfigurator,
                              IN Optional String $filesystemtype = "",
                              IN Optional String $otherpropertyname = NULL,
                              IN Optional String $otherpropertyvalue = NULL);

sub GetFSCapabilityFromFileSystem(IN REF CIM_FileSystem $fs,
                   OUT CIM_FileSystemConfigurationServiceCapabilities $capability,
                    OUT CIM_FileSystemConfigurationService $fsconfigurator);

sub GetExportServiceAndCapabilities(IN REF CIM_FileSystem $fs,
                                    IN String $sharetype,
                                    OUT CIM_FileExportService $feservice,
                                    OUT CIM_ExportedFileShareCapabilities
$efscapability);
```

Conventions used in the NAS recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is used without further validation. Real code will need to be more robust.
- We use Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.

8.2.8.6.6.1 Get the FileSystemSettings associated with a FileSystem

```
//
// Get the FileSystemSettings associated with a FileSystem
//
sub GetFSSetting(IN REF CIM_FileSystem $fs,
                 OUT CIM_FileSystemSettingData  $setting)
{
    //
    // Get a client-side copy of the FileSystemSetting associated with the
    // FileSystem (via ElementSettingData association)
    //
    // In this and other NAS recipes we “cheat” and assume there is one
    // setting in the returned list
    //
    $setting = Associators($fs,
                           “CIM_ElementSettingData”,
                           “CIM_FileSystemSettingData”,
                           “ManagedElement”,
                           “SettingData”)->[0];
}
```

8.2.8.6.6.2 Creation of a Filesystem on a Storage Extent

```
//
// Create a Filesystem on a StorageExtent/LogicalDisk
//

//
// Note: A CIM_LogicalDisk ISA CIM_StorageExtent, see above in conventions
//
sub CreateFileSystem(IN CIM_System $server,
                     IN CIM_LogicalDisk $disk,
                     IN uint64 $desiredsize,
                     IN String $fsname,
                     IN String $filesystemtype,
                    IN String $otherpropertyname[],    // array of property names
                     IN String $otherpropertyvalue[],   // corresponding array of
values
                     OUT CIM_FileSystem $fs,
                     OUT CIM_Job $job)
{
    //
    // Use a subroutine to get a capability from the server.
    //
    &GetFSCapabilityFromServer($server, $filesystemtype,
                              $capability, $fsconfigurator, $filesystemtype);
    if ($capability == NULL) {
        <raise an error>
        $fs = NULL;
        return;
    }

    //
    // Call FSCSCapabilities.CreateGoal(nullTemplate, Goal) to get a seed
    // goal for FSSetting, or just use one of the provided default settings
    // associated with the FSCSCapabilities via AssociatedSetting and
    // DefaultAssociatedSetting.
    //
    $fssgoal = NULL;
    $capability.CreateGoal(NULL, $fssgoal);

    //
    // Inspect Goal and modify properties as desired.
    //
    $fssgoal.ActualFileSystemType = $filesystemtype;
    #i = 0;
    while ($otherpropertyname[#i]) {
        $fssgoal.$otherpropertyname[#i] = $otherpropertyvalue[#i];
        #i++;
    }

    //
    // Call FSCSCapabilities.CreateGoal(Goal-N', Goal-N) to get the next
    // goal for FSSetting -- iterate until satisfied or give up (beware
    // infinite loops)  Note: we don't iterate here, just give up if we
    // don't get what we want.
    //
    $capability.CreateGoal($fssgoal, $fssgoal2);

    #i = 0;
    while ($otherpropertyname[#i]) {
        //
```

```
            // Note: this pseudocode doesn’t check to see if the property named
            // in $otherpropertyname[#i] is an array.  This additional level
            // of horsing around is left as an exercise for the reader.
            //
            if ($fssgoal.$otherpropertyname[#i] != $otherpropertyvalue[#i] {
                { return NULL; }          // give up
            }
        }

        //
        // Call FSCSCapabilities.GetRequiredStorageSize(Goal,
        // DesiredUsableCapacity) to find out how large of a
        // LogicalDisk is needed.
        //
        // (GetRequiredStorageSize also returns the maximum usable
        // size of the disk, given the settings expressed in Goal,
        // which is useful if the disk being used can’t be grown
        // upon demand)
        //
        $requiredsize = $capability.GetRequiredStorageSize($fssgoal2,
                                      NULL,             // no special requirements
                                      $expectedsize,
                                      $minsize,
                                      $maxsize);

        //
        // If a disk of the required size is already available
        //     Call CreateFileSystem(Goal, LogicalDisk)
        // else
        //     Create LogicalDisk (see StorageExtent recipes)
        //     Call CreateFileSystem(Goal, LogicalDisk)
        //
        if ($requiredsize > $disk.BlockSize * $disk.NumberOfBlocks) {
            <CreateDisk>($requiredsize, $newdisk);
            $disk = $newdisk;
        }

        $fsconfigurator.CreateFileSystem($fsname, $job, $fssgoal2, $disk, $fs);
        //
        // not shown: managing the $job if it’s not NULL, and managing any
        // CIM_Errors that get sent.
        //
        return $fs;
    }
```

8.2.8.6.6.3 Increase the size of a FileSystem

```
    //
```

```
// Increase the size of a FileSystem
//
sub IncreaseFileSystemSize(IN CIM_FileSystem $fs,
                           IN uint64 $desiredsize,
                           OUT CIM_Job $job)
{
    //
    // Get a client-side copy of the FileSystemSetting associated with the
    // FileSystem
    //
    $fssnewgoal = Associators($fs,
                             “CIM_ElementSettingData”,
                             “CIM_FileSystemSettingData”,
                             “ManagedElement”,
                             “SettingData”)->[0];

    //
    // Use a subroutine to get a capability from the server.
    //
    &GetFSCapabilityFromFilesystem($fs, $filesystemtype, $capability);
    if ($capability == NULL) {
        <raise an error>
        return;
    }

    //
    // Get the FileSystemConfiguration Service of NAS server using
    // a HostedService association
    //
    $fsconfigurator = Associators($server->,
                                  “CIM_HostedService”,
                                  “CIM_FileSystemConfigurationService”,
                                  “Antecedent”,
                                  “Dependent”)->[0];
    if ($fsconfigurator == NULL) {
        <raise an error>
        return;
    }

    //
    // Call FSCSCapabilities.GetRequiredStorageSize(NewGoal,
    // DesiredUsableCapacity) to find out how large of a
    // LogicalDisk is needed
    //
    $requiredsize = $capability.GetRequiredStorageSize($fssnewgoal,
                                  NULL,             // no special requirements
                                  $desiredsize,
                                  $minsize,
                                  $maxsize);

    //
    // Get Underlying SE using ResidesOnExtent association
    //
    $disk = Associators($fs,
                     “CIM_ResidesOnExtent”,
                     “CIM_LogicalDisk”,
                     “Dependent”,
                     “Antecedent”)->[0];

    //
    // If disk is not large enough, increase size of underlying SE
    //
    $job = NULL;
    if ($requiredsize < $disk.BlockSize * $disk.NumberOfBlocks) {
         <increase size of logical disk, returning a job in $job if
                           necessary -- see storage extent recipes>
    }

    //
    // The filesystem itself doesn’t need modification, so we’re done
    //
}
```

8.2.8.6.6.4 Modify a Filesystem’s Settings

```
//
// Modify a FileSystem’s settings
//
// Rather than attempt a general-purpose settings modification
// recipe, we will simply twiddle a couple settings.  The concept
// extends easily to other Settings attributes, or you can use
// the concept of corresponding arrays to pass in arbitrary settings,
// as shown in the CreateFileSystem recipe.
//
sub ModifyFileSystemObjectLimits(IN CIM_FileSystem $fs,
                                 IN OUT uint64 $objecttype,
                                 IN OUT uint64 $minobjects,
                                 IN OUT uint64 $maxobjects,
                                 IN OUT uint64 $normnobjects,
                                 OUT CIM_Job $job)
{
    //
    // Get a client-side copy of the CIM_FileSystemSettingData associated with the
    // FileSystem (via ElementSettingData association) using GetInstance
    //
```

```
$fssnewgoal = Associators($fs,
                         “CIM_ElementSettingData”,
                         “CIM_FileSystemSettingData”,
                         “ManagedElement”,
                         “SettingData”)->[0];


//
// Use a subroutine to get a capability from the server.
//
&GetFSCapabilityFromFilesystem($fs, $filesystemtype, $capability);
if ($capability == NULL) {
    <raise an error>
    return;
}

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

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

//
// Call FSCSCapabilities.CreateGoal(Goal-N’, Goal-N) to get the next
// goal for FSSetting -- iterate until satisfied or give up (beware
// infinite loops)  Note: we don’t iterate here, just give up.
//
$capability.CreateGoal($fssnewgoal, $fssgoal2);
if ($fssgoal2.ActualFileSystemType != $filesystemtype)
```

```
            { return NULL; }

        //
        // call ModifyFilesystem (management of $job and any CIM_Error not
        // shown)
        //
        $fsconfigurator.ModifyFileSystem(NULL, $job, $fssgoal2, $fs);

        return $fs;
    }
```

8.2.8.6.6.5 Delete a FileSystem and return underlying StorageExtent

```
    //
    // Delete a FileSystem and return underlying LogicalDisk
    //
    sub DeleteFileSystem(CIM_FileSystem $fs)
    //
    // This only deletes the filesystem instance, not the underlying
    // logical disk, which is returned for further disposition.
    //
    // NOTE: if you want to “wipe” or zero out the filesystem, you
    // must either do that via client-level operations over a
    // FileSystemShare before deleting the filesystem, or by means of
    // vendor-specific operations on the logical disk subsequent to
    // deleting the filesystem.
    //
    {
        //
        // Get Underlying SE using ResidesOnExtent association
        //
        $disk = Associators($fs,
                              “CIM_ResidesOnExtent”,
                              “CIM_LogicalDisk”,
                              “Dependent”,
                              “Antecedent”)->[0];

        //
        // Get configuration service
        //
        &GetFSCapabilityFromFileSystem($fs, $capability, $fsconfigurator);

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

        //
        // Return REF to SE
        //
        return $disk;
    }
```

8.2.8.6.6.6 Get a FileSystemConfigurationServiceCapabilities from a NASServer

```
    //
    // Get a FileSystemConfigurationServiceCapabilities from a NASServer
    //
    // this routine returns the configurator that matches a capability,
    // as well as the capability that matches the other given parameters
    //
    sub GetFSCapabilityFromServer(IN REF CIM_System $server,
                                  OUT CIM_FileSystemConfigurationServiceCapabilities
    $capability,
                                  OUT CIM_FileSystemConfigurationService
    $fsconfigurator,
                                  IN OPTIONAL String $filesystemtype = ““,
                                  IN OPTIONAL String $otherpropertyname = NULL,
                                  IN OPTIONAL String $otherpropertyvalue = NULL)
    {
        //
        // Get the FileSystemConfiguration Service of NAS server using
        // a HostedService association
        //
        $fsconfigurators->[] = Associators($server,
                                      “CIM_HostedService”,
                                      “CIM_FileSystemConfigurationService”,
                                      “Antecedent”,
                                      “Dependent”);
        #i = 0;
        while ($fsconfigurator = $fsconfigurators->[#i]) {

            //
            // Find FSCapabilities that supports ActualFileSystemType
            // using ElementCapabilities association from FSConfigurationService.
            // If client does not care about the ActualFileSystemType, use default
            // FileSystemConfigurationServiceCapabilities.  NOTE: there may be more
            // than one FSCSCapabilities for a ActualFileSystemType; each
            // FSCSCapabilities may support a different set of properties (say
            // NFS vs CIFS locking), so this needs to be checked for as well.
            //
            $capabilities->[] = Associators($fsconfigurator,
                                            “CIM_ElementCapabilities”,
                                            “CIM_FileSystemCapabilities”,
                                            “ManagedElement”,
                                            “Capabilities”);
            #j = 0;
            while($capability = $capabilities->[#j]) {
                if ($filesystemtype == "" ||
                       $capability.SupportedActualFileSystemType == $filesystemtype) {
                    if (($otherpropertyname == NULL && $otherpropertyvalue == NULL) ||
                              $capability.$otherpropertyname == $otherpropertyvalue) {
                        //
                        // successful return
                        //
                        return;
                    }
                }
                #j++;
            }
        }
        //
        // no luck
        //
        $capability = NULL;
        $fsconfigurator = NULL;
        return;
    }
```

8.2.8.6.6.7 Get a FileSystemConfigurationServiceCapabilities from an existing FileSystem

```
    //
    // Get a FileSystemConfigurationServiceCapabilities from an existing FileSystem
    //
    sub GetFSCapabilityFromFileSystem(IN REF CIM_FileSystem $fs,
                       OUT FileSystemConfigurationServiceCapabilities $capability,
                       OUT CIM_FileSystemConfigurationService $fsconfigurator)
    {
        //
        // Get a client-side copy of the FileSystemSetting associated with the
        // FileSystem
        //
        &GetFSSetting($fs, $fssetting);

        //
        // Get the ActualFileSystemType from the FileSystemSetting
         //
        $fstype = $fssetting.ActualFileSystemType;

        //
        // Get the ComputerSystem for the FS (via HostedFileSystem association)
         //
        $system = Associators($fs,
                              "CIM_HostedFileSystem",
                              “CIM_Filesystem”,
                              “PartComponent”,
                              “GroupComponent”)->[0];

        //
        // Get the FileSystemConfigurationService from the ComputerSystem
         //   via the HostedService association
         //
        $fsconfigurators->[] = Associators($system,
                                      “CIM_HostedService”,
                                      “CIM_FileSystemConfigurationService”,
                                      “Antecedent”,
                                      “Dependent”);
        #i = 0;
        while ($fsconfigurator = $fsconfigurators->[#i]) {

            //
            // Find FSCapabilities that supports ActualFileSystemType
            // using ElementCapabilities association from FSConfigurationService.
            // If client does not care about the ActualFileSystemType, use default
            // FileSystemConfigurationServiceCapabilities.  NOTE: there may be more
            // than one FSCSCapabilities for a ActualFileSystemType; each
            // FSCSCapabilities may support a different set of properties (say
            // NFS vs CIFS locking), so this needs to be checked for as well.
            //
            $capabilities->[] = Associators($fsconfigurator,
                                            “CIM_ElementCapabilities”,
                                            “CIM_FileSystemCapabilities”,
                                            “ManagedElement”,
                                            “Capabilities”);
            }
            #j = 0;
            while($capability = $capabilities->[#j]) {
                if ($filesystemtype == ““ ||
                        $capability.SupportedActualFileSystemType == $fstype) {
                    //
                    // successful return
                    //
                    return;
                }
                #j++;
            }
        }
        //
        // no luck
        //
        $capability = NULL;
```

```
        $fsconfigurator = NULL;
        return;
    }
```

8.2.8.6.6.8 Filesystem Manipulation Supported Capabilities Patterns

Table 941, “Filesystem Manipulation Supported Capabilities Patterns” lists the patterns that are formally recognized by this version of SMI-S for determining capabilities of various NAS implementations:

Table 941: Filesystem Manipulation Supported Capabilities Patterns

SupportedActualFileSystem Types	Supported SynchronousMethods	Supported SynchronousMethods	InitialAvailablity
Any	none	none	none
Any	CreateFileSystem, DeleteFileSystem, ModifyFileSystem, CreateGoal, GetRequiredStorageSize	none	Any
Any	none	CreateFileSystem, DeleteFileSystem, ModifyFileSystem, CreateGoal	Any

8.2.8.6.7 Registered Name and Version

Filesystem Manipulation version 1.1.0

8.2.8.6.8 CIM Server Requirements

Table 942: CIM Server Requirements for Filesystem Manipulation

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.6.9 CIM Elements

Table 943: CIM Elements for Filesystem Manipulation

Element Name	Description
Mandatory Classes	
CIM_ElementCapabilities (8.2.8.6.9.1)	In this subprofile, associates the Filesystem configuration service to the Filesystem capabilities that it supports.
CIM_ElementCapabilities (8.2.8.6.9.2)	In this subprofile, associates the Filesystem configuration service to the Filesystem configuration capabilities.
CIM_FileStorage (8.2.8.6.9.4)	Associates a Logical File or Directory to the LocalFileSystem that contains it.
CIM_FileSystemCapabilities (8.2.8.6.9.5)	This element represents the Capabilities of the Filesystem configuration service for managing Filesystems. The Service can be associated with multiple Capabilities that are keyed by the ActualFileSystemType property.
CIM_FileSystemConfigurationCapabilities (8.2.8.6.9.6)	This element represents the management capabilities of the Filesystem configuration service.
CIM_FileSystemConfigurationService (8.2.8.6.9.7)	The FileSystemConfigurationService provides the methods to manipulate filesystems.
CIM_HostedFileSystem (8.2.8.6.9.10)	Represents the association between a LocalFileSystem and the Computer System that hosts it.
CIM_HostedService (8.2.8.6.9.11)	In this subprofile, associates the FileSystemConfigurationService to the hosting Computer System.
CIM_LocalFileSystem (8.2.8.6.9.12)	Represents a LocalFileSystem that a Computer System could make available.
CIM_LogicalFile (8.2.8.6.9.14)	A LogicalFile (or Directory subclass) is used to provide mountpoints on a Computer System for LocalFIleSystems.
CIM_ResidesOnExtent (8.2.8.6.9.15)	Represents the association between a local FileSystem and the underlying StorageExtent/LogicalDisk that it is built on.
Optional Classes	
CIM_ElementSettingData (8.2.8.6.9.3)	Associates a configuration setting to the configured element. It is used in this subprofile with LocalFileSystem and FileSystemSetting elements.
CIM_FileSystemSetting (8.2.8.6.9.8)	This element represents sample configuration settings of a Filesystem. It represents "pre-defined" settings supported by a Filesystem configuration service. A FileSystemSetting element specifies a single ActualFileSystemType which is a weak key with respect to the FileSystemConfigurationService.
CIM_FileSystemSetting (8.2.8.6.9.9)	This element represents the configuration settings of a Filesystem. It gets created by theCreateFileSystem extrinsic method when the CIM_LocalFileSystem is created.
CIM_LocalFileSystem (8.2.8.6.9.13)	The represents LocalFileSystems that are hosted by the NAS Head.

Table 943: CIM Elements for Filesystem Manipulation

Element Name	Description
CIM_SettingAssociatedToCapabilities (8.2.8.6.9.16)	Represents the association between a FilesystemCapabilities and a supported FileSystemSetting element.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LocalFileSystem	CQL - Creation of a LocalFileSystem element.
SELECT OBJECTPATH(SourceInstance)AS FSPath, SourceInstance.Name FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LocalFileSystem	CQL - Deletion of a LocalFileSystem element.
Optional Indications	
SELECT OBJECTPATH(IC.SourceInstance)AS FSPath, IC.Name, IC.FileSystemSize, IC.AvailableSpace, IC.FileSystemType, CS.Name, CS.NameFormat, CS.OtherIdentifyingInfo, CS.IdentifyingDescriptions FROM CIM_InstCreation IC, CIM_HostedFileSystem HFS, CIM_ComputerSystem CS WHERE SourceInstance ISA CIM_LocalFileSystem AND OBJECTPATH(CS) = A.GroupComponent AND OBJECTPATH(IC.SourceInstance) = A.PartComponent	CQL - Creation of a LocalFileSystem element. In addition to returning important properties of LocalFileSystem, it also returns the name of the ComputerSystem it is hosted on.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LocalFileSystem	CQL - Modification of a LocalFileSystem element

8.2.8.6.9.1 CIM_ElementCapabilities

In this subprofile, associates the Filesystem configuration service to the Filesystem capabilities that it supports.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 944: SMI Referenced Properties/Methods for CIM_ElementCapabilities (FS Capabilities)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The Filesystem configuration service.
Capabilities		CIM_Capabilities	The FileSystemCapabilties.

8.2.8.6.9.2 CIM_ElementCapabilities

In this subprofile, associates the Filesystem configuration service to the Filesystem configuration capabilities.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 945: SMI Referenced Properties/Methods for CIM_ElementCapabilities (FS Config Capabilities)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The Filesystem configuration service.
Capabilities		CIM_Capabilities	The Filesystem configuration capabilities.

8.2.8.6.9.3 CIM_ElementSettingData

Associates a configuration setting to the configured element. It is used in this subprofile with LocalFileSystem and FileSystemSetting elements.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Static
Deleted By : Extrinsic(s): DeleteFileSystem
Class Mandatory: false

Table 946: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The LocalFileSystem.
SettingData		CIM_SettingData	The current configuration of the LocalFileSystem.

8.2.8.6.9.4 CIM_FileStorage

Associates a Logical File or Directory to the LocalFileSystem that contains it.
Created By : Extrinsic(s): CreateFileSystem or ModifyFileSystem
Modified By : Static
Deleted By : Extrinsic(s): DeleteFileSystem or ModifyFileSystem
Class Mandatory: true

Table 947: SMI Referenced Properties/Methods for CIM_FileStorage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_FileSystem	The LocalFileSystem that contains the LogicalFile.
PartComponent		CIM_LogicalFile	The LogicalFile contained in the LocalFileSystem.

8.2.8.6.9.5 CIM_FileSystemCapabilities

This element represents the Capabilities of the Filesystem configuration service for managing Filesystems. The Service can be associated with multiple Capabilities that are keyed by the ActualFileSystemType property.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 948: SMI Referenced Properties/Methods for CIM_FileSystemCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a capability of a Filesystem configuration service.
ElementName		string	A User-Friendly Name for this Capabilities element.
ActualFileSystemType		uint16	This identifies the type of filesystem that this Capabilities represents.
SupportedProperties		uint16[]	This is the list of configuration properties (of FileSystemSetting) that are supported for specification at creation time by this Capabilities element.
CreateGoal()			This extrinsic method supports the creation of a FileSystemSetting that is a supported variant of a FileSystemSetting passed in as an embedded IN parameter. The method returns the supported FileSystemSetting as an embedded OUT parameter.
Optional Properties/Methods			
GetRequiredStorageSize()			This extrinsic method supports determining the storage space requirements for a filesystem specified by the combination of a FileSystemSetting and a StorageSetting. The StorageSetting requires redundancy and other storage mapping considerations, while the FileSystemSetting transforms client quality-of-service specifications to storage resource requirements.

8.2.8.6.9.6 CIM_FileSystemConfigurationCapabilities

This element represents the management capabilities of the Filesystem configuration service.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 949: SMI Referenced Properties/Methods for CIM_FileSystemConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for the capabilities of a Filesystem configuration service.
ElementName		string	A User-Friendly Name for this Capabilities.

Table 949: SMI Referenced Properties/Methods for CIM_FileSystemConfigurationCapabilities

Property	Flags	Type	Description & Notes
SupportedActualFileSystemTypes		uint16[]	The Service can be associated with multiple Capabilities that are keyed by the ActualFileSystemType property -- the Configuration capabilities lists all of the supported ActualFileSystemTypes in its SupportedActualFileSystem-Types property.
SupportedSynchronousMethods	N	uint16[]	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called synchronously. Note: A supported method shall be listed in this property or in the SupportedAsynchronousMethods property.
SupportedAsynchronousMethods	N	uint16[]	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called asynchronously. Note: A supported method shall be listed in this property or in the SupportedSynchronousMethods property.
InitialAvailability		uint16	This represents the state of availability of a LocalFileSystem on initial creation.

8.2.8.6.9.7 CIM_FileSystemConfigurationService

The FileSystemConfigurationService provides the methods to manipulate filesystems.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 950: SMI Referenced Properties/Methods for CIM_FileSystemConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A User-friendly name for this Service.
SystemCreationClassName		string	The CIM Class name of the Computer System hosting the Service.
SystemName		string	The name of the Computer System hosting the Service.
CreationClassName		string	The CIM Class name of the Service.
Name		string	The unique name of the Service.
CreateFileSystem()			Creates a FileSystem specified by parameters and Capabilities of the service. If appropriate and supported, a Job may be created.

Table 950: SMI Referenced Properties/Methods for CIM_FileSystemConfigurationService

Property	Flags	Type	Description & Notes
ModifyFileSystem()			Modifies a FileSystem specified by parameters and Capabilities of the service. If appropriate and supported, a Job may be created.
DeleteFileSystem()			Deletes a FileSystem specified by its CIM Reference. If appropriate and supported, a Job may be created.

8.2.8.6.9.8 CIM_FileSystemSetting

This element represents sample configuration settings of a Filesystem. It represents "pre-defined" settings supported by a Filesystem configuration service. A FileSystemSetting element specifies a single ActualFileSystemType which is a weak key with respect to the FileSystemConfigurationService.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: false

Table 951: SMI Referenced Properties/Methods for CIM_FileSystemSetting (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a Filesystem Setting.
ElementName		string	A provider supplied user-Friendly Name for this Setting element.
ActualFileSystemType		uint16	This identifies the type of filesystem that this Setting represents.
FilenameCaseAttributes		uint16	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		uint16[]	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
FilenameReservedCharacterSet		String[]	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames.
Optional Properties/Methods			
DataExtentsSharing		uint16	This allows the creation of data blocks (or storage extents) that are shared between files.
CopyTarget		uint16	This specifies if support should be provided for using the created Filesystem as a target of a Copy operation.

Table 951: SMI Referenced Properties/Methods for CIM_FileSystemSetting (Pre-defined)

Property	Flags	Type	Description & Notes
NumberOfObjectsMin		uint64[]	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjectsMax		uint64[]	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjects		uint64[]	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		uint64[]	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMin		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMax		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
FileNameStreamFormats		uint16[]	This is an array that specifies the stream formats supported for filenames by the created array (e.g., UTF-8).
FilenameFormats		uint16[]	This is an array that specifies the formats supported for filenames by the created array (e.g. DOS 8.3 names).
FilenameLengthMax		uint16[]	This specifies the maximum length of a filename supported by this capabilities.
SupportedLockingSemantics		uint16[]	This array specifies the kind of file access/locking semantics supported by this capabilities.
SupportedAuthorizationProtocols		uint16[]	This array specifies the kind of file authorization protocols supported by this capabilities.
SupportedAuthenticationProtocols		uint16[]	This array specifies the kind of file authentication protocols supported by this capabilities.

8.2.8.6.9.9 CIM_FileSystemSetting

This element represents the configuration settings of a Filesystem. It gets created by theCreateFileSystem extrinsic method when the CIM_LocalFileSystem is created.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Extrinsic(s): ModifyFileSystem
Deleted By : Extrinsic(s): DeleteFileSystem

Class Mandatory: false

Table 952: SMI Referenced Properties/Methods for CIM_FileSystemSetting (Attached to FileSystem)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a FileSystemSetting.
ElementName		string	A client defined user-Friendly Name for this Setting element.
ActualFileSystemType		uint16	This identifies the type of filesystem that this Setting represents.
FilenameCaseAttributes		uint16	This specifies the support provided for using upper and lower case characters in a filename.
ObjectTypes		uint16[]	This is an array that specifies the different types of objects that this filesystem may be used to provide and provides further details in corresponding entries in other attributes.
FilenameReservedCharacterSet		String[]	This string or character array specifies the characters reserved (i.e., not allowed) for use in filenames.
Optional Properties/Methods			
DataExtentsSharing		uint16	This allows the creation of data blocks (or storage extents) that are shared between files.
CopyTarget		uint16	This specifies if support should be provided for using the created Filesystem as a target of a Copy operation.
NumberOfObjectsMin		uint64[]	This is an array that specifies the minimum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjectsMax		uint64[]	This is an array that specifies the maximum number of objects of the type specified by the corresponding entry in ObjectTypes[].
NumberOfObjects		uint64[]	This is an array that specifies the expected number of objects of the type specified by the corresponding entry in ObjectTypes[].
ObjectSize		uint64[]	This is an array that specifies the expected size of a typical object of the type specified by the corresponding entry in ObjectTypes[].

Table 952: SMI Referenced Properties/Methods for CIM_FileSystemSetting (Attached to FileSystem)

Property	Flags	Type	Description & Notes
ObjectSizeMin		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
ObjectSizeMax		uint64[]	This is an array that specifies the minimum size of an object of the type specified by the corresponding entry in ObjectTypes[].
FileNameStreamFormats		uint16[]	This is an array that specifies the stream formats supported for filenames by the created array (e.g., UTF-8).
FilenameFormats		uint16[]	This is an array that specifies the formats supported for filenames by the created array (e.g. DOS 8.3 names).
FilenameLengthMax		uint16[]	This specifies the maximum length of a filename supported by this capabilities.
SupportedLockingSemantics		uint16[]	This array specifies the kind of file access/locking semantics supported by this capabilities.
SupportedAuthorizationProtocols		uint16[]	This array specifies the kind of file authorization protocols supported by this capabilities.
SupportedAuthenticationProtocols		uint16[]	This array specifies the kind of file authentication protocols supported by this capabilities.

8.2.8.6.9.10 CIM_HostedFileSystem

Represents the association between a LocalFileSystem and the Computer System that hosts it.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Static
Deleted By : Extrinsic(s): DeleteFileSystem
Class Mandatory: true

Table 953: SMI Referenced Properties/Methods for CIM_HostedFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Computer System that hosts a LocalFileSystem.
PartComponent		CIM_FileSystem	The hosted filesystem.

8.2.8.6.9.11 CIM_HostedService

In this subprofile, associates the FileSystemConfigurationService to the hosting Computer System.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 954: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting Computer System.
Dependent		CIM_Service	The Filesystem configuration service.

8.2.8.6.9.12 CIM_LocalFileSystem

Represents a LocalFileSystem that a Computer System could make available.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Extrinsic(s): ModifyFileSystem
Deleted By : Extrinsic(s): DeleteFileSystem
Class Mandatory: true

Table 955: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	The CIM class of the hosting Computer System.
CSName		string	The Name of the hosting Computer System.
CreationClassName		string	The CIM class of the this instance.
Name		string	A unique name for this filesystem in the context of the hosting Computer System.
OperationalStatus		uint16[]	The current operational status of the LocalFileSystem.
BlockSize		uint64	The size of a block in bytes for certain filesystems that use a fixed block size when creating filesystems.
FileSystemSize		uint64	The total current size of the filesystem in blocks.
AvailableSpace		uint64	The space available currently in the in blocks.
CaseSensitive		boolean	Whether this filesystem is sensitive to the case of characters in filenames.
CasePreserved		boolean	Whether this filesystem preserves the case of characters in filenames when saving and restoring.
MaxFileNameLength		uint32	The length of the longest filename.
FileSystemType		string	This matches ActualFileSystemType.
IsFixedSize		uint16	Indicates that the filesystem cannot be expanded or shrunk.
Optional Properties/Methods			
EnabledState		uint16	Current state of the local filesystem.

Table 955: SMI Referenced Properties/Methods for CIM_LocalFileSystem

Property	Flags	Type	Description & Notes
OtherEnabledState		string	Vendor-specific state of the local file-system indicated by EnabledState = 1("Other").
TimeOfLastStateChange		datetime	A timestamp indicating when the state was last changed.
RequestedState		uint16	Not supported.
Root		string	A path that specifies the root of the file-system in an unitary Computer Systems acting as a FileServer.
ReadOnly		boolean	Indicates that this is a read-only filesystem that does not allow modifications.
EncryptionMethod		string	Indicates if files are encrypted and the method of encryption.
CompressionMethod		string	Indicates if files are compressed before being stored, and the methods of compression.
CodeSet		uint16[]	The codeset used in filenames.
ClusterSize		uint32	
NumberOfFiles		uint64	The actual current number of files in the filesystem.
ResizeIncrement		uint64	The size by which to increase the size of the filesystem when requested.
RequestStateChange()			Not supported.

8.2.8.6.9.13 CIM_LocalFileSystem

The represents LocalFileSystems that are hosted by the NAS Head.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Extrinsic(s): ModifyFileSystem
Deleted By : Extrinsic(s): DeleteFileSystem
Class Mandatory: false
No specified properties or methods.

8.2.8.6.9.14 CIM_LogicalFile

A LogicalFile (or Directory subclass) is used to provide mountpoints on a Computer System for LocalFIleSystems.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Extrinsic(s): ModifyFileSystem
Deleted By : Extrinsic(s): DeleteFileSystem or ModifyFileSystem
Class Mandatory: true

Table 956: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	CIM Class of the Computer System that hosts the Filesystem of this File.
CSName		string	Name of the Computer System that hosts the Filesystem of this File.

Table 956: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
FSCreationClassName		string	CIM Class of the LocalFileSystem on the Computer System that contains this File.
FSName		string	Name of the LocalFileSystem on the Computer System that contains this File.
CreationClassName		string	CIM Class of this instance of LogicalFile.
Name		string	The unique Name of this LogicalFile, weak with respect to a containing Directory.
Optional Properties/Methods			
FileSize		uint64	The size of the file, in bytes.
CreationDate		datetime	A timestamp indicating when the file was created.
LastModified		datetime	A timestamp indicating when the file was last modified.
ElementName		string	A user-friendly name for the file.

8.2.8.6.9.15 CIM_ResidesOnExtent

Represents the association between a local FileSystem and the underlying StorageExtent/LogicalDisk that it is built on.
Created By : Extrinsic(s): CreateFileSystem
Modified By : Static
Deleted By : Extrinsic(s): DeleteFileSystem
Class Mandatory: true

Table 957: SMI Referenced Properties/Methods for CIM_ResidesOnExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_LogicalElement	The local filesystem that is built on top of a StorageExtent.
Antecedent		CIM_StorageExtent	The StorageExtent that underlies a LocalFileSystem.

8.2.8.6.9.16 CIM_SettingAssociatedToCapabilities

Represents the association between a FilesystemCapabilities and a supported FileSystemSetting element.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: false

Table 958: SMI Referenced Properties/Methods for CIM_SettingAssociatedToCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Capabilities	A FileSystemCapabilities element.
Dependent		CIM_SettingData	A FileSystemSetting element.

8.2.8.6.10 Related Standards

Table 959: Related Standards for Filesystem Manipulation

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.7 File Export Manipulation Subprofile

8.2.8.7.1 Description

The File Export Manipulation Subprofile is an extension of the Self-Contained NAS and NAS Head Profiles.

The File Export Manipulation Subprofile provides configuration support for exporting elements ('files') of a FileSystem. The configuration methods are part of a FileExportService. FileExportService(s) are hosted by a ComputerSystem that exports the files (these would be the Filers in the NAS offering). These shared elements (FileShares) are accessed through ServiceAccessPoints hosted by the Filer. FileShares are associated with the FileExportService via ServiceAffectsElement and with the ServiceAccessPoint(s) via SAPAvailableToElement.

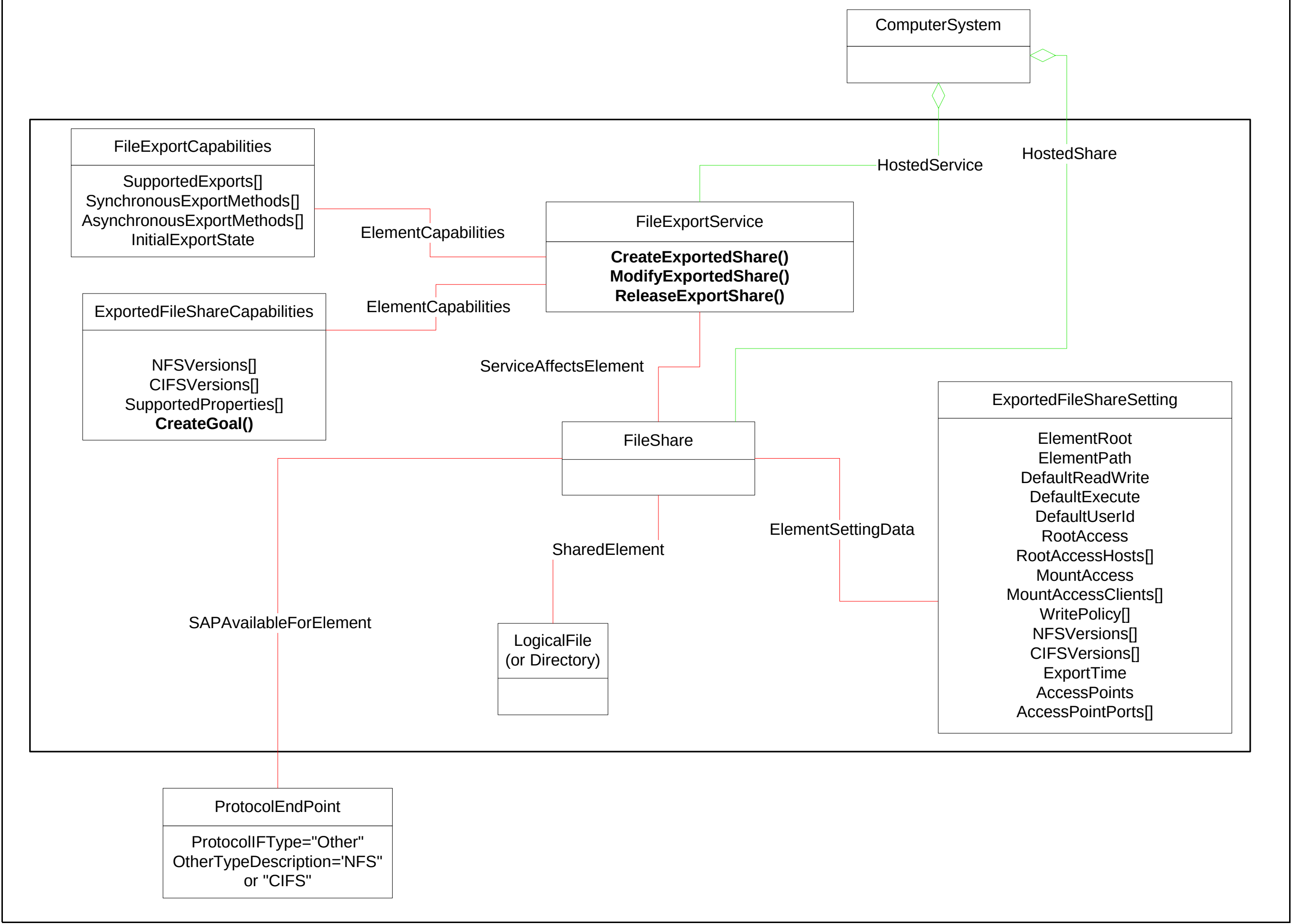
The File Export Manipulation Subprofile supports creation, modification and deletion of FileShares that are exported.

8.2.8.7.1.1 Instance Diagrams

File Export Creation classes and associations

Figure 137: "File Export Manipulation Subprofile Instance Diagram" illustrate the constructs involved with creating and exporting File Shares for NAS.

Figure 137: File Export Manipulation Subprofile Instance Diagram



The FileExportService provides configuration support for exporting elements ('files') of a FileSystem. FileExportService(s) are hosted by a ComputerSystem that exports the files (these would be the Filers in a NAS Head). These shared elements (FileShares) are accessed through ServiceAccessPoint(s) hosted by the Filer. FileShares are associated with the Service via ServiceAffectsElement and with the ServiceAccessPoint(s) via SAPAvailableToElement.

8.2.8.7.2 Health and Fault Management Considerations

Under Consideration for a future standard.

8.2.8.7.3 Cascading Considerations

Not Applicable.

8.2.8.7.4 Supported Subprofiles and Packages

Table 960: Supported Subprofiles for File Export Manipulation

Registered Subprofile Names	Mandatory	Version
Job Control	No	1.1.0

8.2.8.7.5 Methods of the Profile

8.2.8.7.5.1 Extrinsic Methods of the Profile

The extrinsic methods for this profile are defined in Table 961, “FileExportManipulation Methods”.

Table 961: FileExportManipulation Methods

Method	Created Instances	Deleted Instances	Modified Instances
CreateExportedShare	FileShare (Export) ExportedFileShareSetting ElementSettingData HostedShare SharedElement SAPAvailableForElement ServiceAffectsElement LogicalFile (or Directory)	N/A	N/A
ModifyExportedShare	N/A	N/A	ExportedFileShareSet- ting FileShare
ReleaseExportedShare	N/A	FileShare (Export) ExportedFileShareSetting ElementSettingData HostedShare SharedElement SAPAvailableForElement ServiceAffectsElement LogicalFile (or Directory)	N/A
CreateGoal	N/A	N/A	N/A

Creation of a FileShare for export (CreateExportedFileShare), creates a FileShare and a ExportedFileShareSetting, and the FileShare associations to a ComputerSystem, a LogicalFile (or Directory), a Protocol Endpoint, the Service that created it and its export Settings. In addition, if an instance for the LogicalFile (or Directory) does not exist, then the instance will be created.

Releasing a FileShare (via ReleaseExportedFileShare) deletes the FileShare and its ExportedFileShareSetting, and all the FileShare associations to other instances. In addition, it will result in deletion of the LogicalFile (or Directory) instance, if it is the last FileShare defined on that LogicalFile (or Directory).

The only element that can be modified in the File Export Manipulation Subprofile is the ExportedFileShareSetting.

8.2.8.7.5.1.1 CreateExportedShare

Start a Job to create a FileShare from an element of a FileSystem. Makes an element of a FileSystem available as a FileShare - this is returned as the parameter TheShare of type FileShare. The FileSystem whose element is exported is expected to be hosted by the host of the FileExportService.

If 0 is returned, the method completed successfully and no ConcreteJob instance was required. If 0x1000 is returned, a ConcreteJob has been started to create the FileShare. The Job's reference will be returned in the output parameter Job. If the Job succeeds, the FileShare will be created and configured and ready to be exported. If the FileShare has been configured to be exported at a later time, its 'Enabled' attribute will be set to True when it is ready to be exported. The FileShare will have a HostedShare association to the host ComputerSystem.

The parameter TheElementRoot specifies, using a pathname, the FileSystem whose element is being exported. If the default local file system of the host is being exported, the pathname would be the root of the file system hierarchy on the host, but for a re-exported FileSystem, this would be the mount-point at which the FileSystem was mounted.

The shared sub-element is specified by a path relative to the root of the specified FileSystem.

Goal is an embedded parameter of type ExportedFileShareSetting that allows the client to specify the properties desired for the share.features.

If the method is successful, it will return a FileShare in the OUT parameter TheShare. The settings on the FileShare will be specified by the Setting Data associated with the TheShare element via ElementSettingData.

CreateExportedShare (

[OUT, IN (false), Description(Reference to the job (may be null if job completed).)]

    CIM_ConcreteJob REF **Job**,

[IN, OUT, Description (A reference indicating an element whose sub-element is being exported. The class that Root references is a FileSystem, a FileShare that has a MountedElement association (or a derived class of MountedElement) to a LogicalFile (or Directory), or a LogicalFile (or a derived class such as Directory) that has a MountedElement association to a FileShare or FileSystem.

If Root is NULL, it indicates the root of the FileExportService host's default local FileSystem, that is used as the default local name space.)]

    CIM_LogicalElement string **Root**,

[IN, Description (A string representing a path to the shared element from the Directory indicated by Root.

Multiple paths could lead to the same element but the access rights or other privileges could be specific to the path. The client needs to specify the path.

If SharedElementPath is NULL or the empty string, it indicates the \”root\” LogicalElement contained by Root.)]

    string **SharedElementPath**,

[IN, EmbeddedInstance(“CIM_ExportedFileShareSettingGoal “), Description (The client-specified requirements for how the specified FileShare element is to be shared or exported by the FileExportService. This is an element of the CIM_ExportedFileShareSetting class, or a derived class, encoded as a string-valued embedded object parameter. If NULL or the empty string, the default configuration will be specified by the FileExportService.)]

string **Goal**,

[OUT, Description (If successful, this specifies the share.)]

CIM_FileShare REF **TheShare**)

[IN, Description (A reference to a concrete derived class of CIM_Identity that indicates the user id to use for default access to this share. A NULL value indicates that no user id is specified. The provider is expected to surface this access using the privilege model.)]

CIM_Identity REF **DefaultUserId**,

[IN, Description (An array of strings that specify the hosts that have root access to this Share, if the CIM_ExportedFileShareSetting.RootAccess property is set to 'Allow Root Access'. Each entry specifies a host by a vendor-specific host-id, prefixed with '+' or '-' to indicate that access is either Granted or Denied. The name of the host is its Durable Name, which is expected to be a fully-qualified-domain-name or its IP Address. If one of the entries is '+*', root access will be allowed from all hosts. If one of the entries is '-*', root access will be denied to all hosts, effectively overriding the value of the property CIM_ExportedFileShareSetting.RootAccess. The provider is expected to surface this access using the privilege model.

This property needs to be a string because the remote host may not be known to the provider and therefore a reference to the host may not exist.),

ArrayType ("Indexed"),

ModelCorrespondence { "CIM_ExportedFileShareSetting.RootAccess" }]

string **RootAccessHosts[]**,

[IN, Description (An array of strings that specify the ServiceAccessPoints that can connect to this Share, if the CIM_ExportedFileShareSettings.AccessPoints property is set to 'Named Ports'. Each entry specifies one or more access points by its Name, unique within the System hosting the FileShare. The ids may be prefixed with '+' or '-'to indicate that access is to be granted or denied.

If one of the entries is '+*', all access points supported by the service will be supported. If one of the entries is '-*', all access points will be denied access, effectively overriding the value of the property ExportedFileShareSetting.AccessPoints. The provider is expected to surface these access rights (whether granted or denied) using the privilege model. Any AccessPoints granted access via this parameter will also be associated to this share with SAPAvailableForElement. If the AccessPoint is not already enabled it will appear in a disabled state.

This property needs to be a string because the access point may not be known to the provider and therefore a reference to the ServiceAccessPoint may not exist.),

ArrayType ("Indexed"),

ModelCorrespondence { CIM_ExportedFileShareSetting.AccessPoints" }]

string **AccessPointPorts[]**);;

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "FileExportService Not Accessible", "Root is not accessible", "Base Directory element of Root is Not Accessible", “Path does not specify a shareable element", "DMTF Reserved", “Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}

8.2.8.7.5.1.2 ModifyExportedShare

Start a Job to modify an Exported FileShare. This cannot be used to change the LogicalFile element that has been exported.

If 0 is returned, the method completed successfully and no ConcreteJob instance was required. If 0x1000 is returned, a ConcreteJob has been started to modify the FileShare. The Job's reference will be returned in the output parameter Job. If the Job succeeds, the FileShare will be modified and configured and ready to be exported. If the FileShare has been configured to be exported at a later time, its 'Enabled' attribute will be set to True when it is ready to be exported. The FileShare will have a HostedShare association to the host ComputerSystem.

Goal is an embedded parameter of type ExportedFileShareSetting that allows the client to specify the properties desired for the share.

If the method is successful, it will return a reference to the changed FileShare in the OUT parameter TheShare.

The input TheShare shall not be NULL.),

ModifyExportedShare (

[OUT, IN (false), Description(Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, Description (A reference indicating an element whose sub-element is being exported. The class that Root references is a FileSystem, a FileShare that has a MountedElement association (or a derived class of MountedElement) to a Directory, or a Directory that has a MountedElement association to a FileShare or FileSystem. If the FileShare being modified is currently exported or imported, this parameter should indicate the same Root FileSystem or FileShare element.

If Root is NULL, it indicates no change to the current root.)]

CIM_LogicalElement REF **Root**,

[IN, Description (A string representing a path to the shared element from the Directory element indicated by Root. If the FileShare being modified is currently exported or imported, this parameter should specify the same shared element, even if via a different path.

Multiple paths could lead to the same element but the access rights or other privileges could be specific to the path. The client needs to specify the path during creation.

If SharedElementPath is NULL, it indicates no change to the current path. If SharedElementPath is the empty string, it indicates the element indicated by Root.)]

string **SharedElementPath**,

[IN, Description (The client-specified requirements for how the export settings for the specified FileShare element are to be modified by the FileExportService. If the FileShare is currently imported and not exported this will set up the necessary SharedElement, SharedElementRoot, HostedShare, and other associations. Goal is an element of the CIM_ExportedFileShareSetting class, or a derived class, encoded as a string-valued embedded object parameter. If NULL or the empty string, the existing configuration shall include an ExportedFileShareSetting which will not be changed. Any differences in property values will be merged by the FileExportService.),

EmbeddedInstance ("CIM_ExportedFileShareSetting")]

string **Goal**,

[OUT, IN, Description (As an OUT Parameter, if successful, this specifies the share. As an IN Parameter, it specifies the share that is to be modified or whose settings are being queried.)]

CIM_FileShare REF **TheShare**)

[IN, Description (An enumerated integer that specifies the action that the provider should take if the FileShare is still in use when this request is made. The WaitTime parameter indicates the 'specified time' used for this function.

This option is only relevant if the FileShare needs to be made unavailable while the request is being executed.),

ValueMap { "2", "3", "4", "..", "0x1000..0xFFFF" },

Values { "Do Not Execute Request", "Wait for specified time, then Execute Request Immediately", "Attempt Quiescence for specified time, then Execute Request Immediately", "DMTF Reserved", "Vendor Defined" }]

uint16 **InUseOptions**,

[IN, Description (An integer that indicates the time (in seconds) that the provider needs to wait before executing this request if it cannot be done while the FileShare is in use. If WaitTime is not zero, the method will create a job, if supported by the provider, and return immediately. If the provider does not support asynchronous jobs, there is a possibility that the client could time-out before the job is completed.

The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Execute Request' and will be performed asynchronously if possible.),

Units ("seconds")]

uint32 **WaitTime**,

[IN, Description (A reference to a concrete derived class of CIM_Identity that indicates the user id to use for default access to this share. A NULL value indicates that any existing user id is not changed. The provider is expected to surface this access using the privilege model. This method does not support disabling the currently specified default user id, which needs to be done using the privilege model.)]

CIM_Identity REF **DefaultUserId,**

[IN, Description (An array of strings that specify additional hosts that have root access to this Share, if the CIM_ExportedFileShareSetting.RootAccess property is set to 'Allow Root Access'. Each entry specifies a host by a vendor-specific host-id, prefixed with '+' or '-' to indicate that access is either Granted or Denied. The name of the host is its Durable Name, which is expected to be a fully-qualified-domain-name or its IP Address. If one of the entries is '+*', root access will be allowed from all hosts. If one of the entries is '-*', root access will be denied to all hosts, effectively overriding the value of the property CIM_ExportedFileShareSetting.RootAccess. If this is a null entry, the currently configured set of hosts will not be changed. If this is an empty array, the currently configured set of hosts will be cleared. The provider is expected to surface this access using the privilege model.

This property needs to be a string because the remote host may not be known to the provider and therefore a REF to the host may not exist.),

ArrayType ("Indexed"),

ModelCorrespondence { "CIM_ExportedFileShareSetting.RootAccess" }]

string **RootAccessHosts[]**,

[IN, Description (An array of strings that specify additional ServiceAccessPoints that can connect to this Share, if the CIM_ExportedFileShareSettings.AccessPoints property is set to 'Named Ports'. Each entry specifies one or more access points by its Name, unique within the System hosting the FileShare. The ids may be prefixed with '+' or '-'to indicate that access is to be granted or denied.

If one of the entries is '+*', all access points supported by the service will be supported. If one of the entries is '-*', all access points will be denied access, effectively overriding the value of the property ExportedFileShareSetting.AccessPoints. If this is a null entry, the currently configured set of access points will not be changed. If this is an empty array, the currently configured set of access points will be cleared. The provider is expected to surface these access rights (whether granted or denied) using the privilege model. Any AccessPoints granted access via this parameter will also be associated to this share with SAPAvailableForElement. If the AccessPoint is not already enabled it will appear in a disabled state.

This property needs to be a string because the access point may not be known to the provider and therefore a REF to the ServiceAccessPoint may not exist.),

ArrayType ("Indexed"),

ModelCorrespondence { "CIM_ExportedFileShareSetting.AccessPoints" }]

string **AccessPointPorts[]**);

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "FileExportService Not Accessible", "Root is not accessible", "Base Directory element of Root is Not Accessible", "Path does not specify a shareable element", "Share in use and cannot be Modified, Failed", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}]

8.2.8.7.5.1.3 ReleaseExportedShare

Start a Job to release a share exposed by a file system.

If 0 is returned, the method completed successfully and no ConcreteJob instance was required. If 0x1000 is returned, a ConcreteJob will be started to create the service. The Job's reference will be returned in the OUT parameter Job.

If the method is successful, the FileShare element will have been deleted along with all associated references. The element will not be exported anymore through this FileShare. If InUseOptions are specified, this method will succeed only if no more clients are accessing the share.

ReleaseExportedShare (

[OUT, IN (false), Description (Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, Description ("The shared element.")]

CIM_FileShare REF **TheShare,**

[IN, Description (An enumerated integer that specifies the action that the provider should take if the FileShare is still in use when this request is made. The WaitTime parameter indicates the 'specified time' used for this function.

This option is only relevant if the FileShare needs to be made unavailable while the request is being executed.), ValueMap { "2", "3", "4", "..", "0x1000..0xFFFF" },

Values { "Do Not Execute Request", "Wait for specified time, then Release Immediately", "Attempt Quiescence for specified time, then Release Immediately", "DMTF Reserved", "Vendor Defined"}]

uint16 **InUseOptions[]**

[IN, Description ("An integer that indicates the time (in seconds) that the provider needs to wait before releasing this FileShare. If WaitTime is not zero, the method will create a job, if supported by the provider, and return immediately. If the provider does not support asynchronous jobs, there is a possibility that the client could time-out before the job is completed.

The combination of InUseOptions = '4' and WaitTime ='0' (the default) is interpreted as 'Wait (forever) until Quiescence, then Release' and will be performed asynchronously if possible.),

Units ("seconds")]

uint32 **WaitSeconds[]**

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Share in use, Failed", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}

8.2.8.7.5.1.4 CreateGoal

This method is on the ExportedFileShareCapabilities and starts a job to create a ExportedFileShareSetting from a ExportedFileShareSetting provided by the caller. If the operation completes successfully and did not require a long-running ConcreteJob, it will return 0. If 4096/0x1000 is returned, a ConcreteJob will be started to create the element. A Reference to the ConcreteJob will be returned in the output parameter Job.

If the input TemplateGoal is NULL, this method returns a copy of the ExportedFileShareSetting identified by the DefaultCapability association.

The output is returned as the SupportedGoal parameter. Both TemplateGoal and SupportedGoal are embedded objects and are provided by the client.

CreateGoal(

[OUT, IN (false), Description (Reference to the job (may be null if job completed).)]

CIM_ConcreteJob REF **Job**,

[IN, EmbeddedInstance(“CIM_ExportedFileShareSetting”), Description (A CIM_ExportedFileShareSetting element that is used as the goal element to be used for matching.")]

string **TemplateGoal,**

[OUT, IN(false), EmbeddedInstance("CIM_ExportedFileShareSetting"), Description (A CIM_ExportedFileShareSetting element that is returned as the best supported match to the TemplateGoal.")]

string **SupportedGoal**);

Error returns are:

{"Job Completed with No Error", "Not Supported", "Unknown", "Timeout", "Failed", "Invalid Parameter", "Template Goal cannot be matched.", "DMTF Reserved", "Method Parameters Checked - Job Started", "Method Reserved", "Vendor Specific"}]

8.2.8.7.5.2 Intrinsic Methods of the Profile

None.

8.2.8.7.6 Client Considerations and Recipes

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

```
sub GetFSSetting(IN REF CIM_FileSystem $fs,
                 OUT CIM_FileSystemSettingData $setting);

sub GetFSServer(IN REF CIM_FileSystem $fs,
                 OUT CIM_ComputerSystem $system);

sub GetFSCapabilityFromServer(IN REF CIM_System $server,
                              OUT CIM_FileSystemConfigurationServiceCapabilities
$capability,
                              OUT CIM_FileSystemConfigurationService
$fsconfigurator,
                              IN Optional String $filesystemtype = "",
                              IN Optional String $otherpropertyname = NULL,
                              IN Optional String $otherpropertyvalue = NULL);

sub GetFSCapabilityFromFileSystem(IN REF CIM_FileSystem $fs,
                   OUT CIM_FileSystemConfigurationServiceCapabilities $capability,
                   OUT CIM_FileSystemConfigurationService $fsconfigurator);

sub GetExportServiceAndCapabilities(IN REF CIM_FileSystem $fs,
                                    IN String $sharetype,
                                    OUT CIM_FileExportService $feservice,
                                    OUT CIM_ExportedFileShareCapabilities
$efscapability);
```

Conventions used in the NAS recipes:

- When there is expected to be only one association of interest, the first item in the array returned by the Associators( ) call is used without further validation. Real code will need to be more robust.
- We use Values and Valuemap members as equivalent. In real code, client-side magic is required to convert the integer representation into the string form given in the MOF.
- It was requested that we use LogicalDisk--a subclass of StorageExtent--instead of StorageExtent itself in these recipes, for reasons that have nothing to do with the recipes themselves (the extra properties in LogicalDisk are not used herein). You should be able to simply use StorageExtent in your code.

8.2.8.7.6.1 Creation of a FileShare for Export

```
//
// Create an NFS or CIFS FileSystemShare
//
sub CreateFileSystemShare(IN String $sharetype,        // CIFS, NFS, etc.
                          IN CIM_FileSystem $fs,       // the filesystem
                        IN String $fspath,          // subpath in the filesystem, 
or ""
                        IN String[] $propnames,      // names of desired properties
                          IN String[] $propvals,       // values of desired 
properties
                          OUT CIM_FileShare $fssh,
                          OUT CIM_Job $job)
{
    //
    // Get the service and capabilities
    //
    &GetExportServiceAndCapabilities($fs, $sharetype, $feservice, $efscapability);

    //
    // Call ExportedFileShareCapabilities.CreateGoal(nullTemplate, Goal) to get a
    // seed goal for ExportedFileShareSetting
    //
      $efscapability.CreateGoal(NULL, $goal);

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

    //
    // Call ExportedFileShareCapabilities.CreateGoal(Goal-N', Goal-N) to get the
    // next goal for ExportedFileShareSetting -- iterate until satisfied or give up
    // (beware infinite loops).
    //
    // NOTE: this sample code gives up if anything doesn't work
     //
    $efscapability.CreateGoal($goal, $settings);
    #i = 0;
    while ($propnames->[#i] != NULL) {
        if ($goal.$propnames->[#i] != $propvals->[#i]) {
            //
            // give up
            //
            return NULL;
        }
        #i++;
    }
    //
    // Note, the FileSystem in $fs has a name ElementRoot that identifies
    // the FileSystem within the ComputerSystem that is the Filer.
    //
    // Call FileExportService.CreateExportedShare(job, ElementRoot, SubElement,
    // Goal, sharename)
    //
    $feservice.CreateExportedShare($job, $fs.ElementRoot, $fspath,
                                   $settings, $fssh);

    return TRUE;
}
```

8.2.8.7.6.2 Modification of an Exported FileShare

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

    //
    // Modify the copied ExportedFileShareSetting to the new desired properties
    //
    #i = 0;
    while ($propnames->[#i] != NULL) {
        $setting.$propnames->[#i] = $propvals->[#i];
    }
    //
    // Get the sharetype from the FileSystemShare
    //
    $sharetype = $settings.FileSharingProtocol;

    //
    // Get the service and capabilities
    //
    &GetExportServiceAndCapabilities($fs, $sharetype, $feservice, $efscapability);

    //
    // Call ExportedFileShareCapabilities.CreateGoal(Goal-N’, Goal-N) to get the 
next
    // goal for ExportedFileShareSetting -- iterate until satisfied or give up 
(beware
    // infinite loops)
    //
    // Note: this code just gives up if it doesn’t get what it wants
    //
    $efscapability.CreateGoal($goal, $settings);
    #i = 0;
    while ($propnames->[#i] != NULL) {
        if ($goal.$propnames->[#i] != $propvals->[#i]) {
            //
            // give up
            //
            return NULL;
        }
        #i++;
    }

    //
    // Call FileExportService.ModifyExportedFileShare(job, Goal, Share)
    //
    $feservice.ModifyExportedFileShare($job, $goal, $fssh);
    return TRUE;
}
```

8.2.8.7.6.3 Removal of an Exported FileShare

```
//
// UnExport an ExportedFileShare
//



//
// Before calling, decide whether to force connections closed,
// set a uint16 “force” parameter to 0 or 1 accordingly, and
// set the waittime parameter to the desired number of seconds
// to wait before forcibly disconnecting the share (5 minutes
// is standard).
//
sub UnExportFileSystemShare(IN CIM_FileSystemShare $fssh,
                            IN uint16 $force,
                            IN uint32 $waittime,
                            IN String $notification,
                            OUT CIM_Job $job);
{
    //
    // If waittime > 0, set force to 2 to disambiguate a force with no wait
    // and a force with wait -- see the specification of ReleaseExportedShare.
    //
    if ($force > 0 && $waittime > 0) {
        $force = 2;
    }
    //
    // clients of the share may have registered for an indication
    // when a share is disconnected
    //
    <send indication -- see indications recipes>

    //
    // Get the service and capabilities
    //
    &GetExportServiceAndCapabilities($fs, $sharetype, $feservice, $efscapability);

    //
    // Call ReleaseExportedShare()
    // This tells the managed device to wait for an amount of time specified by
    // $waittime if there are any clients still connected.
    //

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

    return TRUE;
}
```

8.2.8.7.6.4 Get a FileExportService and ExportedFileShareCapabilities from a Filesystem

```
//
//  Get a FileExportService and ExportedFileShareCapabilities from a Filesystem
//
sub GetExportServiceAndCapabilities(IN REF CIM_FileSystem $fs,
                                    IN String $sharetype,
                                    OUT CIM_FileExportService $feservice,
                                   OUT ExportFileShareCapabilities $efscapability)
```

```
{
    //
    // Get a FileExportService via the HostedService association to the Filer
    // ComputerSystem
    //
    &GetFSServer($fs, $system);
    $feservices->[] = Associators($system,
                                "CIM_HostedService",
                                "CIM_FileExportService",
                                "Antecedent",
                                "Dependent");
    #i = 0;
    while ($feservices[#i] != NULL) {
        //
       // Get an ExportedFileShareCapabilities from the FileExportService via the
        // ElementCapabilities association to the ComputerSystem (it's indexed by
        // NFS/CIFS/other sharing service and possibly other properties)
        // Note: NFS and CIFS are two instances of the same service type with
different
         // capabilities
         //
        $efscapabilities = Associators($feservices[#i],
                                "CIM_ElementCapabilities"
                                "CIM_ExportedFileShareCapabilities",
                                "ManagedElement",
                                "Capabilities")->[0];
            #j = 0;
            while ($efscapabilities->[#j] != NULL) {
                if ($efscapabilities->[#j].FileSharingProtocol == $sharetype) {
                    $efscapability = $efscapabilities->[#j];
                    $feservice = $feservices->[#i];
                    break;
                }
            }
            #j++;
        }
        #i++;
    }
    if (#efscapability == NULL) {
        <indicate error>
        return NULL;
    }
}
```

8.2.8.7.6.5 File Export Manipulation Supported Capabilities Patterns

Table 962, “SMI-S File Export Supported Capabilities Patterns” lists the capabilities patterns that are formally recognized by this version of SMI-S for determining capabilities of various implementations:

Table 962: SMI-S File Export Supported Capabilities Patterns

SupportedExports	SynchronousExportMethods	AsynchronousExportMethods	InitialExportState
NFS, CIFS	Export Creation, Export Modification, Export Deletion	Null	*
NFS, CIFS	Null	Export Creation, Export Modification, Export Deletion	*
NFS, CIFS	Null	Null	Null

<u>Note:</u> Asterisk (*) means any state is valid.

8.2.8.7.7 Registered Name and Version

File Export Manipulation version 1.1.0

8.2.8.7.8 CIM Server Requirements

Table 963: CIM Server Requirements for File Export Manipulation

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.7.9 CIM Elements

Table 964: CIM Elements for File Export Manipulation

Element Name	Description
Mandatory Classes	
CIM_ElementCapabilities (8.2.8.7.9.1)	In this subprofile, associates the File Export Service to the capabilities that it supports.
CIM_ElementCapabilities (8.2.8.7.9.2)	In this subprofile, associates the FileExportService to its ExportedFileShareCapabilities.
CIM_ElementSettingData (8.2.8.7.9.3)	Associates a configuration setting to the configured element. It is used in this subprofile with FileShare and ExportedFileShareSetting elements.
CIM_ExportedFileShareCapabilities (8.2.8.7.9.4)	This element represents the Capabilities of the File Export Service for managing FileShares. The Service can be associated with multiple Capabilities that are keyed by the FileSharingProtocol property.
CIM_ExportedFileShareSetting (8.2.8.7.9.5)	This element represents pre-defined configuration settings of a File Share supported by a FileExportService. A ExportedFileShareSetting element specifies a single FileSharingProtocol which is a weak key with respect to the File Export Service.
CIM_ExportedFileShareSetting (8.2.8.7.9.6)	This element represents the client defined configuration settings of a File Share intended for Export. It is created as a result of a CreateExportedShare extrinsic.
CIM_FileExportCapabilities (8.2.8.7.9.7)	This element represents the management Capabilities of the File Export Service.
CIM_FileExportService (8.2.8.7.9.8)	The File Export Service provides the methods to create and export file elements as shares.
CIM_FileShare (8.2.8.7.9.9)	Represents the sharing characteristics of a particular file element.
CIM_HostedService (8.2.8.7.9.10)	In this subprofile, associates the File Export Service to the hosting Computer System.
CIM_HostedShare (8.2.8.7.9.11)	Represents that a FileShare element (sharing a LogicalFile or Directory) is hosted by a Computer System.
CIM_LogicalFile (8.2.8.7.9.12)	A LogicalFile (or Directory subclass) is available for export via a fileshare hosted on a ComputerSystem.
CIM_ServiceAffectsElement (8.2.8.7.9.13)	Associates the File Export Service to the elements that the service affects (such as a FileShare configured for exporting a LogicalFile).
CIM_SettingAssociatedToCapabilities (8.2.8.7.9.14)	Represents the association between a ExportedFileShareCapabilities and a supported ExportedFileShareSetting element.
CIM_SharedElement (8.2.8.7.9.15)	In this subprofile, represents the identity association between an exporting FileShare element and the actual shared LogicalFile or Directory. The implication is that these are the same element even though represented by two SMIS objects.

Table 964: CIM Elements for File Export Manipulation

Element Name	Description
CIM_SharedElementRoot (8.2.8.7.9.16)	In this subprofile, represents the association between an exporting FileShare element and a directory or file in the file namespace of the hosting ComputerSystem on which the LocalFileSystem containing the shared LogicalFile or Directory element is mounted.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SharedElement AND SourceInstance.SameElement ISA CIM_FileShare	CQL - Creation of an exported file share.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SharedElement AND SourceInstance.CIM_SharedElement::SameElement ISA CIM_FileShare	CQL - Deletion of an exported file share.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_FileShare AND SourceInstance.CIM_FileShare::OperationalStatus[*] <> PreviousInstance.CIM_FileShare::OperationalStatus[*]	CQL - Change of state of a FileShare. PreviousInstance is optional, but may be supplied by an implementation of the subprofile.
Optional Indications	
SELECT SourceInstance.* SourceInstance.SameElement.* SourceInstance.SystemElement.* FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SharedElement AND SourceInstance.CIM_SharedElement::SameElement ISA CIM_FileShare	CQL - Creation of an exported file share.
SELECT OBJECTPATH(IC.SourceInstance)AS SharePath, IC.InstnaceID, IC.SharingDirectory, LF.FSName, LF.Name FROM CIM_InstCreation IC, CIM_SharedElement SE, CIM_LogicalFile LF WHERE SourceInstance ISA CIM_FileShare AND OBJECTPATH(LF) = A.SystemElement AND OBJECTPATH(IC.SourceInstance) = A.SameElement	CQL - Creation of an exported file share. This indication returns properties of the FileShare and the name of the FS and the name of the Directory it is defined on.
SELECT SourceInstance.* SourceInstance.SameElement.* SourceInstance.SystemElement.* FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SharedElement AND SourceInstance.CIM_SharedElement::SameElement ISA CIM_FileShare	CQL - Deletion of an exported file share.
SELECT OBJECTPATH(SourceInstance)AS SharePath FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FileShare	CQL - Deletion of an exported file share.

8.2.8.7.9.1 CIM_ElementCapabilities

In this subprofile, associates the File Export Service to the capabilities that it supports.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 965: SMI Referenced Properties/Methods for CIM_ElementCapabilities (FileExportCapabilities)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The FileExportService.
Capabilities		CIM_Capabilities	The FileExportCapabilities.

8.2.8.7.9.2 CIM_ElementCapabilities

In this subprofile, associates the FileExportService to its ExportedFileShareCapabilities.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 966: SMI Referenced Properties/Methods for CIM_ElementCapabilities (ExportedFileShareCapabilities)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The FileExportService.
Capabilities		CIM_Capabilities	The ExportedFileShareCapabilities.

8.2.8.7.9.3 CIM_ElementSettingData

Associates a configuration setting to the configured element. It is used in this subprofile with FileShare and ExportedFileShareSetting elements.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Static
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 967: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The FileShare used for exporting an element.
SettingData		CIM_SettingData	The current configuration of the FileShare.

8.2.8.7.9.4 CIM_ExportedFileShareCapabilities

This element represents the Capabilities of the File Export Service for managing FileShares. The Service can be associated with multiple Capabilities that are keyed by the FileSharingProtocol property.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 968: SMI Referenced Properties/Methods for CIM_ExportedFileShareCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for a capability of a File Export Service
ElementName		string	A provider supplied user-Friendly Name for this Capabilities element.
FileSharingProtocol		uint16	This identifies the single file sharing protocol (e.g., NFS or CIFS) that this Capabilities represents. This is a weak key with respect to the FileExportService.
ProtocolVersions		string[]	An array listing the versions of the protocol specified by the FileSharingProtocol property.
SupportedProperties		uint16[]	This is the list of configuration properties (of ExportedFileShareSetting) that are supported for specification at creation time by this Capabilities element.
CreateGoal()			This extrinsic method supports the creation of a ExportedFileShareSetting that is a supported variant of a ExportedFileShareSetting passed in as an embedded IN parameter. The method returns the supported ExportedFileShareSetting as an embedded OUT parameter.

8.2.8.7.9.5 CIM_ExportedFileShareSetting

This element represents pre-defined configuration settings of a File Share supported by a FileExportService. A ExportedFileShareSetting element specifies a single FileSharingProtocol which is a weak key with respect to the File Export Service.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 969: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Sample on Capabilities)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for an exported or owned FileShare Setting.
ElementName		string	A provider supplied user-Friendly Name for this Setting element.

Table 969: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Sample on Capabilities)

Property	Flags	Type	Description & Notes
FileSharingProtocol		uint16	This, together with ProtocolVersions, identifies the protocol for file-sharing that is supported by the Computer System.
ProtocolVersions		string[]	This, together with FileSharingProtocol, identifies the version of the protocol for file-sharing that is supported by the Computer System.
Optional Properties/Methods			
InitialEnabledState		uint16	This indicates the enabled/disabled states initially set for a created file share element.
OtherEnabledState		string	A vendor-specific description of the initial enabled state of a created fileshare if InitialEnabledState=1("Other").
DefaultReadWrite		uint16	Indicates the default privileges that are supported for read and write authorization to the newly created fileshare. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
DefaultExecute		uint16	Indicates the default privileges that are supported for execute authorization to the newly created fileshare. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
ExecuteSupport		uint16	Indicates if the sharing mechanism provides specialized support for executing an element shared through this fileshare (for instance, does it provide paging support for text pages).
DefaultUserIdSupported		uint16	Indicates whether the FileShare will use a default user id to control access to the share if the id of the importing client is not provided. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
RootAccess		uint16	Indicates whether the FileShare will support default access privileges to administrative users from specific hosts specified at creation time. The resulting access privileges will be surfaced using the CIM Privilege model when that is supported by SMI-S.

Table 969: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (Sample on Capabilities)

Property	Flags	Type	Description & Notes
WritePolicy		uint16	Indicates whether writes to the shared element will be handled synchronously or asynchronously by default. This policy may be overridden or surfaced using the CIM Policy model when that is supported by SMI-S.
AccessPoints		uint16	Specifies the service access points that are available to this FileShare by default (to be used by clients for connections). These default access points can always be overridden by the privileges explicitly defined by a supported authorization mechanism(s). Any ServiceAccessPoints that actually connect to this share will be associated to it by CIM_SAPAvailableForElement. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.

8.2.8.7.9.6 CIM_ExportedFileShareSetting

This element represents the client defined configuration settings of a File Share intended for Export. It is created as a result of a CreateExportedShare extrinsic.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Extrinsic(s): ModifyExportedShare
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 970: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (On FileShare)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for an exported or owned FileShare Setting.
ElementName		string	A client defined user-Friendly Name for this Setting element.
FileSharingProtocol		uint16	This, together with ProtocolVersions, identifies the protocol for file-sharing that is supported by the Computer System.
ProtocolVersions		string[]	This, together with FileSharingProtocol, identifies the version of the protocol for file-sharing that is supported by the Computer System.
Optional Properties/Methods			
InitialEnabledState		uint16	This indicates the enabled/disabled states initially set for a created file share element.

Table 970: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (On FileShare)

Property	Flags	Type	Description & Notes
OtherEnabledState		string	A vendor-specific description of the initial enabled state of a created fileshare if InitialEnabledState=1("Other").
DefaultReadWrite		uint16	Indicates the default privileges that are supported for read and write authorization to the newly created fileshare. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
DefaultExecute		uint16	Indicates the default privileges that are supported for execute authorization to the newly created fileshare. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
ExecuteSupport		uint16	Indicates if the sharing mechanism provides specialized support for executing an element shared through this fileshare (for instance, does it provide paging support for text pages).
DefaultUserIdSupported		uint16	Indicates whether the FileShare will use a default user id to control access to the share if the id of the importing client is not provided. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.
RootAccess		uint16	Indicates whether the FileShare will support default access privileges to administrative users from specific hosts specified at creation time. The resulting access privileges will be surfaced using the CIM Privilege model when that is supported by SMI-S.
WritePolicy		uint16	Indicates whether writes to the shared element will be handled synchronously or asynchronously by default. This policy may be overridden or surfaced using the CIM Policy model when that is supported by SMI-S.

Table 970: SMI Referenced Properties/Methods for CIM_ExportedFileShareSetting (On FileShare)

Property	Flags	Type	Description & Notes
AccessPoints		uint16	Specifies the service access points that are available to this FileShare by default (to be used by clients for connections). These default access points can always be overridden by the privileges explicitly defined by a supported authorization mechanism(s).Any ServiceAccessPoints that actually connect to this share will be associated to it by CIM_SAPAvailableForElement. The resulting access privileges will be surfaced using the CIM_Privilege model when that is supported by SMI-S.

8.2.8.7.9.7 CIM_FileExportCapabilities

This element represents the management Capabilities of the File Export Service.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 971: SMI Referenced Properties/Methods for CIM_FileExportCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	An opaque, unique id for the capabilities of a File Export Service.
ElementName		string	A provider supplied user-Friendly Name for this Capabilities element.
FileSharingProtocol		uint16[]	The Service can be associated with multiple ExportedFileShareCapabilities that are keyed by the FileSharingProtocol property -- the Configuration capabilities lists all of the supported protocols in this FileSharingProtocol array. Duplicate entries are permitted because the corresponding entry in the ProtocolVersions array property indicates the supported version of the protocol.
ProtocolVersions		string[]	An array listing the versions of the protocol specified in the corresponding entry of the FileSharingProtocol array property.
SupportedSynchronousMethods	N	uint16[]	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called synchronously. A supported method shall be listed in this property or in the SupportedAsynchronousMethods property.

Table 971: SMI Referenced Properties/Methods for CIM_FileExportCapabilities

Property	Flags	Type	Description & Notes
SupportedAsynchronousMethods	N	uint16[]	The Service supports a number of extrinsic methods -- this property identifies the ones that can be called asynchronously. Note: A supported method shall be listed in this property or in the SupportedSynchronousMethods property.
Optional Properties/Methods			
InitialEnabledState		uint16	This represents the state of initialization of a FileShare on initial creation.

8.2.8.7.9.8 CIM_FileExportService

The File Export Service provides the methods to create and export file elements as shares.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 972: SMI Referenced Properties/Methods for CIM_FileExportService

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

8.2.8.7.9.9 CIM_FileShare

Represents the sharing characteristics of a particular file element.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Extrinsic(s): ModifyExportedShare
Deleted By : Extrinsic(s): ReleaseExportedShare

Class Mandatory: true

Table 973: SMI Referenced Properties/Methods for CIM_FileShare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	A unique id for the FileShare element.
SharingDirectory		boolean	Indicates if the shared element is a file or a directory. This is useful when importing but less so when exporting.

8.2.8.7.9.10 CIM_HostedService

In this subprofile, associates the File Export Service to the hosting Computer System.
Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 974: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting Computer System.
Dependent		CIM_Service	The FileExportService.

8.2.8.7.9.11 CIM_HostedShare

Represents that a FileShare element (sharing a LogicalFile or Directory) is hosted by a Computer System.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Static
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 975: SMI Referenced Properties/Methods for CIM_HostedShare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_Share	The FileShare that is hosted by a Computer System
Antecedent		CIM_System	The Computer System that hosts a FileShare.

8.2.8.7.9.12 CIM_LogicalFile

A LogicalFile (or Directory subclass) is available for export via a fileshare hosted on a ComputerSystem.
Created By : Extrinsic(s): CreateExportedShare
Modified By : External
Deleted By : Extrinsic(s): ReleaseExportedShare

Class Mandatory: true

Table 976: SMI Referenced Properties/Methods for CIM_LogicalFile

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CSCreationClassName		string	CIM Class of the Computer System that hosts the Filesystem of this File.
CSName		string	Name of the Computer System that hosts the Filesystem of this File.
FSCreationClassName		string	CIM Class of the LocalFileSystem on the Computer System that contains this File.
FSName		string	Name of the LocalFileSystem on the Computer System that contains this File.
CreationClassName		string	CIM Class of this instance of Logical-File.
Name		string	Name of this LogicalFile.

8.2.8.7.9.13 CIM_ServiceAffectsElement

Associates the File Export Service to the elements that the service affects (such as a FileShare configured for exporting a LogicalFile).
Created By : Extrinsic(s): CreateExportedShare
Modified By : Static
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 977: SMI Referenced Properties/Methods for CIM_ServiceAffectsElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AffectedElement		CIM_ManagedElement	The FileShare.
AffectingElement		CIM_Service	The File Export Service.
ElementEffects		uint16[]	In the context of this subprofile, the possible element effects are either Exclusive use or Element Integrity. We allow Other to support vendor exten-sions.
OtherElementEffectsDescriptions		string[]	A description of other element effects that this association might be exposing.

8.2.8.7.9.14 CIM_SettingAssociatedToCapabilities

Represents the association between a ExportedFileShareCapabilities and a supported ExportedFileShareSetting element.
Created By : Static
Modified By : Static
Deleted By : Static

Class Mandatory: true

Table 978: SMI Referenced Properties/Methods for CIM_SettingAssociatedToCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Capabilities	The ExportedFileShareCapabilities element.
Dependent		CIM_SettingData	The ExportedFileShareSetting element.

8.2.8.7.9.15 CIM_SharedElement

In this subprofile, represents the identity association between an exporting FileShare element and the actual shared LogicalFile or Directory. The implication is that these are the same element even though represented by two SMIS objects.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Static
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 979: SMI Referenced Properties/Methods for CIM_SharedElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemElement		CIM_LogicalElement	The LogicalFile or Directory element that is being shared.
SameElement		CIM_Share	The exported FileShare that represents the element being shared.

8.2.8.7.9.16 CIM_SharedElementRoot

In this subprofile, represents the association between an exporting FileShare element and a directory or file in the file namespace of the hosting ComputerSystem on which the LocalFileSystem containing the shared LogicalFile or Directory element is mounted.
Created By : Extrinsic(s): CreateExportedShare
Modified By : Static
Deleted By : Extrinsic(s): ReleaseExportedShare
Class Mandatory: true

Table 980: SMI Referenced Properties/Methods for CIM_SharedElementRoot

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_Share	The FileShare that represents the element being exported.
Antecedent		CIM_LogicalElement	The element that indicates the mount-point of the FileSystem containing the shared LogicalFile or Directory.

8.2.8.7.10 Related Standards

Table 981: Related Standards for File Export Manipulation

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.8.8 Pool Management Policy Subprofile

8.2.8.8.1 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 6.4, "Policy".

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.

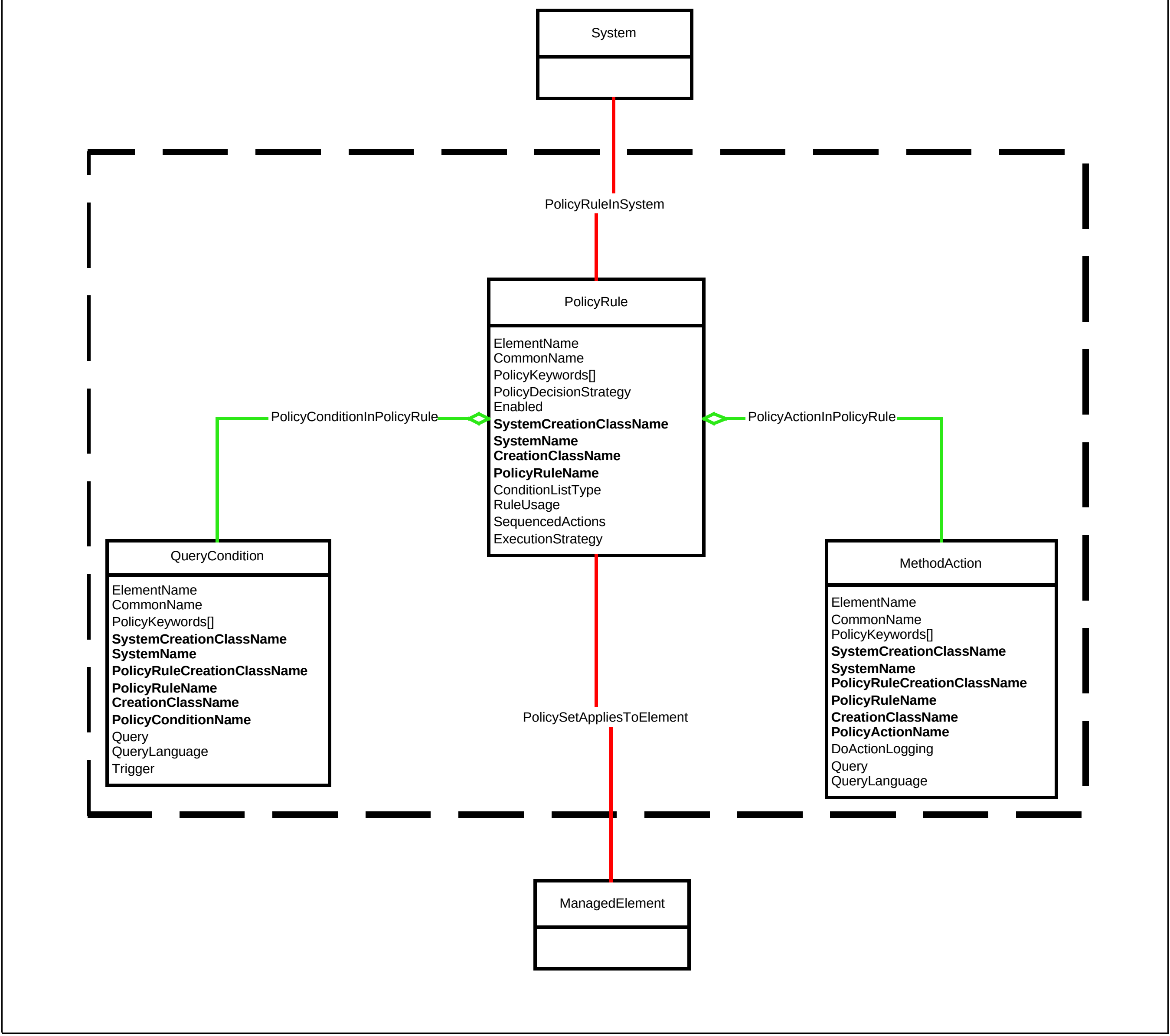
8.2.8.8.1.1 Instance Diagrams

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 and 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 138: "Basic Pool Management Instance Diagram"

Figure 138: Basic Pool Management Instance Diagram



The Managed Element in this case is the concrete StoragePool that is desired to be managed.

8.2.8.8.1.2 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(primoridal) = hostedpool.PartComponent
    and OBJECTPATH(system) = hostedpool.GroupComponent
    and OBJECTPATH(system) = hosted.Antecedent
    and OBJECTPATH(service) = hosted.Dependent
    and service ISA "CIM_StorageConfigurationService"
```

8.2.8.8.1.3 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
```

8.2.8.8.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.8.3 Cascading Considerations

Not defined in this standard.

8.2.8.8.4 Supported Subprofiles and Packages

Table 982, “Pool Management Policy Subprofiles” list the subprofiles available to this subprofile.

Table 982: Pool Management Policy Subprofiles

Dependency Name	Type	Mandatory	Notes
Policy Subprofile	subprofile	Yes	

8.2.8.8.5 Methods of the Profile

8.2.8.8.5.1 Extrinsic Methods of the Profile

None.

8.2.8.8.5.2 Intrinsic Methods of the Profile

Table 983, “Static Policy Instance Methods” 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 983: 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 984, “Instance Methods of Dynamic Rules and Static Conditions and Actions” 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 984: 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 985, “Dynamic Policy Instance Methods” 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 985: 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
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)

Table 985: Dynamic Policy Instance Methods (Continued)

Method	Created Instances	Deleted Instances	Modified Instances
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

8.2.8.8.6 Client Considerations and Recipes

See 6.4.5, "Policy Recipes" for usage of PolicySetAppliesToElement.

8.2.8.8.7 Required CIM Elements

Table 986: Required CIM Elements

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	

Table 987: Instance Creation, Deletion or Modification for Pool Management Policy Subprofile Classes

Class / Association	Creation	Deletion	Modification
CompoundPolicyAction			
CompoundPolicyCondition			

Table 987: Instance Creation, Deletion or Modification for Pool Management Policy Subprofile Classes

Class / Association	Creation	Deletion	Modification
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			

8.2.8.8.8 Classes Used in the Profile

The *flags* columns of the tables below employ the values detailed in Table 988, “Valid Flag Values”:

Table 988: 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
M	Modifiable	The property can be modified by ModifyInstance
N	Null Okay	Client may set this property to null

8.2.8.8.8.1 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 989, “Properties for MethodAction”.

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 DeleteInstace intrinsic method.

Modified by: “Static” MethodActions shall not be modified. However, client defined MethodActions may be modified.

ConditionalRequirements: none

Table 989: Properties for MethodAction

Property	Type	Flags	Description / Notes
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 Policy-Action 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.
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 parame-ters to that method. See 8.2.8.8.1.2 for the actual query string text.
QueryLanguage	Uint16	R	This defines the query language being used and shall be set to « 2 » (CQL).

8.2.8.8.8.2 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 990: 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\".
ReturnValue	string		When PreCall is FALSE, ReturnValue contains a string representa-tion 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 Mapping Specification*-based Instance Path to the MethodAction instance that produced this result.

8.2.8.8.8.3 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 991, “Properties for PolicyActionInPolicyRule”.

Created by: PolicyAction can be "Static" (predefined by the provider).

Deleted by: “Static” PolicyActions shall not be deleted.

Modified by: “Static” PoilicyActions shall not be modified.

Conditional Requirements: none

Table 991: 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.

EXPERIMENTAL

8.2.8.8.8.4 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 992, “Properties for PolicyCapabilities”.

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 992: 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 { “” }

EXPERIMENTAL

8.2.8.8.8.5 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 993, “Properties for PolicyConditionInPolicyRule”.

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 993: Properties for PolicyConditionInPolicyRule

Property	Type	Flags	Description / Notes
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.

8.2.8.8.8.6 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 995, “Properties for PolicyRuleInSystem”.

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 994: 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

Table 994: Properties for PolicyRule

Property	Type	Flags	Description / Notes
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" }
Enabled	uint16	R	Indicates whether this PolicySet is administratively enabled, administratively disabled, or enabled for debug. The \"Enabled-ForDebug\" 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 Policy-Group 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 -- I don't care about the order. The default value is 3 (\"DontCare\"). Values { "Mandatory", "Recommended", "Dont Care" }

Table 994: Properties for PolicyRule

Property	Type	Flags	Description / Notes
ExecutionStrategy	uint16		ExecutionStrategy defines the strategy to be used in executing the sequenced actions aggregated by this PolicyRule. There are three execution strategies: Do Until Success - execute actions according to predefined order, until successful execution of a single action. 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. Do Until Failure - execute actions according to predefined order, until the first failure in execution of an action instance. Values { "Do Until Success", "Do All", "Do Until Failure" }

8.2.8.8.8.7 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 995: 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.

8.2.8.8.8.8 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 996: 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.

8.2.8.8.8.9 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 997: 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
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 Policy-Condition is defined.
PolicyRuleCreationClassName	String	R	For a rule-specific PolicyCondition, the CreationClassName of the PolicyRule object with which this Condition is associ-ated. For a reusable Policy Condition, a special value, 'NO RULE', should be used to indicate that this Condition is reus-able and not associated with a single PolicyRule.
PolicyRuleName	String	R	For a rule-specific PolicyCondition, the name of the Poli-cyRule 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.

Table 997: Properties for QueryCondition

Property	Type	Flags	Description / Notes
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.

8.2.8.8.8.10 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 998: 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.

Table 998: Properties for QueryConditionResult

Property	Type	Flags	Description / Notes
QueryConditionPath	string	R	This shall be a fully qualified, *WBEM URI Mapping Specification*-based Object Name to the QueryCondition instance that produced this result.

8.2.8.8.9 Dependencies on Other Standards

Table 999: Pool Management Policy Profile Subprofile Standards 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

8.2.8.8.9.1 CIM Server Requirements

Functional Profiles

Table 1000: 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

Extrinsic Methods

None.

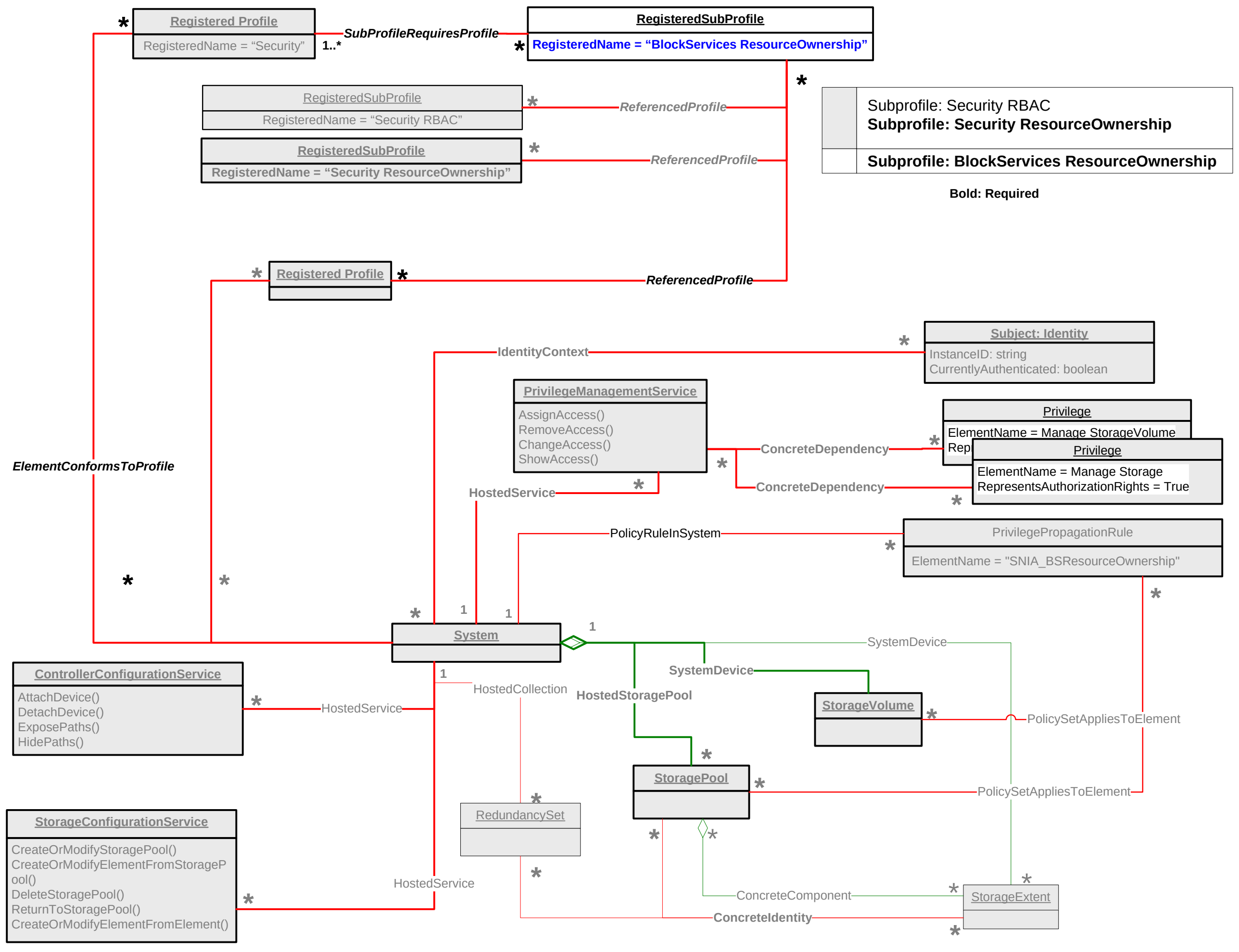
Discovery

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.

8.2.8.9 Resource Ownership Subprofile

8.2.8.9.1 Description

Figure 139: Resource Ownership for Block Services


Registered Profile
RegisteredName = "Security"
SubProfileRequiresProfile
1..*
RegisteredSubProfile
RegisteredName = "BlockServices ResourceOwnership"
RegisteredSubProfile
RegisteredName = "Security RBAC"
ReferencedProfile
RegisteredSubProfile
RegisteredName = "Security ResourceOwnership"
ReferencedProfile
Subprofile: Security RBAC
Subprofile: Security ResourceOwnership
Subprofile: BlockServices ResourceOwnership
Bold: Required
Registered Profile
ReferencedProfile
Subject: Identity
InstanceID: string
CurrentlyAuthenticated: boolean
IdentityContext
PrivilegeManagementService
AssignAccess()
RemoveAccess()
ChangeAccess()
ShowAccess()
Privilege
ElementName = Manage StorageVolume
Rep
Privilege
ElementName = Manage Storage
RepresentsAuthorizationRights = True
ConcreteDependency
ConcreteDependency
ElementConformsToProfile
HostedService
PolicyRuleInSystem
PrivilegePropagationRule
ElementName = "SNIA_BSResourceOwnership"
System
SystemDevice
SystemDevice
StorageVolume
HostedStoragePool
HostedCollection
ControllerConfigurationService
AttachDevice()
DetachDevice()
ExposePaths()
HidePaths()
HostedService
PolicySetAppliesToElement
StoragePool
PolicySetAppliesToElement
RedundancySet
StorageConfigurationService
CreateOrModifyStoragePool()
CreateOrModifyElementFromStoragePool()
DeleteStoragePool()
ReturnToStoragePool()
CreateOrModifyElementFromElement()
HostedService
ConcreteComponent
ConcreteIdentity
StorageExtent

The Block Services Resource Ownership common subprofile[3] 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.

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.

3.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.

There is one Privilege to "Manage StorageVolume" and a superset of that to "Manage Storage". Each is described in Table 1001, “Block Service Management Rights”.

Table 1001: 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 MehtodAction 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.

8.2.8.9.1.1 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"s 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 140: "ServiceAffectsElement associations for ResourceOwnership".) 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 141: "AuthorizedPrivilege associations for ResourceOwnership".) 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 140: ServiceAffectsElement associations for ResourceOwnership

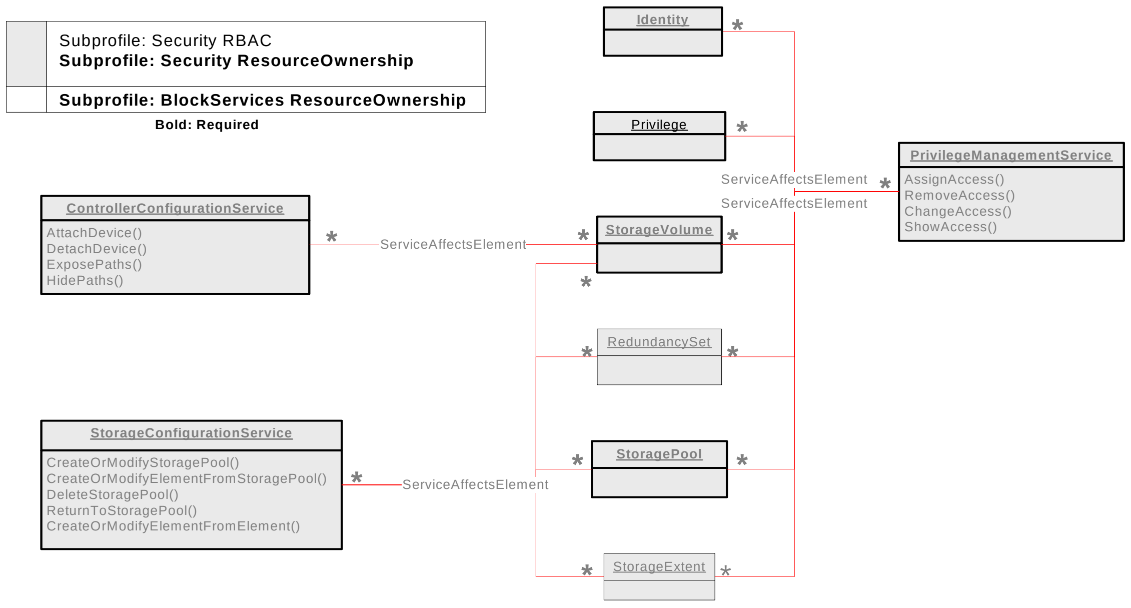
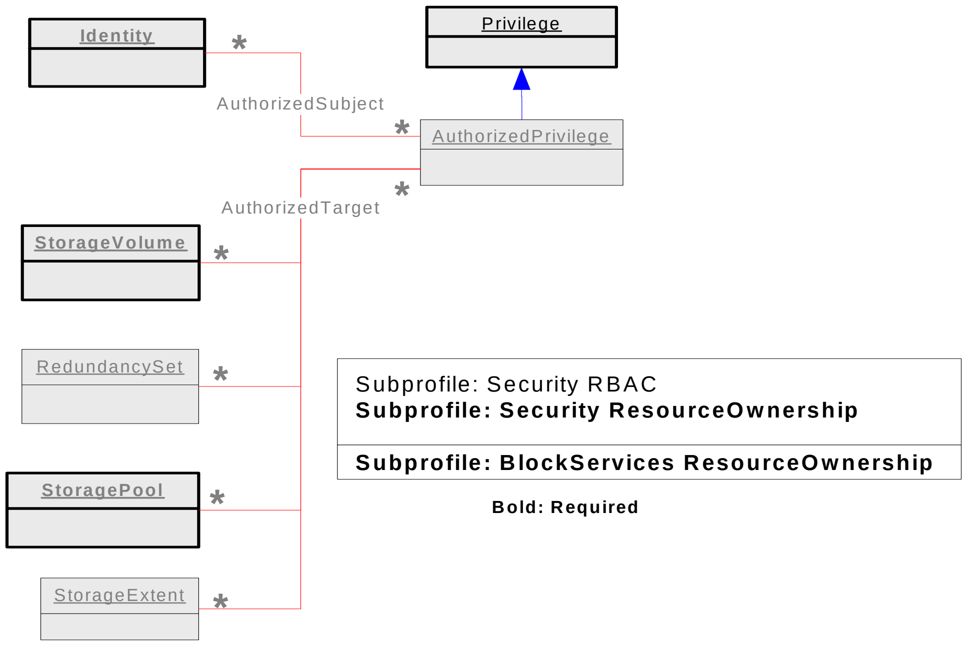


Figure 141: AuthorizedPrivilege associations for ResourceOwnership



- 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.

8.2.8.9.1.2 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 ConcreteIdentity to a StorageExtent aspect. (For instance when a Raid 5 extent is used as a StoragePool.)
3) Privilege restrictions on a StorageExtent propagate across ConcreteIdentity 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.

8.2.8.9.2 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 ConcreteIdentity 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 3rd Party Authentication 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.

8.2.8.10 Block Services Package

8.2.8.10.1 Description

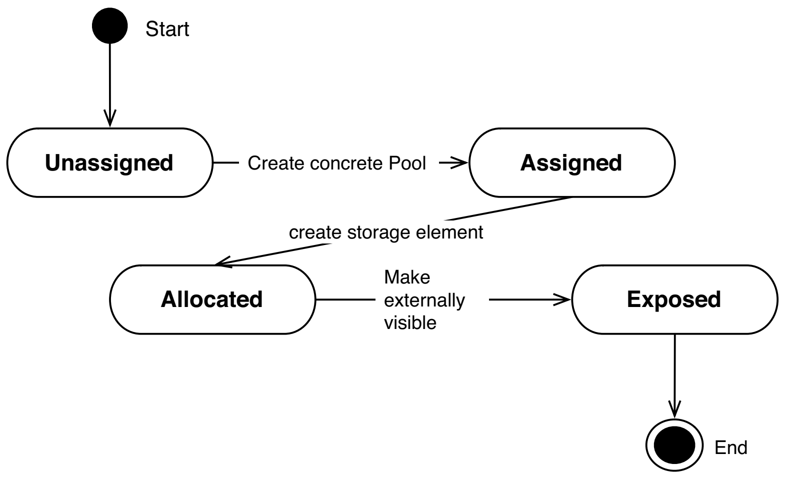
Many types of devices or applications provide their storage capacity through block-based IO. These devices or applications provide block services to other devices or applications, block consumers, that are external to them. This subprofile defines a standard expression of existing storage capacity, the assignment of capacity to StoragePools, and allocation of capacity for use for 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.

Figure 142: "Storage Capacity State" illustrates the state of a block of storage that is discussed in this subprofile.

Figure 142: Storage Capacity State



A given block of capacity within a storage device or application has a state in this subprofile. The storage elements, StorageVolume and LogicalDisk, described in this section, are a grouping of blocks. It is useful put terms to the state transitions that blocks groups by this storage elements to differentiate the steps in the storage configuration process. A unconfigured storage device or application may have none of its capacity organized into concrete pools. All blocks within that device or application have an unassigned state. Once a block is a member of a concrete pool, capacity can be said to be assigned. Once a block is a member of a storage element, like a Volume or Logical Disk, the 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.

Storage Pools

A StoragePool is collection of storage capacity with a given set of capabilities. A Pool has certain ‘StorageCapabilities’, which indicate the range of 'Quality of Service' requirements that can be applied to objects created from the Pool.

StoragePools are a mandatory part of modeling disk storage systems that support this package. However, user manipulation of StoragePools is optional and may not be supported by any given disk storage system. This subprofile defines the support mandatory if the storage system exposes functions for creating and modifying storage pools.

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

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

There are two properties on StoragePool that describe the size of the ‘underlying’ storage. TotalManagedStorage describes the total storage in the pool and RemainingManagedStorage describes the storage currently remaining in the pool.

Primordial Pool

The Primordial Pool is a type of StoragePool. This Pool may contain unformatted or unprepared capacity. Or this type of Pool may simply contain unassigned capacity. Storage capacity is drawn from the Primordial StoragePool to create concrete StoragePools. The Primordial StoragePool aggregates storage capacity that has not been assigned to a concrete StoragePool. StorageVolumes and LogicalDisks are allocated from concrete StoragePools.

At least one primordial Pool shall always exists 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 Pools.

Primordial Pool can be used to determine the amount of capacity left on the block storage system; that is, not already assigned to a concrete StoragePool.

Concrete Pool

The Concrete Pool is a type of StoragePool. This concrete Pool is the only type of Pool created or modified by behaviors described in this Package. A concrete Pool is used to subdivide the storage capacity available in a block server for the creation or modification of StorageVolumes and LogicalDisks. Concrete Pools can be used to assign capacity based QoS or other factors like cost per megabyte or ownership of storage. A concrete Pool may aggregate the capacity of one or many RAID groups or RAID ranks. The RAID group or rank may be created when the StorageVolume or LogicalDisk is created.

Blocks, Metadata, and Capacity Reported

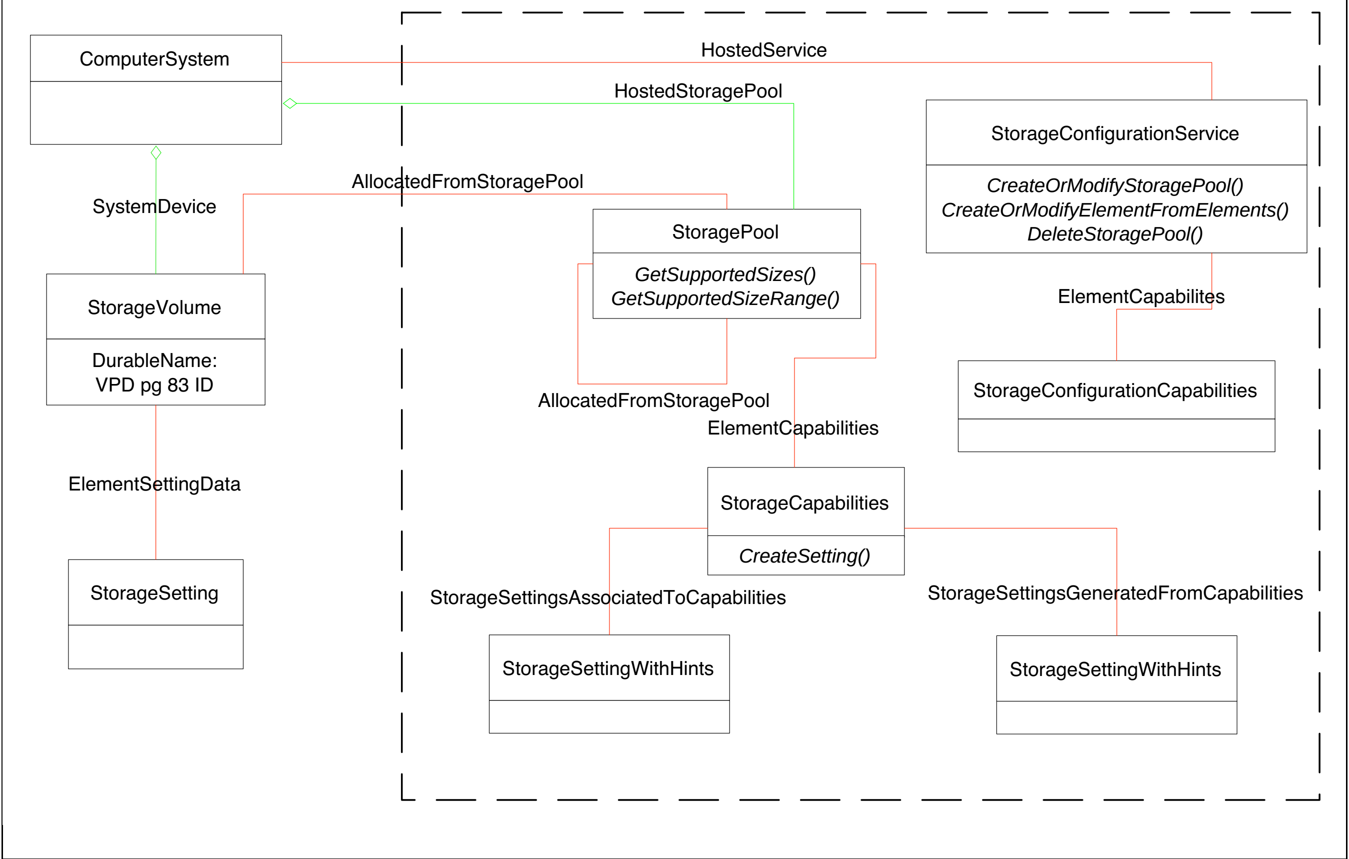
This subprofile uses the term “metadata” to signify the capacity drawn for the creation of stripes, data copies, and the like. The capacity removed for such constructs when creating storage elements, like Pools, Volumes, and LogicalDisks is reported in the difference between the capacity of the parent Pool 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 new storage element or expand an existing storage element. One may use this profile to calculate capacity used for metadata.

Additionally, there is likely to be a difference between the capacity one can calculate adding up all the capacity of the disks, as reported by the manufacturers, or LUNs consumed by a block server, as reported by the block server that exposes them, and the capacity that can be used to create other storage organizations or constructs from this capacity, like Pools, Volumes, and Logical Disks. For example, this difference in capacity can be used for disk formatting and the like. The difference in the capacity of the primordial Pool and the capacity used to produce the primordial Pool is not reported through this subprofile.

Pool Management Instance Diagram

Figure 143: "Pool Manipulation Instance Diagram" provide an instance diagram for pool manipulation.

Figure 143: Pool Manipulation Instance Diagram



Pool, Volume and Logical Disk Manipulation

Storage Volumes 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.

Logical Disks 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 volume manager. In the CIM class hierarchy, they are also a subclass of a StorageExtent. Logical Disk are a mandatory part of modeling a host based Volume Managers.

Storage Volumes and Logical Disks are consumable storage capacity. That is these storage elements are the only extents that are available to consumers of the block service and a block device.

However, creation or modification of Volumes or Logical Disks from Pools 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 storage volumes from storage pools.

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

- CreateOrModifyStoragePool: Create a pool of storage with some set of Capabilities defined by the input StorageSetting. The source of the storage can be other Pool(s) or Extents. Alternatively an existing Pool can be modified. The method can also be used to modify a Pool to increase or decrease its capacity.
- DeleteStoragePool: Delete a Pool and return the freed up storage to the underlying entities.

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

- CreateOrModifyElementFromStoragePool: Create Volume or Logical Disk, possibly with a specific StorageSetting, from a source Pool. The method can also be used to modify a Volume or Logical Disk to increase or decrease its capacity.
- CreateOrModifyElementFromElements: Create a Volume or Logical Disk using Component Extents of a parent and source Pool. The method can also be used to alter the set of member Extents of a Volume or Logical Disk or change the consumption of an existing set of member Extents.
- ReturnToStoragePool: Return an Element previously created with CreateOrModifyElementFromStoragePool to the originating StoragePool.

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

- CreateSetting: Creates a setting that is consistent with the StorageCapabilities and may be modified before use in creating a Pool, Volume, or Logical Disk.
- GetSupportedStripeLengths and GetSupportStripeLengthRange: Returns the possible stripe lengths for that capability
- GetSupportedStripeDepths and GetSupportedStripeDepthRange: Returns the possible strip depths for that capability
- GetSupportedParityLayouts: Returns the possible parity layouts, rotated or non-rotated, for that capability.

See 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration" for details on the associations from Setting to Capabilities.

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. This method can also be used to return the discontiguous capacity in the Pool not yet assigned to a concrete Pool or allocated to a storage element.
- GetSupportedSizeRange: Returns the range of possible sizes given a goal.
- GetAvailableExtents: Returns an array of extent references which match a given goal and are components of the Pool and are not already members of an existing consumable storage element, child Pool, Volume, or LogicalDisk.

Declaring Storage Configuration Options

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

Table 1002, “Supported Actions to Method Mapping” defines how the SupportedSynchronousActions

Table 1002: Supported Actions to Method Mapping

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

and SupportedAsynchonousActions array values map to methods in the StorageConfigurationService class. The presence of an ‘Action’ from Table 1002 in the SupportedSynchronousActions array means that the associated ‘SCS Method’ does not produce a Job by side-effect. Likewise, the presence of an ‘Action’ from Table 1002 in the SupportAsynchronousActions array means 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, then the implementation may or may not produce a Job by side effect.

The SupportedStorageElementTypes array declares what storage elements may be created or modified by this implementation. Since Pools are a mandatory part of this implementation, a client may assume that the support of the Pools methods also implies support of creation or modification of storage elements of type Pool.

The SupportedStoragePoolFeatures array declares what Pool behavior is supported.

- 2 “InExtents” means that a Pool may be created from Extents.
- 3 “Single InPools”, 4 “Multiple InPools"
  A Pool may be the source of capacity of for Pool creation or modification. In other words, concrete Pools may be created from other Pools.

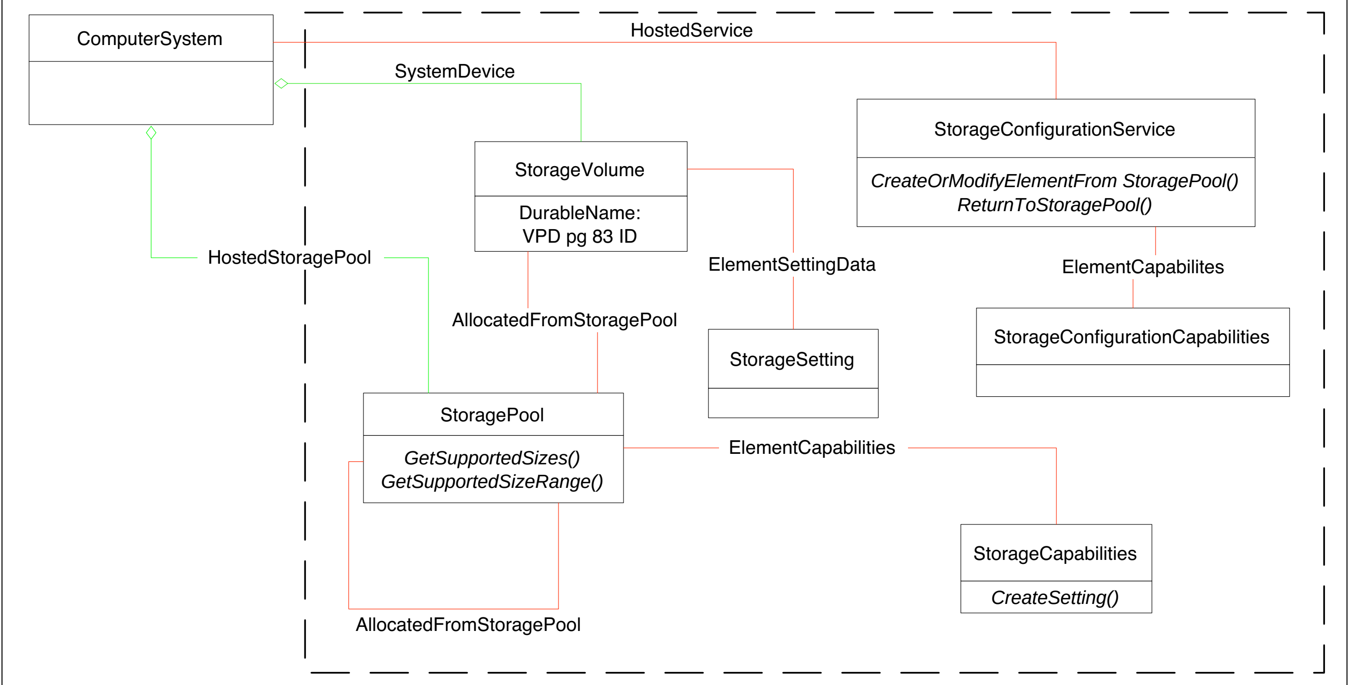
The SupportedStorageElementFeatures array declares what special features the configuration methods support.

- 2 "StorageExtent Creation", 4 "StorageExtent Modification"
  These feature elements declare the ability of the SMI-S implementation to create or modify StorageExtents respectively.
- 3 "StorageVolume Creation", 5 "StorageVolume Modification"
  These feature elements declare the ability of the SMI-S implementation to create or modify StorageVolumes respectively.
- 8 "LogicalDisk Creation", 9 "LogicalDisk Modification"
  These feature elements declare the ability of the SMI-S implementation to create or modify LogicalDisks 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 either the creation or modification of concrete Pools from a single Pool only or from multiple input Pools

Volume Creation Instance Diagram

Figure 144: "Volume Creation Instance Diagram" provides an instance diagram from volume creation.

Figure 144: Volume Creation Instance Diagram



Backwards Compatibility

This package is designed to be backward compatible with the “Pool Manipulation Capabilities, and Settings” Subprofile and the “LUN Creation” Subprofile from IS24775-2006, *Storage Managemen*t (SMI-S 1.0). These subprofiles are deprecated. In fact, this 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 IS24775-2006, *Storage Managemen*t 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 8.2.4.1, "Server Profile"

Capacity Management

The capacity characteristics of many storage system vary greatly in the cost and performance. Additionally, the capacity may need to be partitioned by these and other factors. StoragePool provide a means to aggregate this storage by characteristics determined by the storage administrator or determined at the factory when the storage system is assembled.

A Storage Pool is an aggregation of storage suitable for configuration and allocation or “provisioning”. However, it may have been preformatted into a form (such as a RAID group) that makes volume creation easier.

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

A StoragePool has a set of capabilities held in the StorageCapabilities class that reflect the configuration parameters that are possible for element created from this pool. The StorageCapabilities define, in terms common across all storage system implementation, what characteristics an administrator can expect from the storage capacity. These capabilities are expressed in ranges. The storage implementation has the choice to delineate the capabilities and define the ranges of these capabilities as appropriate. Some implementation 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 storage pools can change over time as well.

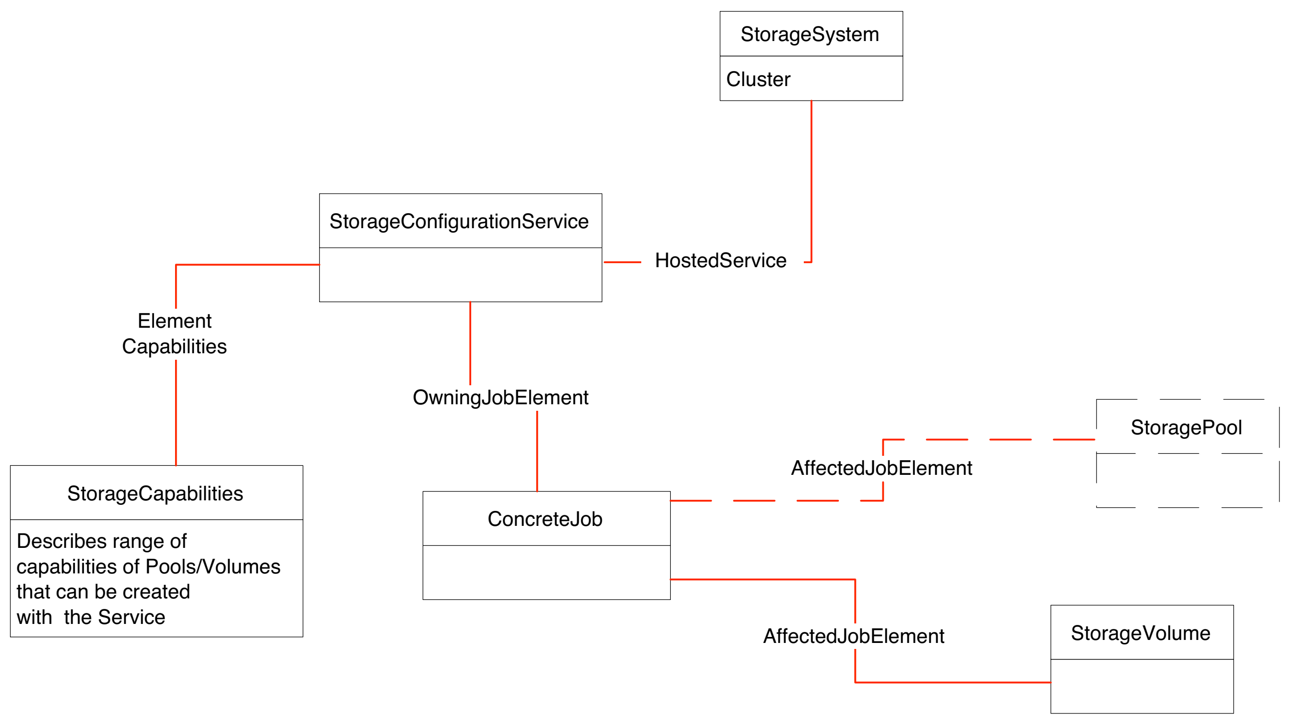
These storage capabilities are given the scope of the storage system when they are associated to the StorageConfiguratonService or the scope of a single StoragePool when associated to same. The capabilities expressed at the service scope is equal to the union of all Primordial StoragePools capabilities. 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 having done so 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 system can be inventoried and managed. Note that the storage system will differ in other significant ways, and these characters can also be the basis for capacity pooling decisions. A sample configuration is illustrated in Figure 145: "Storage Configuration"

Figure 145: Storage Configuration



See 8.2.1.7, "Job Control Subprofile" for details on the usage of the StorageConfigurationJob, AssociatedStorageConfigurationJob, and OwningJobElement associations.

The definition of storage capabilities in this way intentionally avoids vendor specific details of volume configuration such as RAID types. Although RAID types imply performance and availability levels, these levels can't be easily compared between vendor implementations - particular in comparisons with reliability of non-RAID storage (i.e., certain virtualization appliances). Furthermore, 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 pool can support storage configured with No Single Points of Failure within the storage system. This does not include the path from the system to the host.
- Data Redundancy. This describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and mirroring where 2 or more copies are maintained.
- Package Redundancy. This describes how many physical components (packages), like disk spindles, can fail without data loss (including a spare, but not more than a single global spare). Examples would be 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. This is also known as the number of 'members' or 'columns'. For non-striped organizations (e.g., mirror or JBOD), the ExtentStripeLength shall be one.
- UserDataStripeDepth describes the number of bytes forming a strip in common striping-based storage organizations. The strip is defined as the size of the portion of a stripe that lies on one extent. Thus ExtentStripeLength times UserDataStripeDepth will yield the size of one stripe of user data.
- ParityLayout specifies whether a parity-based storage organization is using rotated or non-rotated parity.

An example of what the Package Redundancy and Data Redundancy means in terms of RAID levels is defined in Table 1003.

Mapping of RAID levels to Data Redundancy, Package Redundancy

Table 1003: RAID Mapping Table

RAID Level	Package Redundancy	Data Redundancy	Extent Stripe Length	User Data Stripe Depth	Parity Layout
JBOD	0	1	1	NULL	NULL
0 (Striping)	0	1	2 to N	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)	1	1	3 to N	Vendor Dependent	2
6, 5DP	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

Table 1003, “RAID Mapping Table” was produced using generally available definitions of RAID levels. The character ‘N’ represents the variable for the total number of StorageExtents. ‘DP’ is double parity. ‘3/5’ is RAID 5 implementations that are sometimes called RAID 5.

It is the nature of RAID technology that even though the RAID Level is named 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 means for a particular implementation and instead defines the storage provided in service 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 exactly with 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, it may sense for the capability be expressed in broader ranges for more flexible storage systems.

For those existing StorageSetting instances whose "ChangableType" 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 include, 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 (optional) 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".

Storage Setting Associations to Storage Capabilities

Storage Setting instances can be associated its the parent StorageCapabilities instance through either the StorageSettingsAssociatedToCapabilities and 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 setting are used to define the possible configurations of Storage Pools, Storage Volumes or Logical Disks where the number of possibilities are small because the capabilities of the device itself is likewise limited. When a Capabilities is created by side-effect of creating a concrete Pool, this type of Storage Setting may also be created or an existing Storage Setting associated to this new Capabilities as well. A client can use the StorageSettingsAssociatedToCapabilities association to find the default goal for the Capabilities, 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, it is transient (e.g., ChangeableType = 1). This means that the Setting instance may not remain for long. This Setting may be removed from the CIMOM after reboots and simply after some 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.

Read-Only Model Requirements

This package defines that classes and behavior to both express the assignment and allocation of storage capacity as well as 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 be a *read-only* implementation. In such a model, only classes listed in Table 1004, “Classes Required In Read-Only Implementation” are required. Classes not explicitly stated are not required for *read-only* implementations.

Table 1004: 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

Extent Conservation

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

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

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

- Allocating capacity from a Pool shall not reduce the total size of the Pool.
- A given extent instance shall not be a component of more than one Pool. However, an given block may be accounted for in the range of blocks represented by more than one extent instance. In other words, a given block may be associated to more than one Pool.
- The use of all or some of the capacity of an extent directly, by passing the reference to the extent in a method call, or indirectly, by passing the size of the desired storage element, shall result in the creation of new Extents that are components of the new Volume or LogicalDisk.
- Any remaining capacity from the extent shall be represented by a new component extent of the source Pool that is based on the partitioned extent. This extent is called a *remaining extent.*
    1) If the Size requested is smaller than the total consumable size of the Extents or Pools, then these resources are partially used. In this case, the model shall reflect what capacity was used and what capacity remains of the Extents or Pools passed as arguments to CreateOrModifyStoragePool and CreateOrModifyElementFromElements methods.
    2) Once the capacity represented by a remaining extent is used to assign or allocate capacity, the remaining extent either shrinks in size or is removed from the model. A remaining extent shall not be molded to have other extents based on it.
- An extent 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 extent in the first place.

Figure 146, Figure 147, and Figure 148 illustrate the use of Extents to represent the partitioning of an extent's capacity. An implementation of this subprofile may implement the 8.2.8.16, "Extent Composition Subprofile" as well. Extent Conservation requires the instantiation of additional Component Extents that represent remaining space. These Extents are in addition to those modeled by the Extent Composition Subprofile. Available extents, (including remaining space Extents), which meet specific goal requirements, are found using the GetAvailableExtents method of the StorageConfigurationService.

The modeling of remaining Extents is not within the scope of the Extent Composition Subprofile. However, the recipes written for the Extent Composition Subprofile will tolerate these additional extents. The modeling of free/unused extents is defined only in the Extent Conservation section of Block Services package.

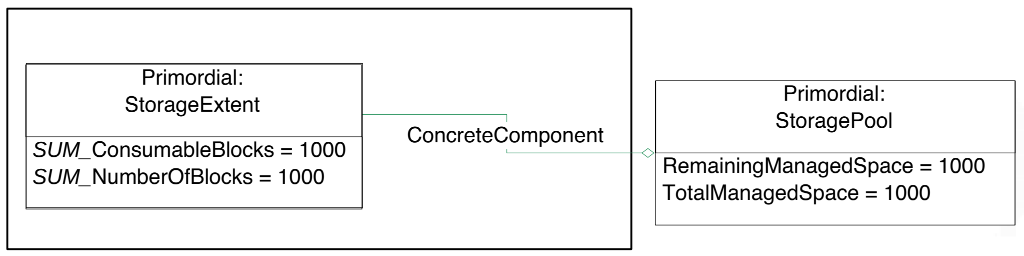
Support of the GetAvailableExtents and CreateOrModifyElementFromElements methods are not required by the Block Services package nor the Extent Composition Subprofile. An implementation may support the representation of StorageVolume or LogicalDisk structure through the Extent Composition Subprofile,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

If a implementation supports the GetAvailableExtents and CreateOrModifyElementFromElements methods and the Block Services Package, then it shall also implement the Extent Composition Subprofile (see 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration"). Additionally, the implementation shall implement both methods if it implements one of the methods or neither of the methods.

Figure 146, Figure 147, and Figure 148 represent how Extents are partitioned to represent the partial usage of the capacity in the construction of a concrete Pool and a concrete Volume. 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 group those classes that are defined in the Extent Composition Subprofile.

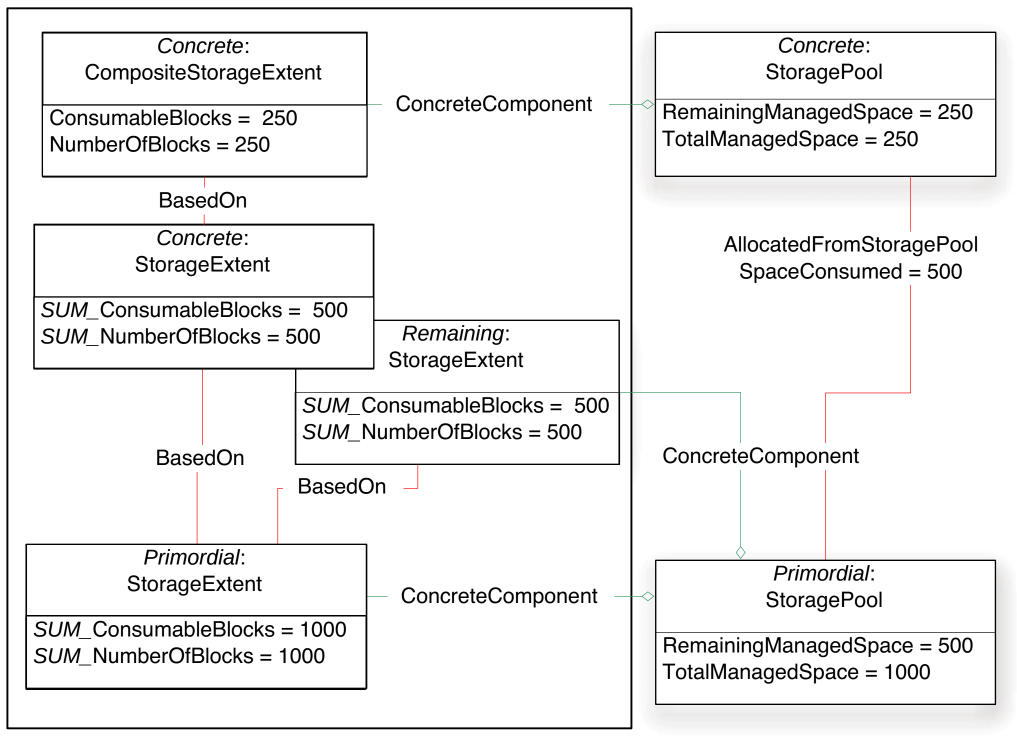
The initial state in Figure 146: "Extent Conservation, Step 1" starts with a Primordial Pool that is realized by a Primordial Extent. This extent is part of the initial capacity of the device or added to the device in process defined outside of this subprofile. The process of assigning capacity to a Pool and allocating capacity to a Volume or Logical Disk is defined inside of this subprofile. To make the diagram simpler, the Pool has only one component extent box that represents many Extents. The “SUM_” prefix states that the size of the Extents as a summation. Both the Pool and extent start with 1000 blocks of storage capacity.

Figure 146: Extent Conservation, Step 1



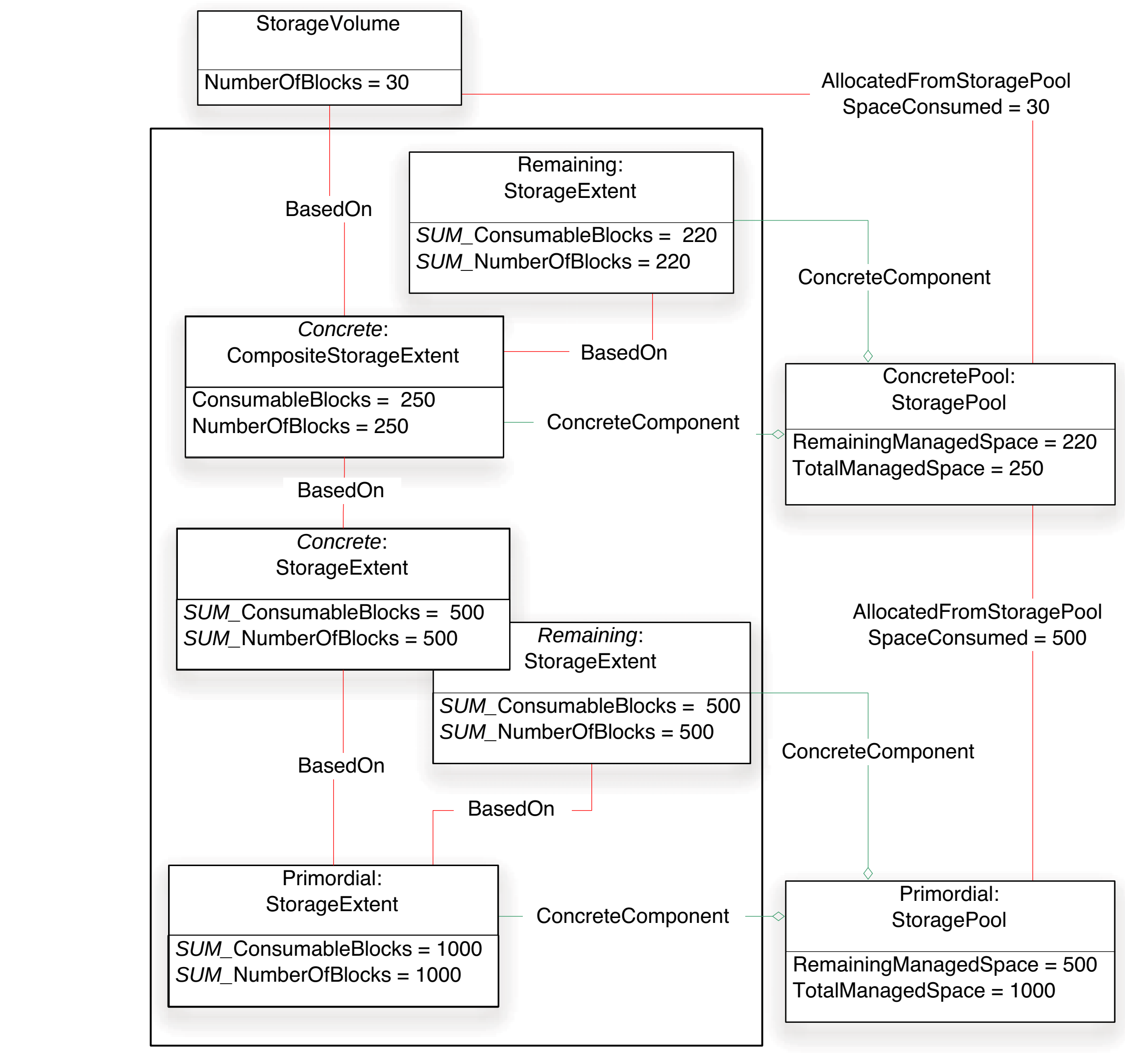
A concrete Pool is drawn from the primordial Pool in Figure 147: "Extent Conservation, Step 2". The next three figures group the instances modeled using the Extent Composition Subprofile with a dark box. The concrete Pool takes only half the capacity of the parent Pool. In this particular example, the metadata required by the implementation is written to the storage after this step. Another extent is create to represent the remaining capacity of the primordial Pool that was not used in the creation of the concrete Pool. ConsumableBlocks remain constant after the creation of the extent as a representation of the space actually available for use is other storage elements that are based on the extent. The remaining space extent can used to used for the creation of other Volumes or Logical Devices. If GetAvailableExtents were called on the primordial Pool at this point, only a reference to the remaining extent is be returned and not a reference to the primordial extent because the primordial extent is entirely used.

Figure 147: Extent Conservation, Step 2



A Volume is created in the Figure 148: "Extent Conservation, Step 3". This particular implementation draws storage capacity for metadata (for itsown house keeping) during the creation of the Volume. Not shown is the case where the metadata is drawn from capacity during the creation of the concrete Pool. A RAID 1 stripe is written over three Extents. These Extents are likely to be something like disk drives. Again, a remaining extent is created to represent the capacity of the parent concrete Pool that is not used in the creation of the Volume. Like before, a call to the concrete Pool's GetAvailableExtents method would yield a reference to the remaining extent.

Figure 148: Extent Conservation, Step 3



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

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

Formulas For Calculating Capacity

The following formulas define what calculations shall be valid in a conferment implementation.

- RemainingManagedSpace plus AllocatedFromStoragePool.SpaceConsumed from all of the Storage Volumes, Logical Disks, and Storage Pools allocated from the Pool shall equal TotalManagedSpace.
- A parent Pool’s TotalManagedSpace equals RemainingManagedSpace plus the sum of all related AllocatedFromStoragePool SpaceConsumed.
- If the extent Composition Subprofile is implemented
  1) The Pool’s TotalManagedSpace is equal to the sum of all the ConcreteComponent StorageExtent’s BlockSize times ConsumableBlocks. The StorageExtents shall be concrete or primordial, but not remaining StorageExtents.
  2) Using the BasedOn association from the Pool’s component Extents (found using ConcreteComponent), the sum of the dependent extent’s NumberOfBlocks is equal to the ConsumableBlocks of the antecedent extent.

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 Pool to advertise its capabilities (using implementation independent attributes) and a client to set the attributes it desires.

The concept of ‘hints’ is also included that allows 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 them in determining a matching configuration or it may choose to ignore the hints.

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

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 because, for some reason, a copy of the data is no longer available.

StorageVolumes or LogicalDisks are created from StoragePools via a StorageConfigurationService’s CreateOrModifyElementFromStoragePool( ) method. A volume create operation may take some period of time, however, and a Client needs to be aware that the operation is not complete until the StorageVoume.OperationalStatus is OK. A Client may also follow the progress of the operation using the ConcreteJob class and its properties.

8.2.8.10.2 Health and Fault Management Considerations

The extrinsic methods should produce Errors instead of some of the failure return codes. CIM Errors can include parameter errors, hardware efforts, time-out errors, and so on. See 8.2.1.6, "Health Package" for details.

EXPERIMENTAL

The standard messages specific to this profile are listed in Table 1005. See 6.5, "Standard Messages" for a list and description of all standard messages.

Table 1005: 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

8.2.8.10.3 Cascading Considerations

Not defined in this standard.

8.2.8.10.4 Supported Subprofiles and Packages.

Not defined in this standard.

8.2.8.10.5 Methods of this Profile

8.2.8.10.5.1 Extrinsic Methods on StorageCapabilities

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 defines what type 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 can not 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 Logical Disk 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 and 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 LogicaDisk, 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 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 in done as part of the creation of the concrete Pool 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.

StorageSettingHints may be used to by a client to imply what striping characteristics are desired, amongst other characteristics. The striping and parity methods and properties may be used in combination with hints. However, 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 Pool or storage element using its best effort.

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

Getting Stripe Length

```
uint32 GetSupportedStripeLengths(
                             [Out] unint16 StripeLengths[])
```

This method is used to report discrete ExtentStripeLengths for Volume, LogicalDisk, or Pool creation. Some systems may only support 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 Volume, Logical, or Pool creation. Some systems may only support a range of sizes. This method reports the minimum and maximum sizes in bytes and a divisor that defines what the candidate size shall be a multiple of.

Either method may be supported. Return codes:

- 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 not supported for this Capability. Usually, this return code indicates that the stripe length has already been set in the parent Pool and may not be changed.
- 3, “Use [GetSupportedStripeLengths|GetSupportStripeLengthRange] instead”. This return code tells the client that the stripe method is not supported, but the other one is supported.

Getting Stripe Depth

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

This method is used to report discrete UserDataStripeDepths for Volume, LogicalDisk, and Pool creation. Some systems may only support 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 Volume, Logical, or Pool creation. Some system may only support a range of sizes. This method reports the minimum and maximum sizes in bytes and a divisor that defines what the candidate size shall be a multiple of.

Either method may be supported. Return codes:

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

Getting Parity

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

This method is used to return the type of parity, non-rotated or rotated, that the capabilities 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 not supported for this Capability. Usually, this return code indicates that the parity has already been set in the parent Pool and may not be changed.

8.2.8.10.5.2 Intrinsic Methods on StorageSetting

In addition to this extrinsic, the following Intrinsic write methods are supported on StorageSetting:

- DeleteInstance;
- ModifyInstance

8.2.8.10.5.3 Extrinsic Methods on StorageConfiguration

Element Naming

Several of the following methods allow for a client to specify a name for the storage element that is being create or to change the name of an 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.

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 Pool from either a source pool or a list of storage extents. Any required associations (such as HostedStoragePool) are created in addition to the instance of Storage Pool. 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 Pool 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 pool. If it is not possible to create a pool 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 *CIM Operations* for format) to source Storage Pools. At least one of the Pool references shall be a primordial Storage Pool.

- InExtents[]: This is an array of strings containing Object references (see 4.11.5 of *CIM Operations* for format) to source Storage Extents. Note that an array of source Pools or an array of source Extents or both can be defined. See “Extent Conservation” on page 945 in 8.2.8.10.1.
- 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 storage element that is created can be determined by using the Job model.

<u>The CreateOrModifyStoragePool method and the Primordial Pool</u>

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

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

A client should pass only concrete Pools when creating a Pool from several Pools.

<u>DeleteStoragePool</u>

```
Uint32 DeleteStoragePool(
                 [Out] CIM_ConcreteJob ref Job,
                 [in] CIM_StoragePool ref Pool);
```

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

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

<u>CreateOrModifyElementFromStoragePool</u>

```
 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 Storage Pool. The parameters are as follows:

- 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 8.2.1.7, "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 Storage Pool. Goal may be a null value, in which case the default Setting for the Pool is used.
- Size: As an input this is the desired size of the element. If it is not possible to create a volume 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 Storage Pool.
- TheElement:
  - As Input: If the TheElement parameter is not null, then this method shall attempt to modify the reference element. Otherwise, then this method shall attempted 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, then storage element that is created can be determined by using the Job model.

<u>CreateOrModifyElementFromElements</u>

```
unint32 CreateOrModifyElementFromElements(
[In,
Values {“StoragePool”, “StorageVolume”, “StorageExtent”, “LogicalDisk”},
 ValueMap{”2”,”3”, “4”, “5”}]
unit16 ElementType,
[In, Out] CIM_ConcreteJob REF Job,
[In] CIM_ManagedElement REF Goal,
[In, Out] unit64 Size,
[In] CIM_StorageExtent REF InElements[],
[In, Out] CIM_LogicalElement REF TheElement);
```

The parameters are as follows:

- 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. 8.2.1.7, "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 Storage Pool. Goal may be a null value, in which case the default Setting for the Pool is used.
- Size: As an input this is the desired size of the element. If it is not possible to create a volume 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 Extents to be used for the storage element creation or modification.
- TheElement:
  - As Input: If the TheElement parameter is not null, then this method shall attempt to modify the reference element. Otherwise, then this method shall attempted 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, then storage element that is created can be determined by using the Job model.

ReturnToStoragePool

```
   Uint32 ReturnToStoragePool (
                         [Out] CIM_ConcreteJob ref Job,
                         [In] CIM_LogicalElement ref Element);
```

This method is provided to allow a client to delete a previously created element such as a Storage Volume.

Return Values

Each method has this set of defined return codes:

```
   ValueMap {“0”, “1”, “2”, “3”, “4”, “5”, “..”, “0x1000”,”0x1001”},
   Values {“Job completed with no error”, “Not Supported”, “Unknown”, “Timeout”,
       “Failed”,
       “Invalid Parameter”, “DMTF Reserved”, “Method parameters checked - job
       started”,
       “Size not supported”}]
```

- If the method completes immediately with no errors (and with no asynchronous execution required), “Job completed with no error” is returned.
- If the method parameters have been checked and the method is being executed asynchronously, “Method parameters checked - job started” is returned.
- If, for a Create/Modify method, the requested size is not supported then “Size not supported” is returned and the Size parameter is set to the nearest supported size.
- If one of the method parameters is incorrect (for instance invalid object paths), then “Invalid Parameter” is returned.
- “Timeout” or “Failed” may be returned if the provider has problems accessing the hardware or for other implementation specific reasons.
- A vendor shall not extend the Value map to express vendor specific error situations not catered for by the standard messages.

8.2.8.10.5.4 Extrinsic Methods on StoragePool

All the following methods return sizes in units of bytes.

GetSupportedSizes

```
  unit32 GetSupportedSizes(
                      [In] uint16 ElementType,
                      [In] CIM_StorageSetting ref Goal,
                      [Out] uint64 Sizes[ ]);
```

- ElementType: This enumeration specifies what type of object to create.
- 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 Storage Pool. Goal may be a null value, in which case the default Setting for the Pool is used.
- Sizes: An array containing all the possible sizes an element of type can take on either as a creation or modification operation. If a possible value would be 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, ex. StoragePool, StorageVolume or LogicalDisk, that can be created or modified using capacity from the StoragePool. The method is used for storage system 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 sizes reported may not differ from each other by a fixed size.

GetSupportedSizeRange

```
unint32 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: 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 Storage Pool. Goal may be a null value, in which case the default Setting for the Pool is used.
- MinimumVolumeSize: The minimum size an element can take on either as a creation of modification operation.
- MaximumVolumeSize: The maximum size an element can take on either as a creation of modification operation
- VolumeSizeDivisor: The value used to determine what the sizes between the MinimumVolumeSize and the MaximumVolumeSize numbers.

This method is used to determine the possible sizes of child element, ex. 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 can prove 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 varies depending on the state of the StoragePool. For example, when a StoragePool is first created that allows for possible sizes to be in 1024 byte blocks, then the GetSupportedSizeRange method would be better to report the 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 is better in reporting the non-continuous regions of capacity that may be used for element creation. Another example, there are some storage system that can only allocate the StorageVolume or Logical Disk in whole disks and these disks need not be of a uniform size. In this case, the storage system would only support the GetSupportedSizes method.

However, both methods may be supported at the same time and report different values when discontiguous and contiguous capacity is present in the Pool. In this case, the GetSupportSizes method would be used to report the fragments of available capacity. The remaining contiguous capacity is reported as the largest element size possible. The GetSupportSizeRange would be used to report what elements sizes may be drawn from the contiguous capacity.

If there is no notion of continuity as being a stable state of the system, as in capacity is continuously and automatically being defragmented, then only support of the GetSupportSizeRange method is warranted.

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 did not complete successfully, then either the method is not supported or it is suggested to use the other method instead. The GetSupportSizes method can notify the SMI-S client that it should use the GetSupportSizeRanges instead or the GetSupportedSizeRange method can notify the SMI-S client that it should use the GetSupportedSizes method instead.

GetAvailableExtents

```
unit32 GetAvailableExtents(
                  [In] CIM_StorageSetting REF Goal,
                  [Out] CIM_StorageExtent REF AvailableExtents[ ]);
```

This method is used to retrieve that component Extents of the Pool that do not form that basis for Volumes and LogicalDisks allocated from the Pool - the available Extents - and can support the passed Goal parameter. If a NULL is passed for a Goal, then all the available, component Extents of the Pool are returned.

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

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”}]
```

If the method completes immediately with no errors (and with no asynchronous execution required), “Job completed with no error” is returned.

If the implementation does not support the method, then a “Not Supported” return code is returned.

If one of the method parameters is incorrect (for instance invalid object paths), then “Invalid Parameter” is returned.

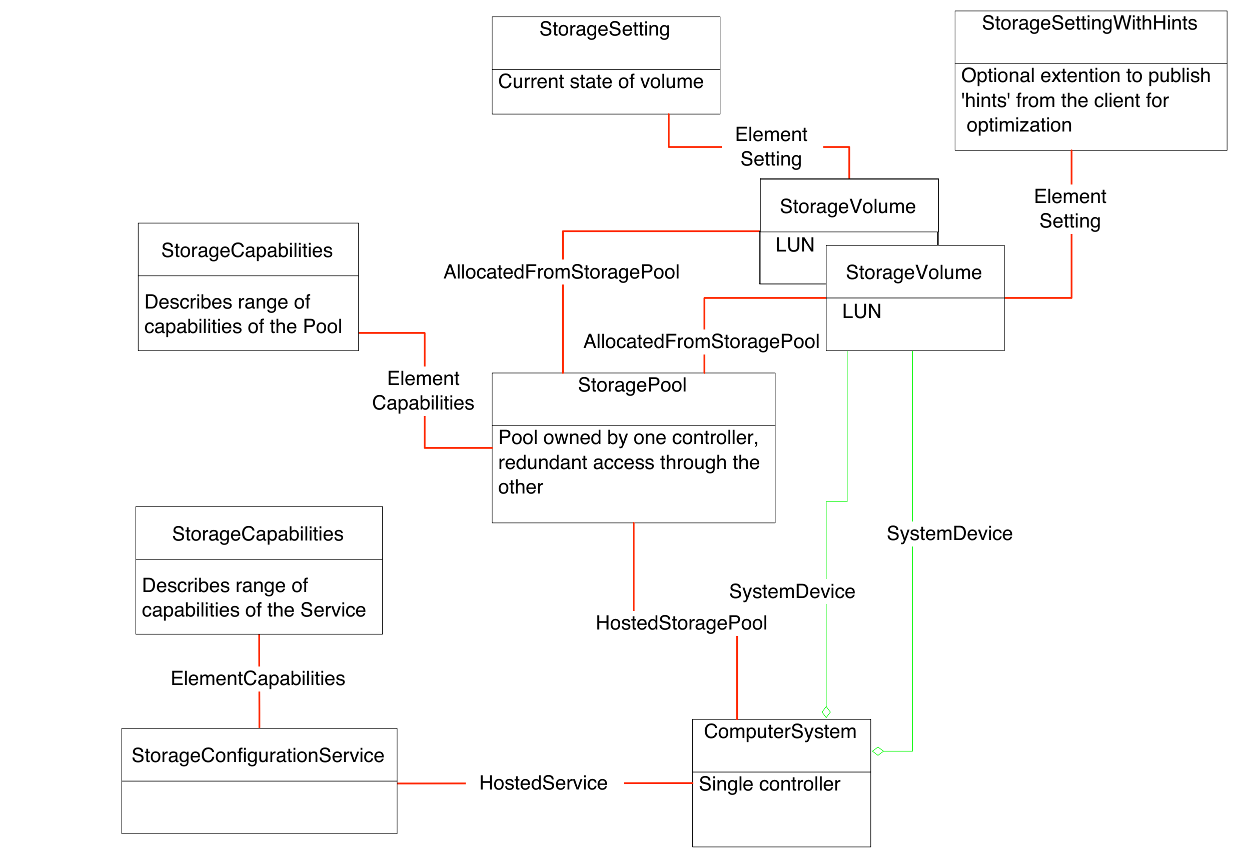
“Timeout” or “Failed” may be returned if the provider has problems accessing the hardware or for other implementation specific reasons.

8.2.8.10.6 Client Considerations and Recipes

Representative Instance Diagram

Figure 149: "Representative Block Service Instance Diagram" shows the classes and associations needed to model a single Pool with two StorageVolumes

Figure 149: Representative Block Service Instance Diagram



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 can not 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 different between the initial Setting state and the current state for package redundancy, for example, that is in the StorageVolume or LogicalDisk.

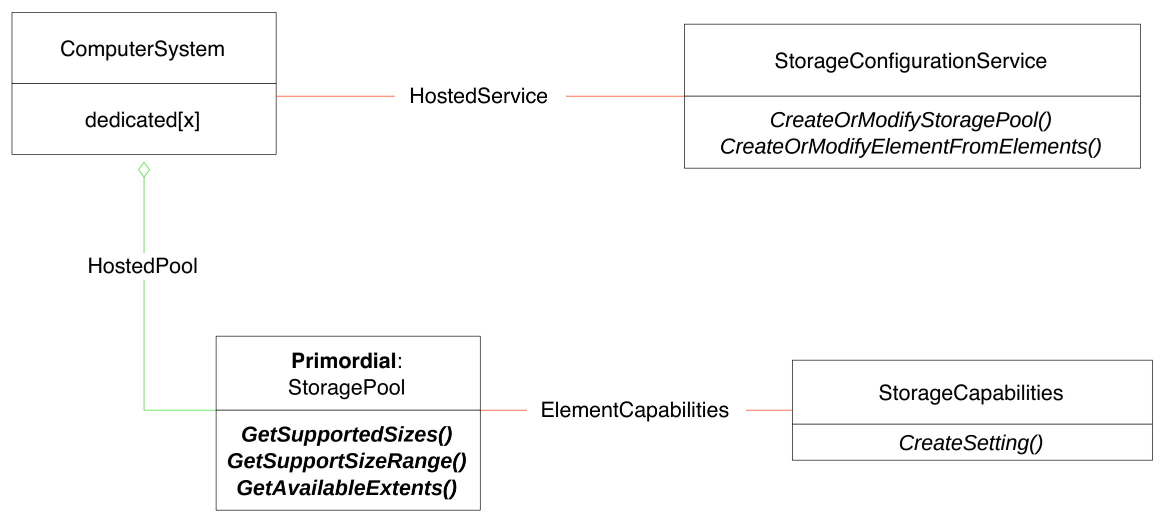
If a client specifies a goal asking for no single point of failure, the implementation shall return an error if the system is not capable of supporting that goal. However, if a client specifies that single points of failure are allowed, the implementation may return storage that has potential single 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 means that additional copies of the data will be made. If allowed, the client request of additional package redundancy means that additional drives, for example, will be assigned to this storage element. The redundant package may be overassigned, that is assign as extra packages for more than one storage element, or they may be dedicated. See 8.2.8.15, "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.

Representative Storage Pool Creation Example.

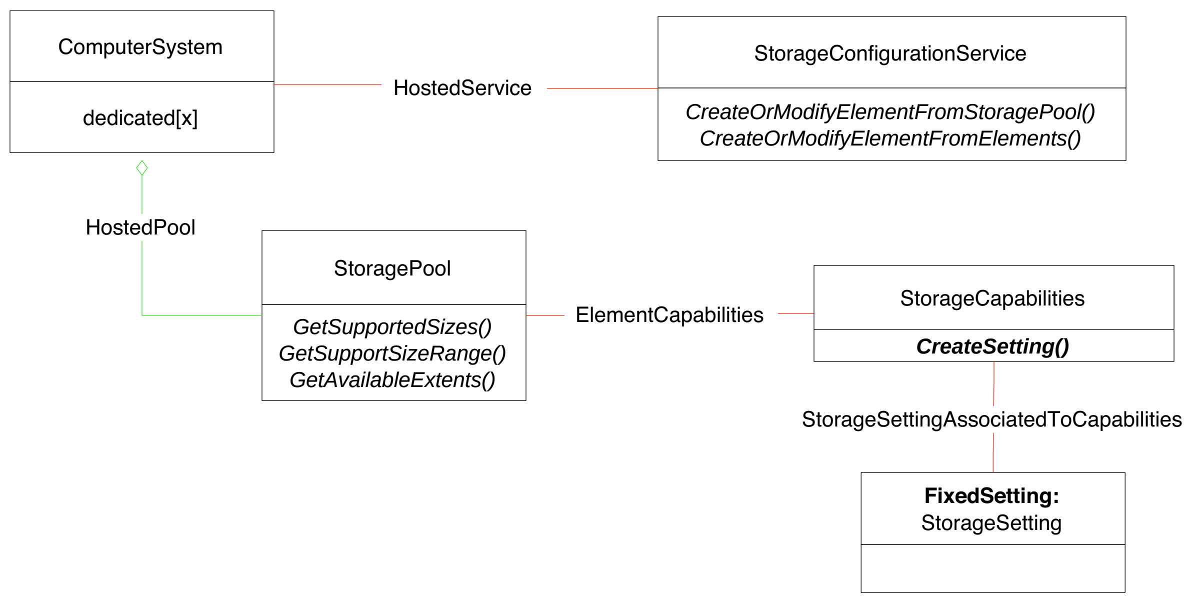
Figure 150: "Pool Creation, Initial State" shows the initial state of the block storage system - a single ‘primordial pool that advertises its capabilities. One can make use of the GetSupportedSizes() and GetSupportedSizeRange() methods to determine what sizes of pools can be created from the Primordial Pool, given a goal StorageSetting. Alternatively, if the Pool is to be created from Extents, then one can make use of the GetAvailableExtents() to obtain a list of available component Extents of the Pool that also match the Goal.

Figure 150: Pool Creation, Initial State



Next, (Figure 151: "Pool Creation - Step 2") 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.ChageableType = “2” (“Changeable - Persistent”); these Settings have been generated but were modified to persist.

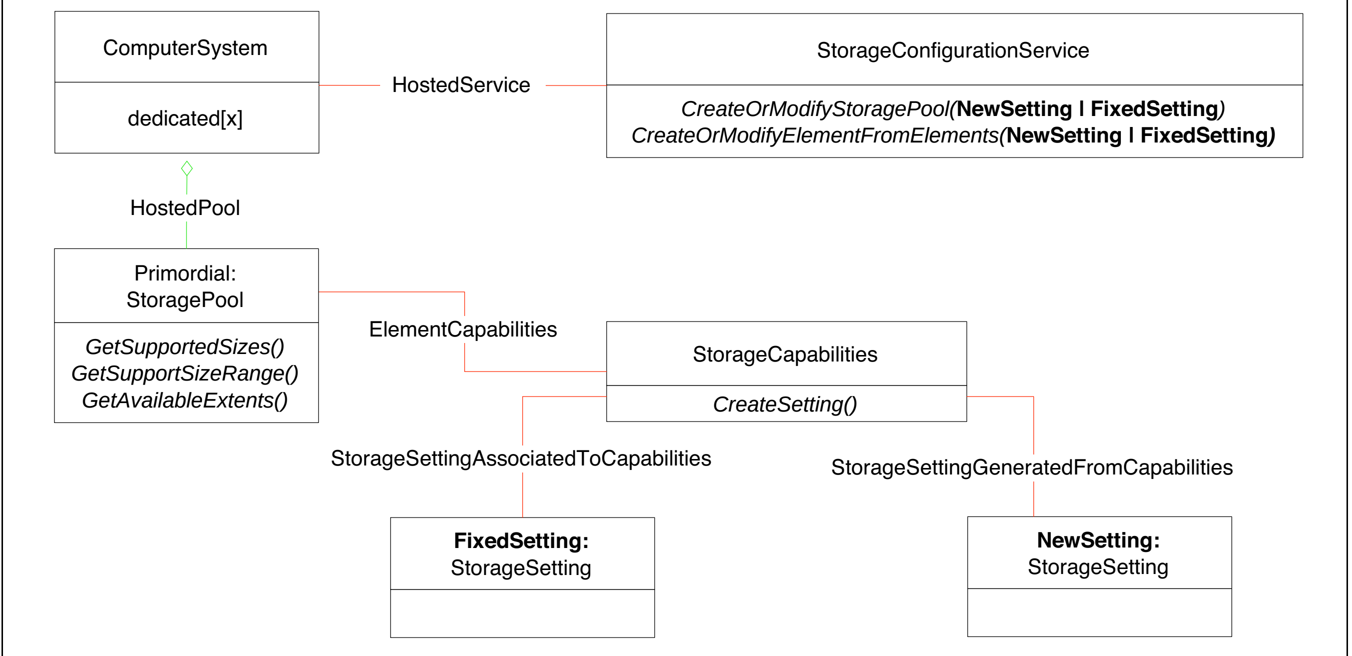
Figure 151: Pool Creation - Step 2



Once this generated Setting as been altered as required or alternatively a pre-defined Setting is used, the Goal Setting is passed as an argument to the CreateOrModifyStoragePool method in the StorageConfigurationService. (Shown in Figure 152: "Pool Creation - Step 3"). Alternatively, the client can create the Pool by passing the Goal, the desired component Extents, 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, component, Extents. The list of available Extents is best retrieved using the GetAvailableExtents() method. If the Size is less than the desired Extents by a size less than smallest extent passed, then one of the Extents is partitioned into the used and free parts. See “Extent Conservation” on page 945 in 8.2.8.10.1.)

.

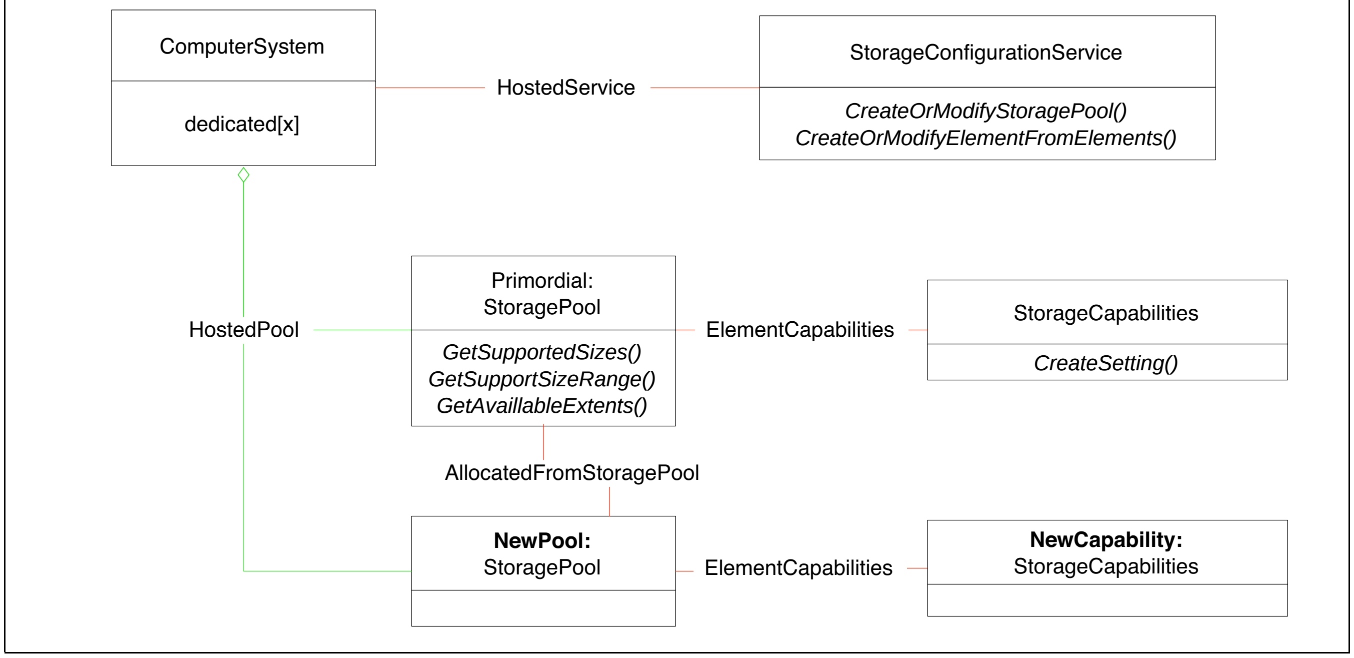
Figure 152: Pool Creation - Step 3



ComputerSystem
dedicated[x]
HostedService
StorageConfigurationService
CreateOrModifyStoragePool(NewSetting I FixedSetting)
CreateOrModifyElementFromElements(NewSetting I FixedSetting)
HostedPool
Primordial: StoragePool
GetSupportedSizes()
GetSupportSizeRange()
GetAvailableExtents()
ElementCapabilities
StorageCapabilities
CreateSetting()
StorageSettingAssociatedToCapabilities
StorageSettingGeneratedFromCapabilities
FixedSetting: StorageSetting
NewSetting: StorageSetting

The Pool is then created. If the generated Setting was used as the Goal, then this ‘temporary’ StorageSetting is replaced with an equivalent object linked to the new pool with ElementCapabilities. (Shown in Figure 153: "Pool Creation - Step 4").

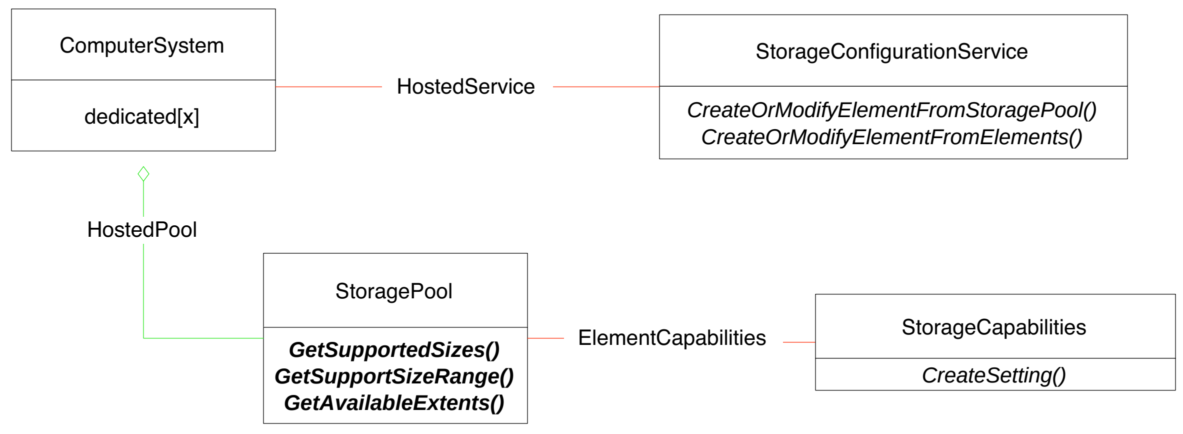
Figure 153: Pool Creation - Step 4



Representative example of Storage Volume or Logical Disk Creation

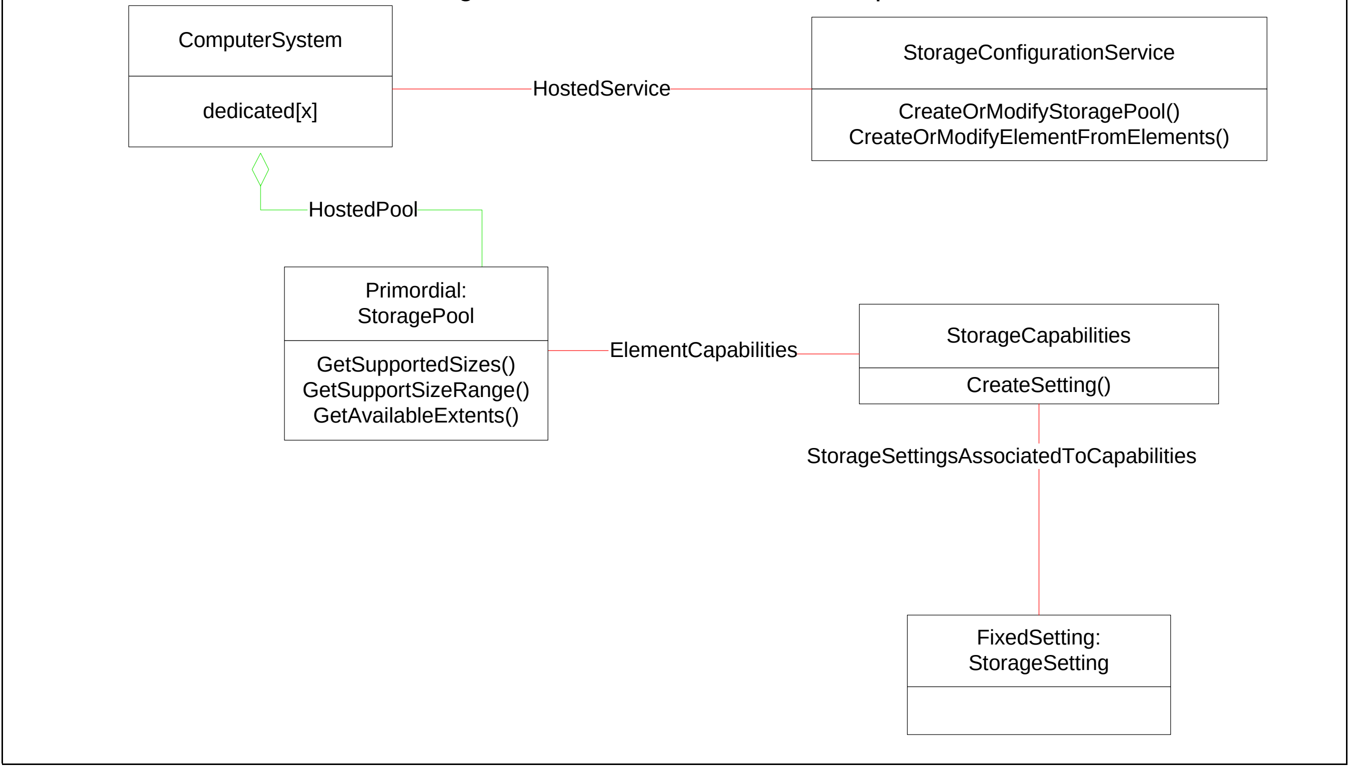
Similarly to with Storage Pools, a client chooses a suitable source Pool by referencing the StorageCapabilities objects and using the GetSupportedSizes() and GetSupportSizeRange() methods, given a goal Setting. Alternatively, a client can retrieve the available, component Extents of the Pool, given a goal StorageSetting, with the GetAvailableExtents() methods. The client may either create a Volume or LogicalDisk by specifying a size, source Extents, or a combination. This is indicated in Figure 154: "Volume Creation - Initial State"

Figure 154: Volume Creation - Initial State



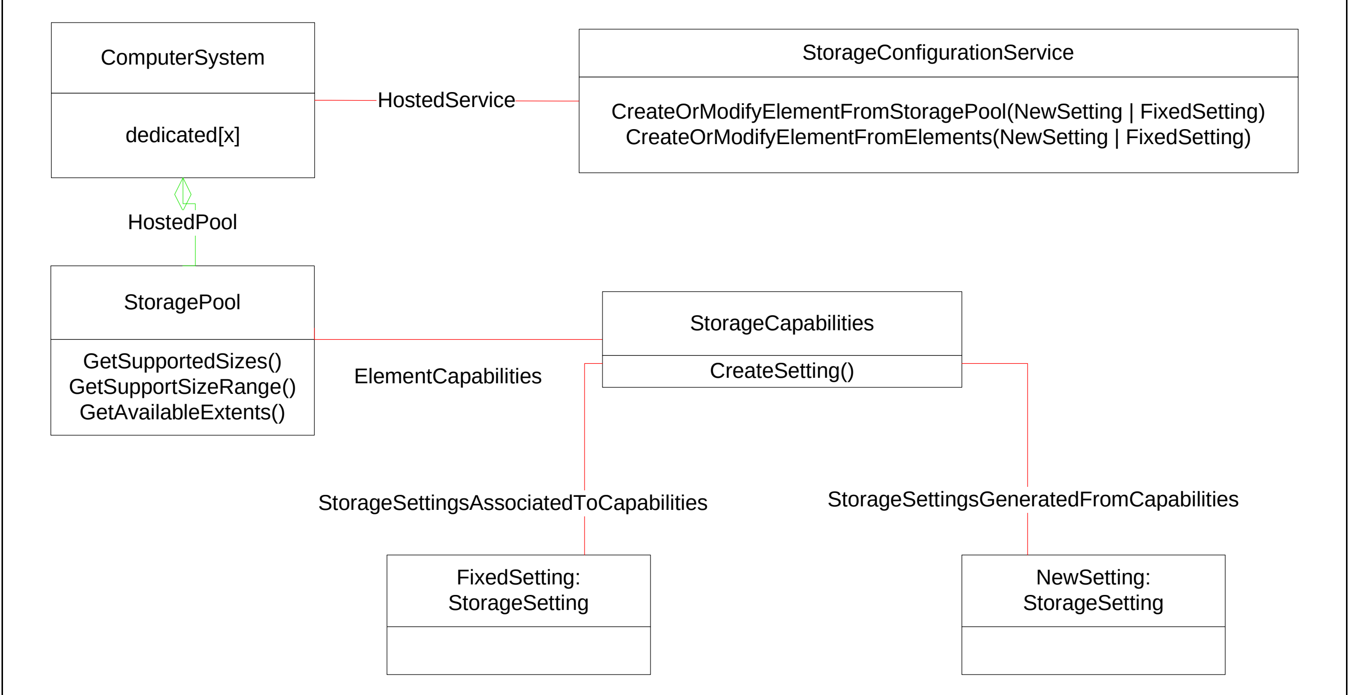
Once a suitable pool is found, a StorageSetting instance can be created using the CreateSetting method on the StorageCapabilities object (see Figure 154: "Volume Creation - Initial State"). If a suitable StorageSetting already exists it could be used instead. Pre-existing Settings can be located by using 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. This is illustrated in Figure 155: "Volume Creation - Step 1". Another Setting already associated to a storage element can be used as a goal, but it shall not be modifiable.

Figure 155: Volume Creation - Step 1



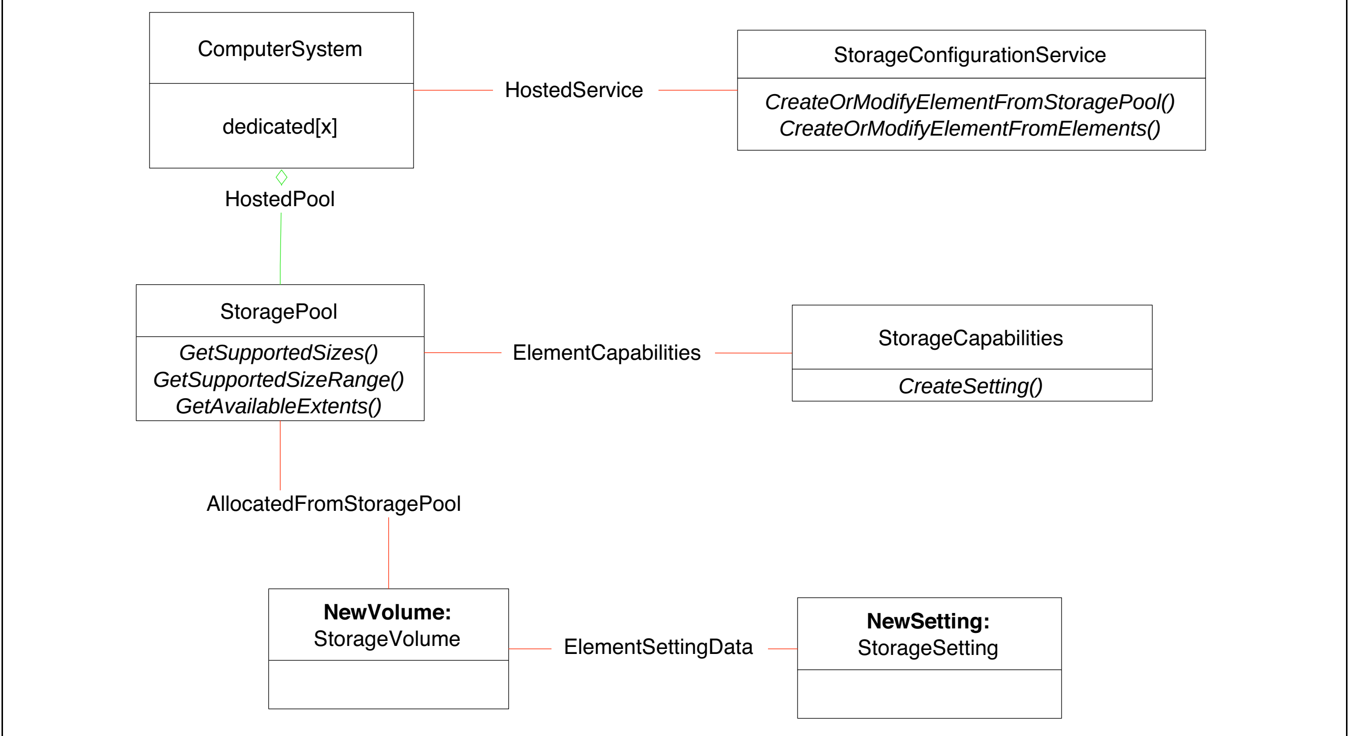
If a new Setting is created, it is linked back to the originating StorageCapabilities object until it is used as an argument in a StorageConfiguration method. (see Figure 156: "Volume Creation - Step 2"). Alternatively, the client can create the Volume or LogicalDisk, for example, by passing the Goal, the desired component Extents, 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, component, Extents. The list of available Extents is best retrieved using the GetAvailableExtents() method. If the Size is less than the desired Extents by a size less than smallest extent passed, then one of the Extents is partitioned into the used and free parts. See “Extent Conservation” on page 945 in 8.2.8.10.1.

Figure 156: Volume Creation - Step 2


ComputerSystem
dedicated[x]
HostedService
StorageConfigurationService
CreateOrModifyElementFromStoragePool(NewSetting | FixedSetting)
CreateOrModifyElementFromElements(NewSetting | FixedSetting)
HostedPool
StoragePool
GetSupportedSizes()
GetSupportSizeRange()
GetAvailableExtents()
StorageCapabilities
CreateSetting()
ElementCapabilities
StorageSettingsAssociatedToCapabilities
StorageSettingsGeneratedFromCapabilities
FixedSetting: StorageSetting
NewSetting: StorageSetting

Once the Volume 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 can not assume that the instances are the *same* instance. (see Figure 157: "Volume Creation - Step 3")

Figure 157: Volume Creation - Step 3



8.2.8.10.6.1 Summarize the Pools in a block storage system and verify the capacity reported.

```
// DESCRIPTION
//  This recipe retrieves and validates the total, remaining and consumed
// storagepool 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>
    }
}
```

8.2.8.10.6.2 Create Storage Pool and Storage Element on Block Server (e.g., Array or Volume Manger)

```
// 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[])
{
     <Sort the $CapabilitiesOffered[] so that the capability with the
      greatest DataRedundanctMax, 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].SupportedAsyncronousActions[])
#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].Primodial == #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)
        {
            <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 = “SELECT * FROM CIM_InstModification
               WHERE SourceInstance ISA CIM_ConcreteJob
                 AND ANY SourceInstance.OperationalStatus[*] = 17
                AND ANY SourceInstance.OperationalStatus[*] = 2 “
    @{Determine if Indications already exist or have to be
created}&createIndication(#Filter1)

    // ‘17’ (“Completed”) ‘6’ (“Error”)
    #Filter2 = “SELECT * FROM CIM_InstModification
               WHERE SourceInstance ISA CIM_ConcreteJob
                 AND ANY SourceInstance.OperationalStatus[*] = 17
                AND ANY SourceInstance.OperationalStatus[*] = 6 “
    @{Determine if Indications already exist or have to be
created}&createIndication(#Filter2)
```

```
}

// Step 6. Create the Storage Pool
if(#SupportsPoolCreation)
{
     %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,
     %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()
}
```

8.2.8.10.6.3 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.>
    }
}

// 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”,
     “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 = “SELECT * FROM CIM_InstModification
               WHERE SourceInstance ISA CIM_ConcreteJob
                 AND ANY SourceInstance.OperationalStatus[*] = 17
                AND ANY SourceInstance.OperationalStatus[*] = 2 “
     @{Determine if Indications already exist or have to be
created}&createIndication(#Filter1)
```

```
     // ‘17’ (“Completed”) ‘6’ (“Error”)
     #Filter2 = “SELECT * FROM CIM_InstModification
               WHERE SourceInstance ISA CIM_ConcreteJob
                 AND ANY SourceInstance.OperationalStatus[*] = 17
                AND ANY SourceInstance.OperationalStatus[*] = 6 “
     @{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 coersed from %OutArguments[“Job”]
        “CIM_AffectedJobElement”,
        #ElementClassName,
        null,
        null,
        false,
        false,
        null)
     // Only one Storage Element will be created,
    $CreatedElement-> = $CreatedElement[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 >
     }
}
```

8.2.8.10.6.4 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
//   createdlike CIM_StorageVolume or CIM_LogicalDisk.
// 6.   #ElementType is set to the element to created
//        2 - StorageVolume
//        4 - LogicalDisk
//   See CreateOrModifyElementFromStoragePool.ElementType

// 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].SupportedAsyncronousActions[])
#SupportsElementCreation1 = contains(12, // Storage Element from Element Creation
     $ServiceCapabilities[0].SupportedSynchronousActions[])
#SupportsElementCreation2 = contains(3, // LogicalDiskCreation
     $ServiceCapabilities[0].SupportedStorageElementFeatures[])
#ElementCreationProducesJob = contains(12, // 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
//   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].Primodial == #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 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 = “SELECT * FROM CIM_InstModification
            WHERE SourceInstance ISA CIM_ConcreteJob
            AND ANY SourceInstance.OperationalStatus[*] = 17
            AND ANY SourceInstance.OperationalStatus[*] = 2 “
     @{Determine if Indications already exist or have to be
created}&createIndication(#Filter1)

     // ‘17’ (“Completed”) ‘6’ (“Error”)
     #Filter2 = “SELECT * FROM CIM_InstModification
            WHERE SourceInstance ISA CIM_ConcreteJob
            AND ANY SourceInstance.OperationalStatus[*] = 17
            AND ANY SourceInstance.OperationalStatus[*] = 6 “
     @{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,
        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-> = $CreatedElements[0].getObjectPath()
}
```

8.2.8.10.6.5 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
//      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 >
    }
```

8.2.8.10.7 Registered Name and Version

Block Services version 1.1.0

8.2.8.10.8 CIM Server Requirements

Table 1006: CIM Server Requirements for Block Services

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.10.9 CIM Elements

Table 1007: CIM Elements for Block Services

Element Name	Description
Mandatory Classes	
CIM_AllocatedFromStoragePool (8.2.8.10.9.1)	AllocationFromStoragePool as defined in the Array Profile
CIM_ElementCapabilities (8.2.8.10.9.2)	Associates StorageConfigurationCapabilities withStorageConfigurationService.
CIM_ElementSettingData (8.2.8.10.9.3)	
CIM_LogicalDisk (8.2.8.10.9.5)	A LogicalDisk is allocated from a concrete StoragePool.
CIM_StorageCapabilities (8.2.8.10.9.6)	
CIM_StoragePool (8.2.8.10.9.9)	Common elements to Primordial and Concrete Pools.
CIM_StoragePool (8.2.8.10.9.10)	The Primordial Storage Pool. 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 (8.2.8.10.9.11)	The Concrete Storage Pools. 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_StorageSetting (8.2.8.10.9.12)	
CIM_StorageSettingsGeneratedFromCapabilities (8.2.8.10.9.15)	This class associates the StorageCapabilies 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_StorageVolume (8.2.8.10.9.16)	A SCSI logical unit representing a virtual disk. A StorageVolume is allocated from a concrete StoragePool.
Optional Classes	
CIM_HostedService (8.2.8.10.9.4)	
CIM_StorageConfigurationCapabilities (8.2.8.10.9.7)	
CIM_StorageConfigurationService (8.2.8.10.9.8)	
CIM_StorageSettingWithHints (8.2.8.10.9.13)	

Table 1007: CIM Elements for Block Services

Element Name	Description
CIM_StorageSettingsAssociatedToCapabilities (8.2.8.10.9.14)	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").
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool	Creation/Deletion of StoragePool
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StoragePool	Deletion of StoragePool
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageVolume	Creation of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageVolume	Deletion of StorageVolume, if the StorageVolume storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageVolume AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	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	CQL - Change of status of a Storage Volume, if Storage Volume is implemented.
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_LogicalDisk	Creation of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_LogicalDisk	Deletion of LogicalDisk, if the LogicalDisk storage element is implemented.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_LogicalDisk AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	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	CQL - Change of status of LogicalDisk, if LogicalDisk is implemented.

8.2.8.10.9.1 CIM_AllocatedFromStoragePool

AllocationFromStoragePool as defined in the Array Profile
Class Mandatory: true

Table 1008: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StoragePool	
Dependent		CIM_LogicalElement	
SpaceConsumed		uint64	

8.2.8.10.9.2 CIM_ElementCapabilities

Associates StorageConfigurationCapabilities withStorageConfigurationService.
Class Mandatory: true

Table 1009: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element.
Capabilities		CIM_Capabilities	The Capabilities object associated with the element.

8.2.8.10.9.3 CIM_ElementSettingData

Class Mandatory: true

Table 1010: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The ManagedElement.
SettingData		CIM_SettingData	The Setting Data object associated with the ManagedElement.
IsDefault		uint16	An enumerated integer indicating that the referenced setting is a default setting for the element, or that this information is unknown."),||ValueMap {"0", "1", "2"}, ||Values {"Unknown", "Is Default", "Is Not Default"}
IsCurrent		uint16	An enumerated integer indicating that the referenced setting is currently being used in the operation of the element, or that this information is unknown."),||ValueMap {"0", "1", "2"}, ||Values {"Unknown", "Is Current", "Is Not Current"}

8.2.8.10.9.4 CIM_HostedService

Class Mandatory: false

Table 1011: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System.
Dependent		CIM_Service	The Service hosted on the System.

8.2.8.10.9.5 CIM_LogicalDisk

A LogicalDisk is allocated from a concrete StoragePool.
Deleted By : Extrinsic(s): StorageConfigurationService.ReturnToStoragePool
Class Mandatory: true

Table 1012: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
Name		string	OS Device Name
NameFormat		uint16	Format for name
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
BlockSize		uint64	
NumberOfBlocks		uint64	The number of blocks as reported by the volume manager.
ConsumableBlocks		uint64	The number of usable blocks.
IsBasedOnUnderlyingRedun-dancy		boolean	
NoSinglePointOfFailure		boolean	
DataRedundancy		uint16	
PackageRedundancy		uint16	
DeltaReservation		uint8	
Optional Properties/Methods			
ElementName		string	User-friendly name

8.2.8.10.9.6 CIM_StorageCapabilities

Created By : Static
Class Mandatory: true

Table 1013: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	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.
ElementType		uint16	Enumeration indicating the type of instance to which this StorageCapabilities applies. Only "6", StorageConfigurationService and "5" StoragePool are valid.
NoSinglePointOfFailure		boolean	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		boolean	Indicates the default value for the NoSinglePointOfFailure property.
DataRedundancyMin		uint16	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		uint16	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.

Table 1013: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Property	Flags	Type	Description & Notes
DataRedundancyDefault		uint16	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		uint16	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		uint16	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		uint16	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.
CreateSetting()			Generate a setting to use as a goal for creating or modifying storage elements.
Optional Properties/Methods			
ExtentStripeLengthDefault		uint16	ExtentStripeLengthDefault 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.

Table 1013: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Property	Flags	Type	Description & Notes
ParityLayoutDefault		uint16	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 assign-ment
UserDataStripeDepthDefault		uint64	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 orassign-ment.
GetSupportedStripeLengths()			List the possible discrete stripe lengths supported at this time of this method's execution.
GetSupportedStripe-LengthRange()			List the possible stripe length ranges supported at the time of this method's execution
GetSupportedParityLayouts()			List the possible parity layouts sup-ported at the time of this method's exe-cution.
GetSupportedStripeDepths()			List the possible stripe depths sup-ported at the time of this method's exe-cution.
GetSupportedStripeDepthRange()			List the possible strip depth ranges supported at the time of this method's execution.

8.2.8.10.9.7 CIM_StorageConfigurationCapabilities

Created By : Static
Class Mandatory: false

Table 1014: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
SupportedStoragePoolFeatures		uint16[]	Lists what StorageConfigurationService functionalities are implemented.
SupportedStorageElementTypes		uint16[]	Lists the type of storage elements that are supported by this implementation.

Table 1014: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Property	Flags	Type	Description & Notes
SupportedStorageElementFeatures		uint16[]	Lists actions supported through the invocation of StorageServiceService.CreateOrModifyElementFromStoragePool().
Optional Properties/Methods			
SupportedSynchronousActions		uint16[]	Lists what actions, invoked through StorageConfigurationService methods, shall not produce Concrete jobs.
SupportedAsynchronousActions		uint16[]	Lists actions, invoked through StorageConfigurationService methods, that may produce Concrete jobs.

8.2.8.10.9.8 CIM_StorageConfigurationService

Created By : Static
Class Mandatory: false

Table 1015: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

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

8.2.8.10.9.9 CIM_StoragePool

Common elements to Primordial and Concrete Pools.

Class Mandatory: true

Table 1016: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
PoolID		string	A unique name in the context of this system that identifies this Pool.
TotalManagedSpace		uint64	
RemainingManagedSpace		uint64	
Primordial		boolean	Default is FALSE, TRUE for Primordial Pools.
GetSupportedSizes()			List the discrete storage element sizes that can be created or expanded from this Pool.
GetSupportedSizeRange()			List the size ranges for storage element that can be created or expanded from this Pool.
Optional Properties/Methods			
ElementName		string	
GetAvailableExtents()			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.

8.2.8.10.9.10 CIM_StoragePool

The Primordial Storage Pool. 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.
Created By : Static
Class Mandatory: true

Table 1017: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Primordial		boolean	

8.2.8.10.9.11 CIM_StoragePool

The Concrete Storage Pools. A concrete StoragePool shall be allocated from another StoragePool. It shall be used for allocating StorageVolumes and LogicalDisks as well as other concrete StoragePools.
Created By : Extrinsic(s): StorageConfigurationService.CreateOrModifyStoragePool
Modified By : Extrinsic(s): StorageConfigurationService.CreateOrModifyStoragePool
Deleted By : Extrinsic(s): StorageConfigurationService.DeleteStoragePool

Class Mandatory: true

Table 1018: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Primordial		boolean	

8.2.8.10.9.12 CIM_StorageSetting

Created By : Extrinsic(s): StorageCapabilities.CreateSetting
Class Mandatory: true

Table 1019: SMI Referenced Properties/Methods for CIM_StorageSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	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		boolean	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		uint16	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		uint16	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		uint16	
PackageRedundancyMin		uint16	PackageRedundancyMin describes the minimum number of spindles or logicaldevices 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.

Table 1019: SMI Referenced Properties/Methods for CIM_StorageSetting

Property	Flags	Type	Description & Notes
PackageRedundancyMax		uint16	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		uint16	
ChangeableType		uint16	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.
Optional Properties/Methods			
ExtentStripeLength		uint16	ExtentStripeLength describes the desired stripe length goal.
ExtentStripeLengthMin		uint16	ExtentStripeLengthMin describes the minimum acceptable stripe length.
ExtentStripeLengthMax		uint16	ExtentStripeLengthMax describes the maximum acceptable stripe length.
ParityLayout		uint16	ParityLayout describes the desired parity layout.
UserDataStripeDepth		uint64	UserDataStripeDepth describes the desired stripe depth.
UserDataStripeDepthMin		uint64	UserDataStripeDepthMin describes the minimum acceptable stripe depth.
UserDataStripeDepthMax		uint64	UserDataStripeDepthMax describes the maximum acceptable stripe depth.

8.2.8.10.9.13 CIM_StorageSettingWithHints

Class Mandatory: false

Table 1020: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	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 1020: SMI Referenced Properties/Methods for CIM_StorageSettingWithHints

Property	Flags	Type	Description & Notes
NoSinglePointOfFailure		boolean	
DataRedundancyMin		uint16	
DataRedundancyMax		uint16	
PackageRedundancyMin		uint16	
PackageRedundancyMax		uint16	
DataAvailabilityHint		uint16	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		uint16	This hint is an indication from a client of the randomness of accesses. Values are 0=Entirely Sequential to 10=Entirely Random.
AccessDirectionHint		uint16	This hint is an indication from a client of the direction of accesses. Values are 0=Entirely Read to 10=Entirely Write
AccessSizeHint		uint16[]	This hint is an indication from a client of the optimal access sizes. Several sizes can be specified. Units("Megabytes")
AccessLatencyHint		uint16	This hint is an indication from a client how important access latency is. `Values are 0=Don't Care to 10=Very Important.
AccessBandwidthWeight		uint16	This hint is an indication from a client of bandwidth prioritization. Values are 0=Don't Care to 10=Very Important.
StorageCostHint		uint16	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		uint16	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		uint16	

8.2.8.10.9.14 CIM_StorageSettingsAssociatedToCapabilities

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").

Class Mandatory: false

Table 1021: SMI Referenced Properties/Methods for CIM_StorageSettingsAssociatedToCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageCapabilities	The StorageCapabilities reference.
Dependent		CIM_StorageSetting	The StorageSetting reference.
DefaultSetting		boolean	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.

8.2.8.10.9.15 CIM_StorageSettingsGeneratedFromCapabilities

This class associates the StorageCapabilies 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.
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: true

Table 1022: SMI Referenced Properties/Methods for CIM_StorageSettingsGeneratedFromCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageCapabilities	The StorageCapabilities reference.
Dependent		CIM_StorageSetting	The StorageSetting reference.

8.2.8.10.9.16 CIM_StorageVolume

A SCSI logical unit representing a virtual disk. A StorageVolume is allocated from a concrete StoragePool.
Deleted By : Extrinsic(s): StorageConfigurationService.ReturnToStoragePool
Standard Names: The Name, NameFormat,NameNamespace, OtherIdentifyingInfo, and IdentifyingDescriptions properties shall follow the requirements in 6.2.4.5.1

Class Mandatory: true

Table 1023: SMI Referenced Properties/Methods for CIM_StorageVolume

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
Name	CD	string	SCSI identifier for this volume
NameFormat		uint16	Format for Name property.
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
BlockSize		uint64	
NumberOfBlocks		uint64	The number of blocks as reported by the hardware.
ConsumableBlocks		uint64	The number of usable blocks.
IsBasedOnUnderlyingRedun-dancy		boolean	
NoSinglePointOfFailure		boolean	
DataRedundancy		uint16	
PackageRedundancy		uint16	
DeltaReservation		uint8	
Optional Properties/Methods			
ElementName		string	User-friendly name
OtherIdentifyingInfo	CD	string[]	Additional correlatable names
IdentifyingDescriptions		string[]	

8.2.8.10.10 Related Standards

Table 1024: Related Standards for Block Services

Specification	Revision	Organization
CIM Infrastructure	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.11 Block Server Performance Subprofile

8.2.8.11.1 Description

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. A mapping between fabric statistics and block server statistics is identified in the section 8.2.8.11.7, "Registered Name and Version".

- 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).

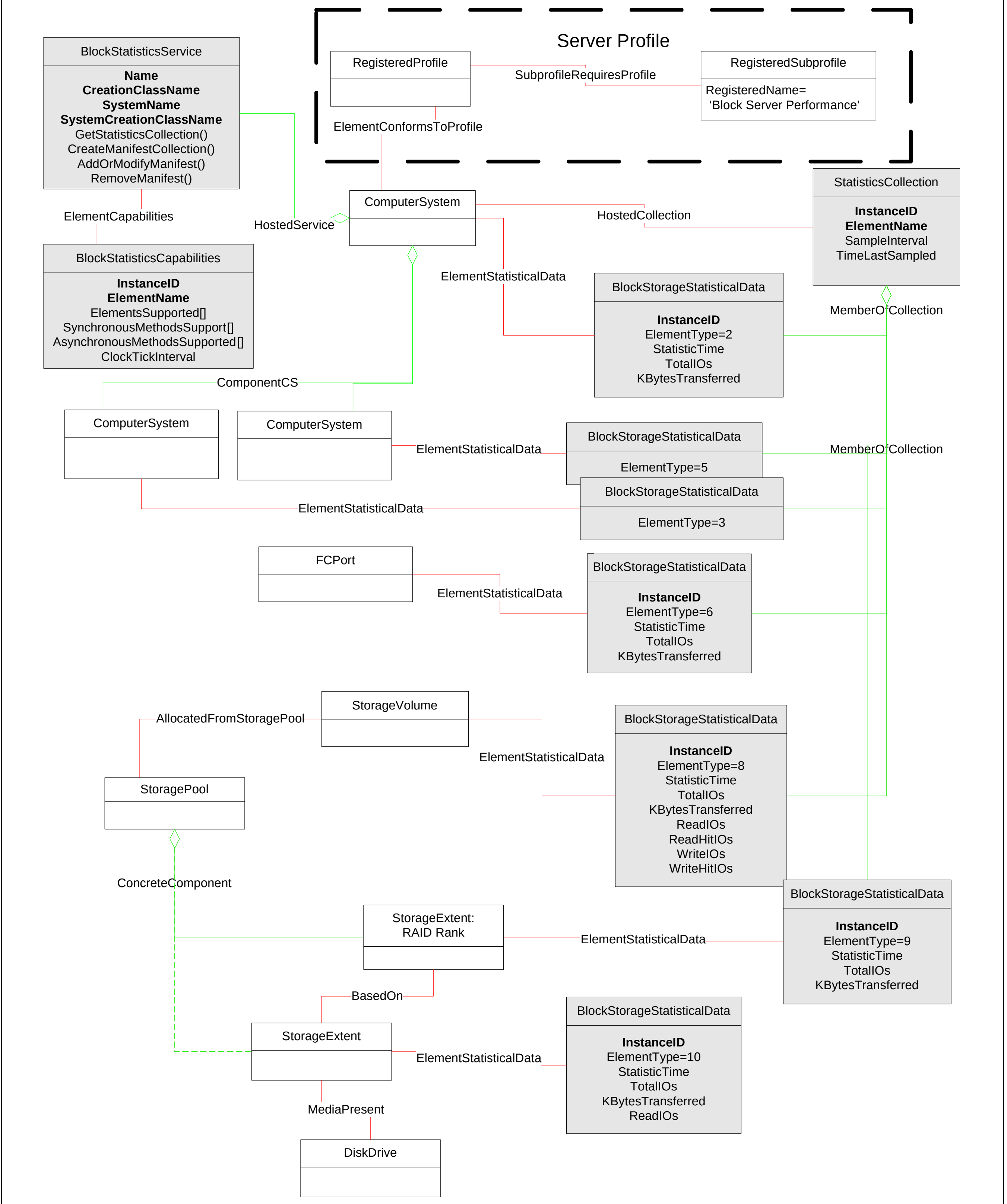
The function of this subprofile is to support the aforementioned SRM applications.

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).

8.2.8.11.1.1 Performance Additions Overview

Figure 158: "Block Server Performance Subprofile Summary Instance Diagram" 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 158: Block Server Performance Subprofile Summary Instance Diagram



<u>Note:</u> 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 8.2.8.11.9, "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).

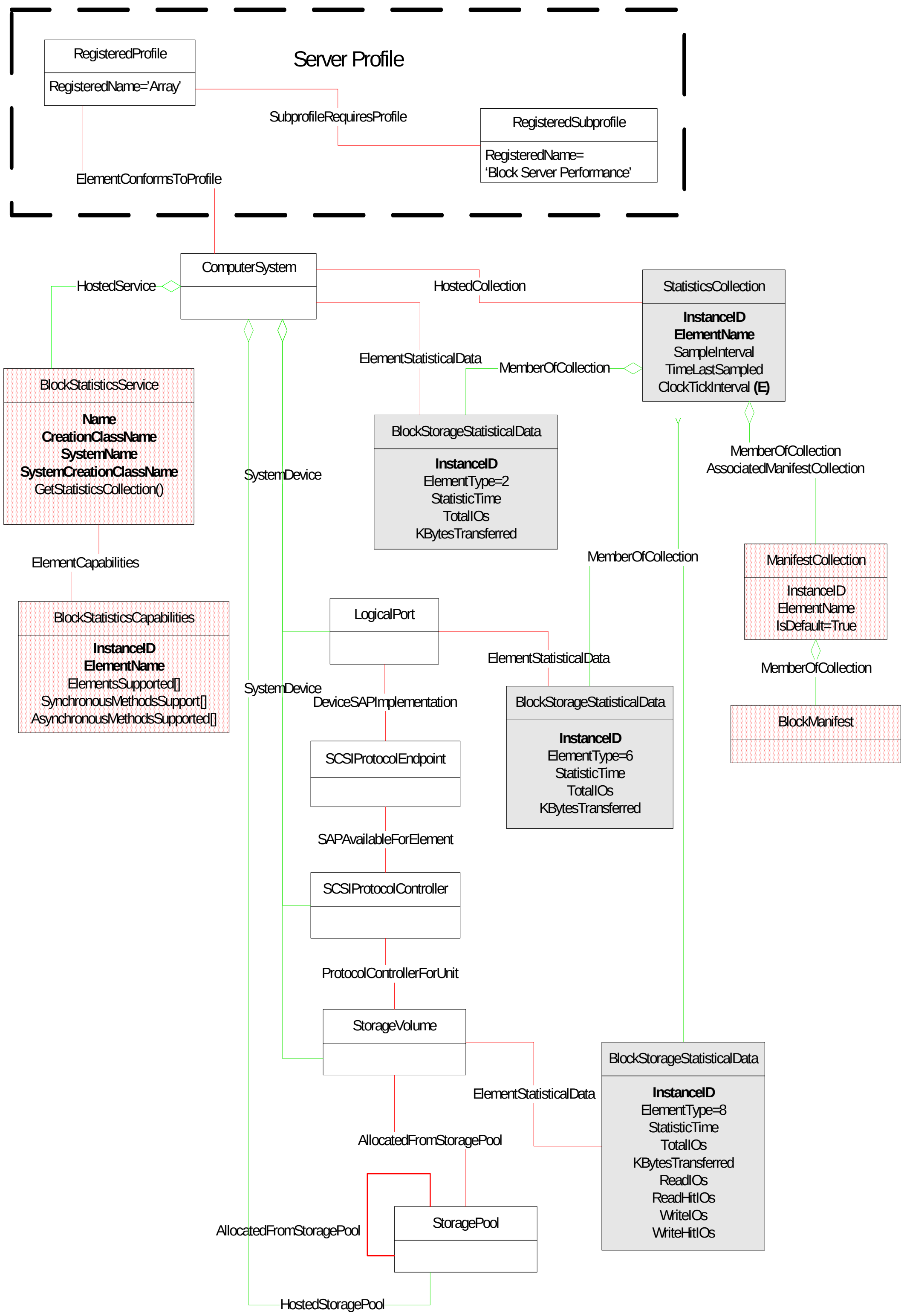
Finally, Figure 168: "Block Server Performance Manifest Collections" illustrates the BlockStatisticsService for Bulk retrieval of all the statistics data and creation of manifest collections. These methods will be discussed later. They are shown here for completeness. Associated with the BlockStatisticsService 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).

8.2.8.11.1.2 Performance Additions to base Array Profile

Figure 159: "Base Array Profile Block Server Performance Instance Diagram" 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 159: 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 8.2.8.11.9, "CIM Elements".

8.2.8.11.1.3 Performance Additions to base Storage Virtualizer Profile

Figure 160: "Base Storage Virtualizer Profile Block Server Performance Instance Diagram" 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.

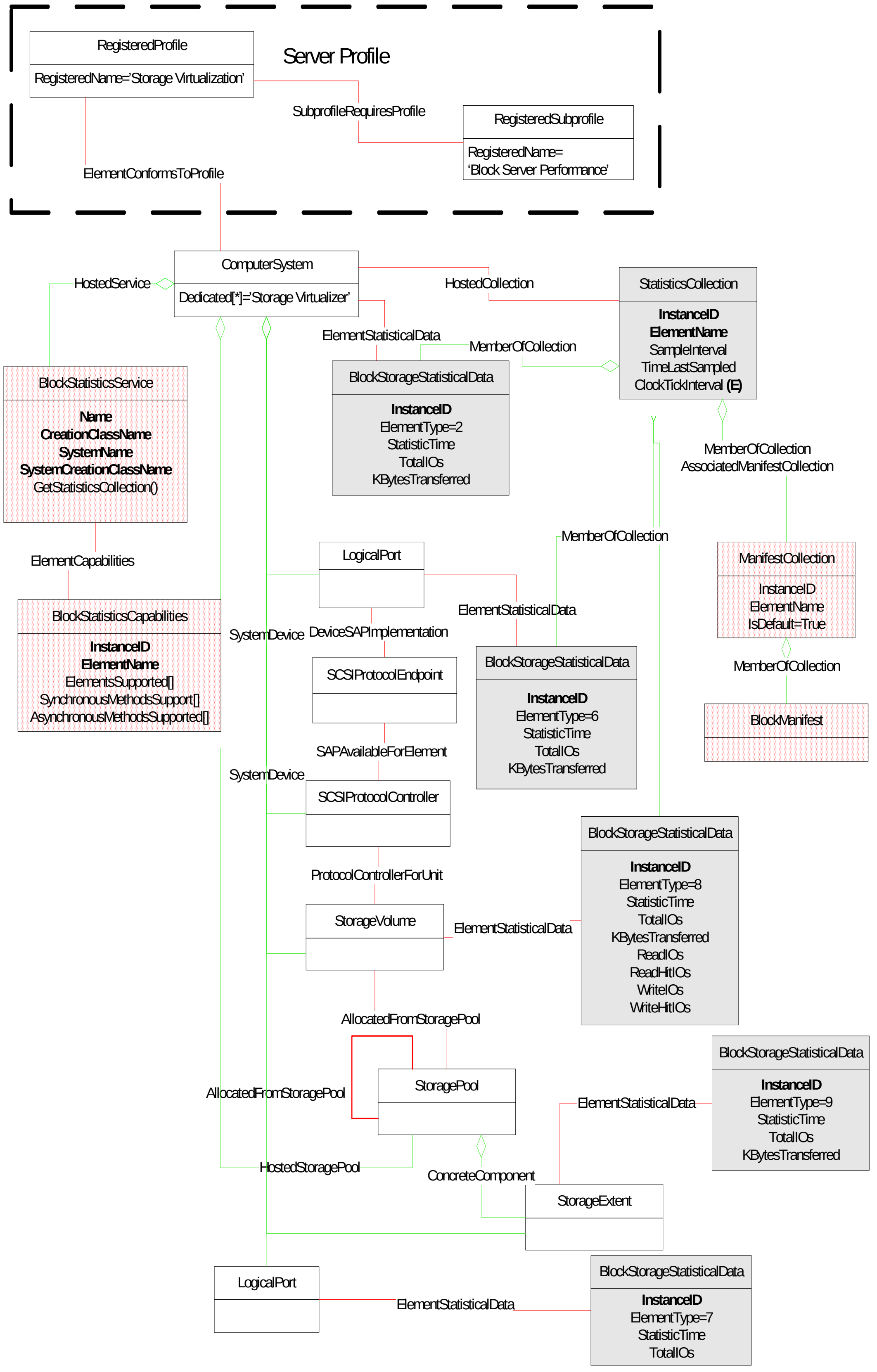
Only the GetStatisticsCollection method of the BlockStatisticsService would be supported. The actual elements for which the statistics would be kept would be reported in the “ElementsSupported” property of the BlockStatisticsCapabilities instance.

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 8.2.8.11.9, "CIM Elements".

8.2.8.11.1.4 Performance Additions to base Volume Management Profile

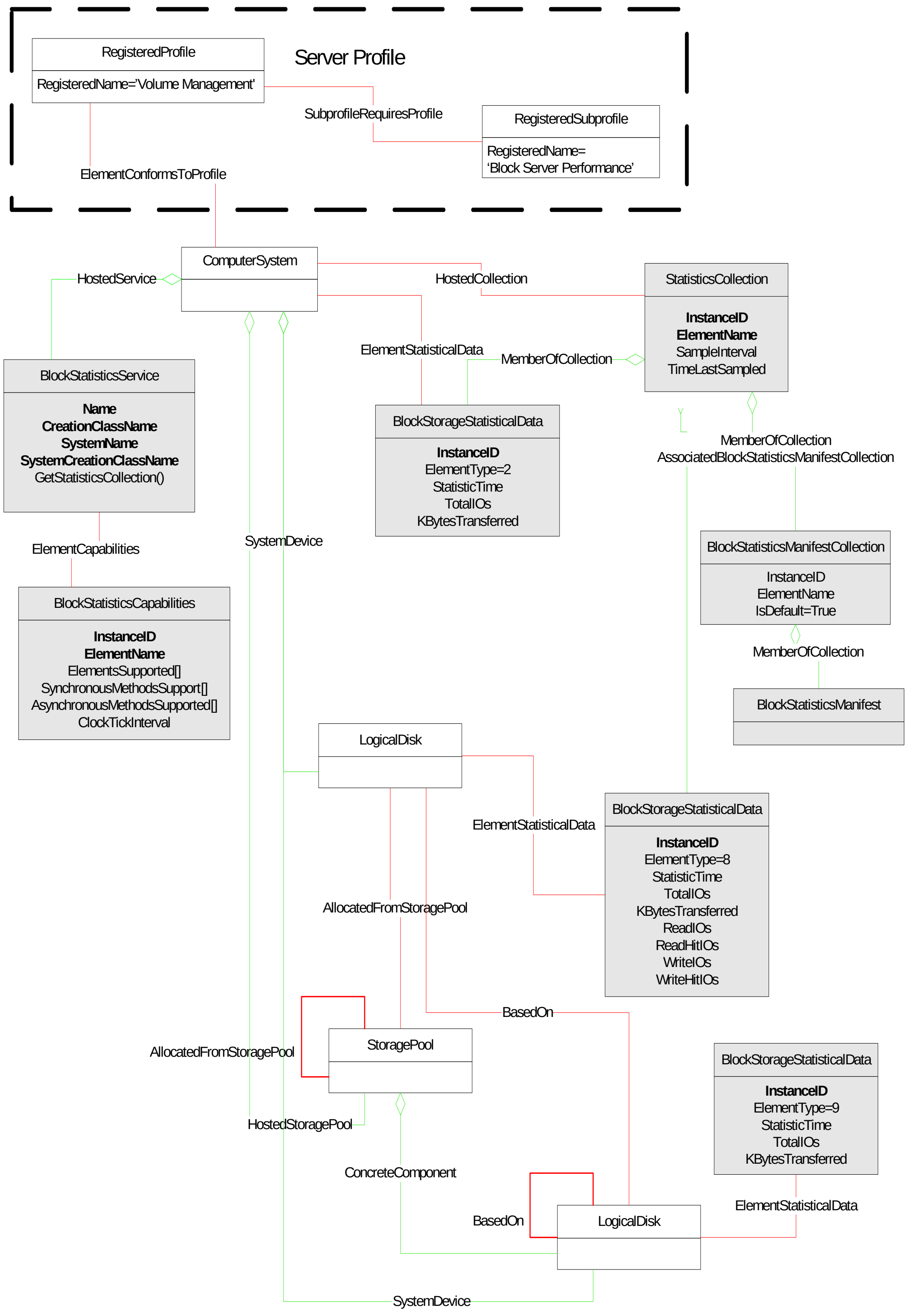
Figure 161: "Base Volume Management Profile Block Server Performance Instance Diagram" illustrates the class instances that would be supported if the volume manager only implemented the base Volume Management Profile and the Block Server Performance Subprofile. Only the StatisticsCollection, the BlockStorageStatisticalData 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 160: Base Storage Virtualizer Profile Block Server Performance Instance Diagram



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 161: 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 8.2.8.11.9, "CIM Elements".

8.2.8.11.1.5 Summary of BlockStorageStatisticsData support by Profile

Table 1025: "Summary of Element Types by Profile" defines the Element Types (for BlockStorageStatisticalData instances) that may be supported by profile.

Table 1025: 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	NO	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.

8.2.8.11.1.6 Server Profile Support for the Block Server Performance Subprofile

At the top of Figure 159: "Base Array Profile Block Server Performance Instance Diagram" 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.

8.2.8.11.1.7 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 a 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.

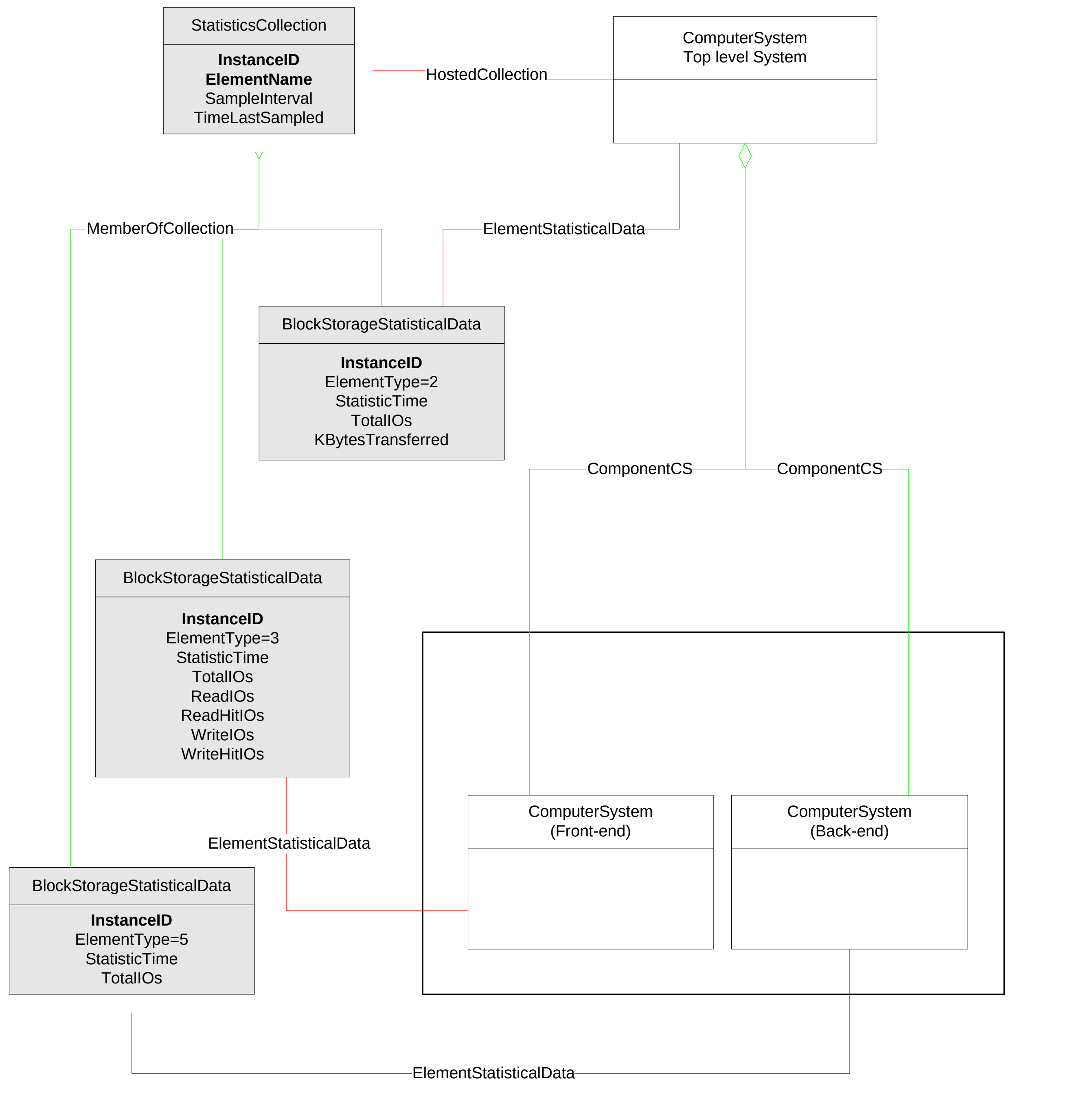
8.2.8.11.1.8 Performance Additions applied to Multiple Computer Systems

Figure 162: "Multiple Computer System Subprofile Block Server Performance Instance Diagram" 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 162: 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 8.2.8.11.9, "CIM Elements".

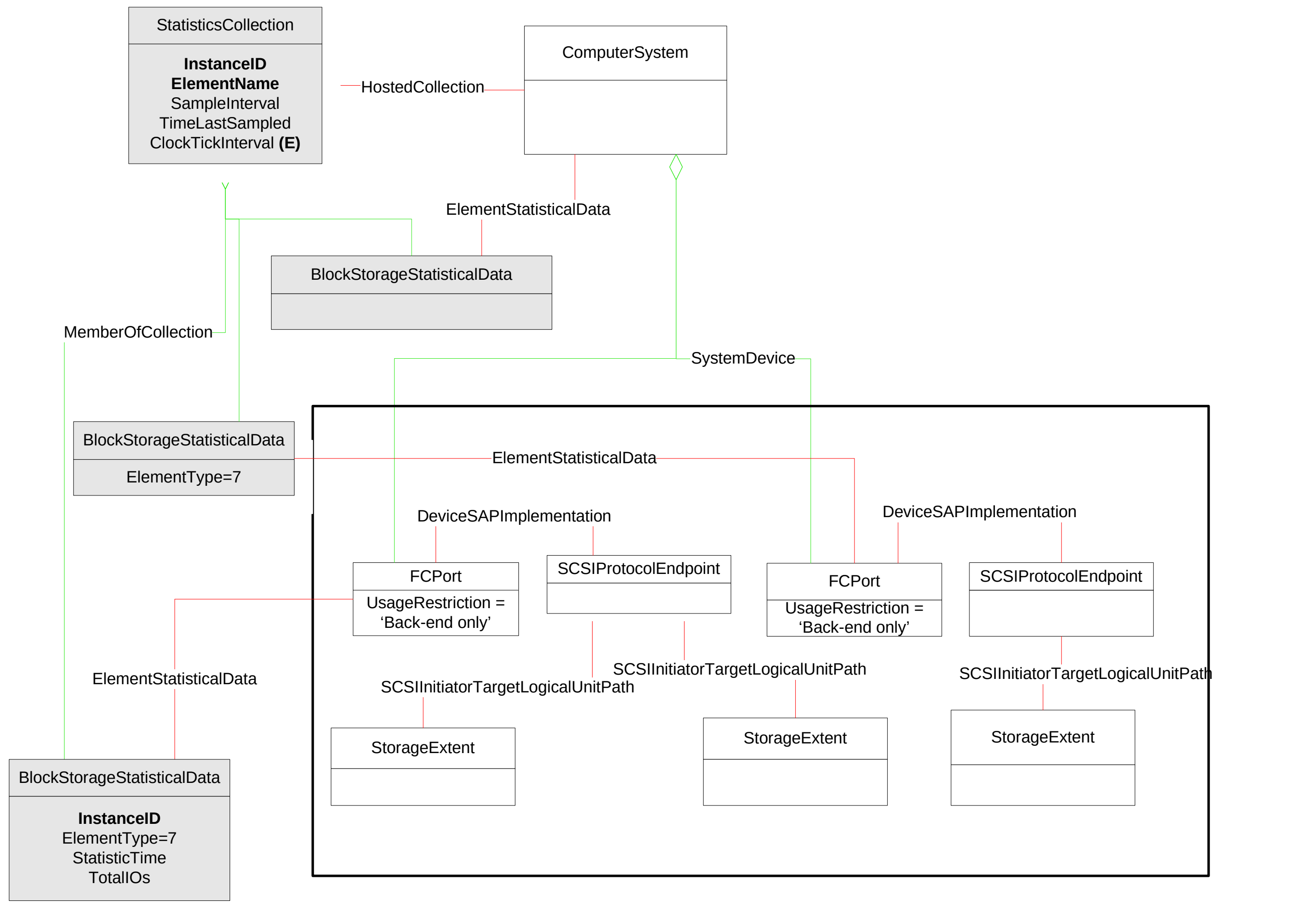
8.2.8.11.1.9 Performance Additions to Backend Ports

Figure 163: "Fibre Channel Initiator Port Subprofile Block Server Performance Instance Diagram" 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 163: **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 the "8.2.8.11.9, "CIM Elements" section.

8.2.8.11.1.10 Performance Additions to Extent Composition

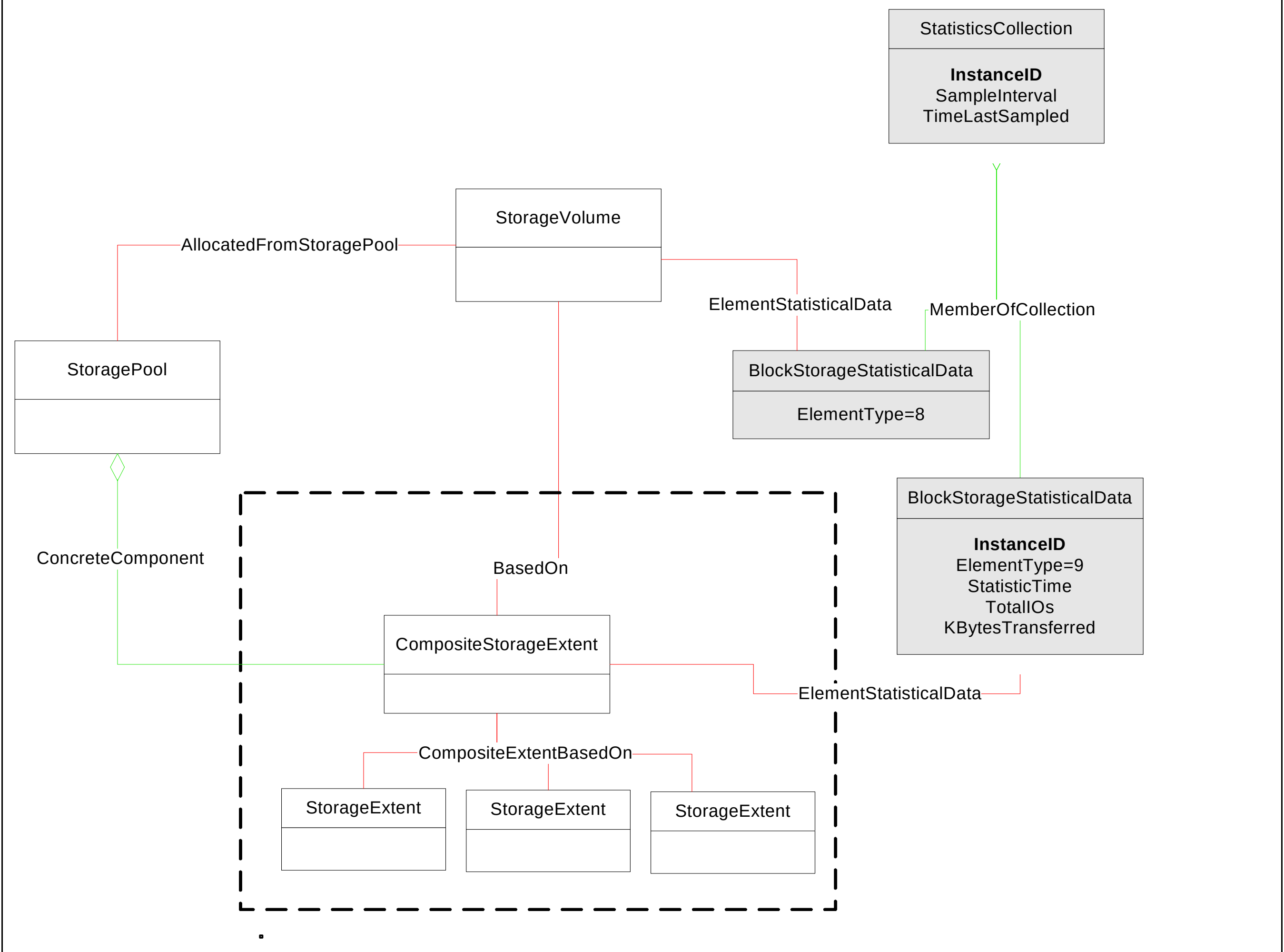
Figure 164: "Extent Composition Subprofile Block Server Performance Instance Diagram" illustrates the class instances that would be supported if an Array also implemented the Extent Composition Subprofile (and the Block Server Performance Subprofile). In this case, BlockStorageStatisticalData instances 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 164: 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 8.2.8.11.9, "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.

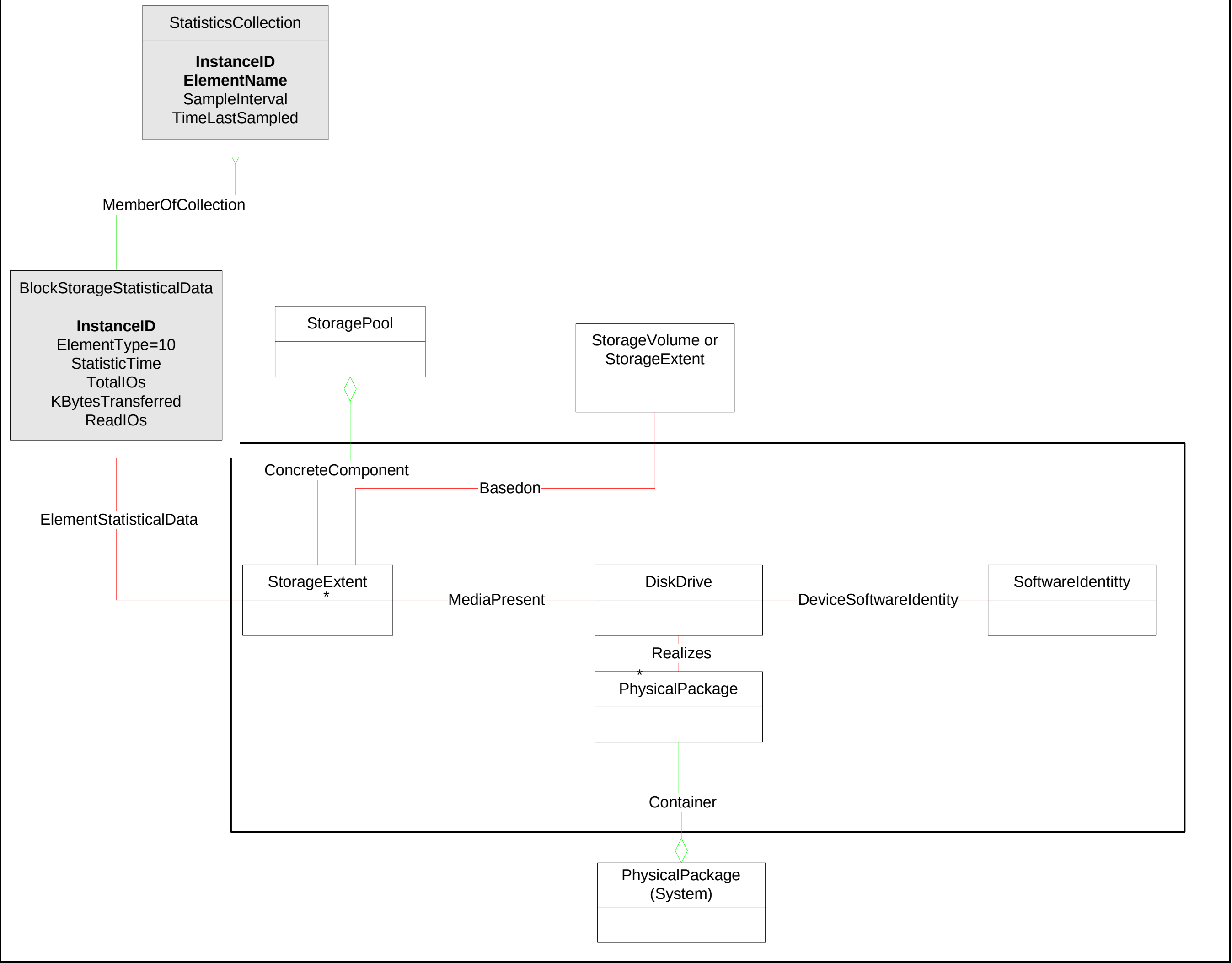
8.2.8.11.1.11 Performance Additions to Disk Drives

Figure 165: "Disk Drive Lite Subprofile Block Server Performance Instance Diagram" 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 Storage Virtualizer and Volume Management Profiles would NEVER show the “Disks” statistics. Also note that an Array 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 165: 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 8.2.8.11.9, "CIM Elements".

8.2.8.11.1.12 Performance Additions to SCSIArbitraryLogicalUnits (Controller LUNs)

Figure 166: SCSIArbitraryLogicalUnit Block Server Performance Instance Diagram

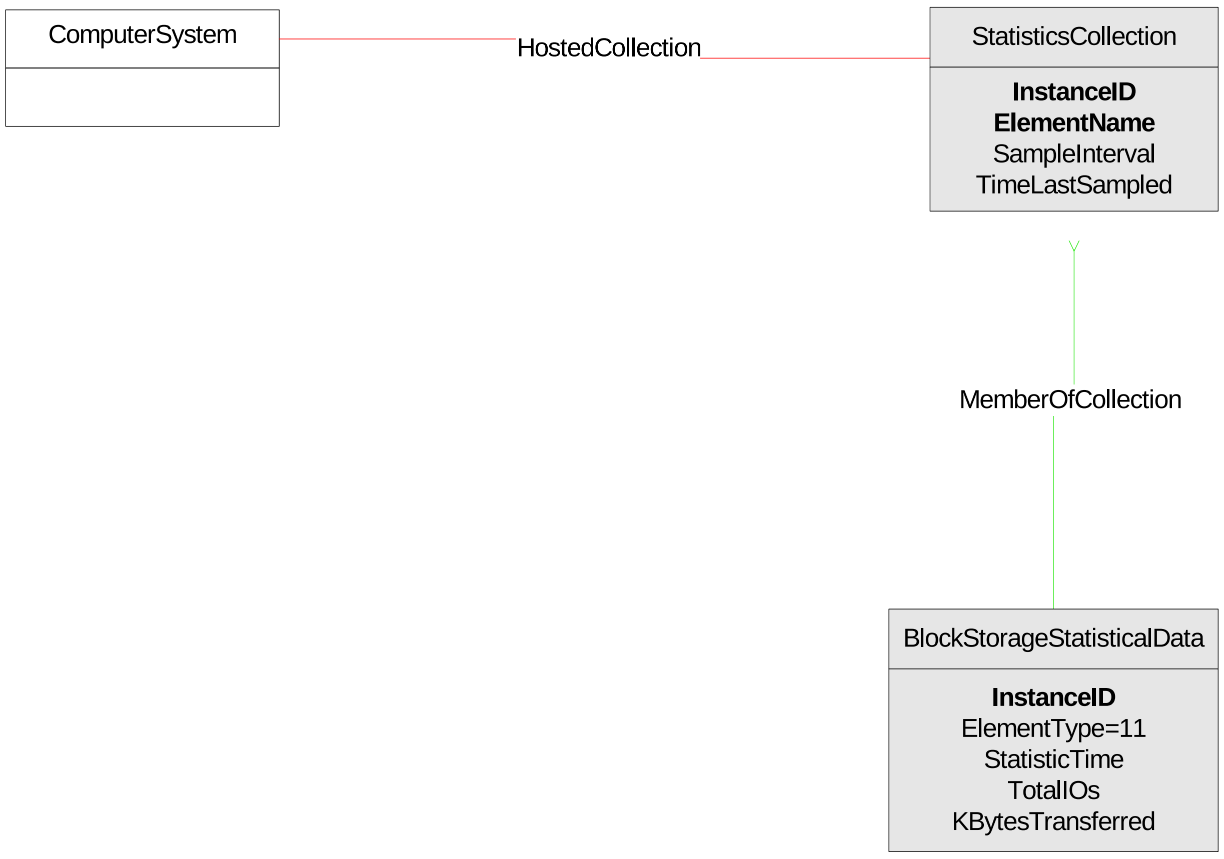


Figure 166: "SCSIArbitraryLogicalUnit Block Server Performance Instance Diagram" illustrates the class instances that would be supported if an Array (or Storage Virtualizer) has Controller LUNs (e.g., SCSIArbitraryLogicalUnits). In this case, BlockStorageStatisticalData instances would exist for each 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 8.2.8.11.9, "CIM Elements".

EXPERIMENTAL

8.2.8.11.1.13 Performance Additions for Remote Mirrors

Figure 167: Remote Mirrors Block Server Performance Instance Diagram

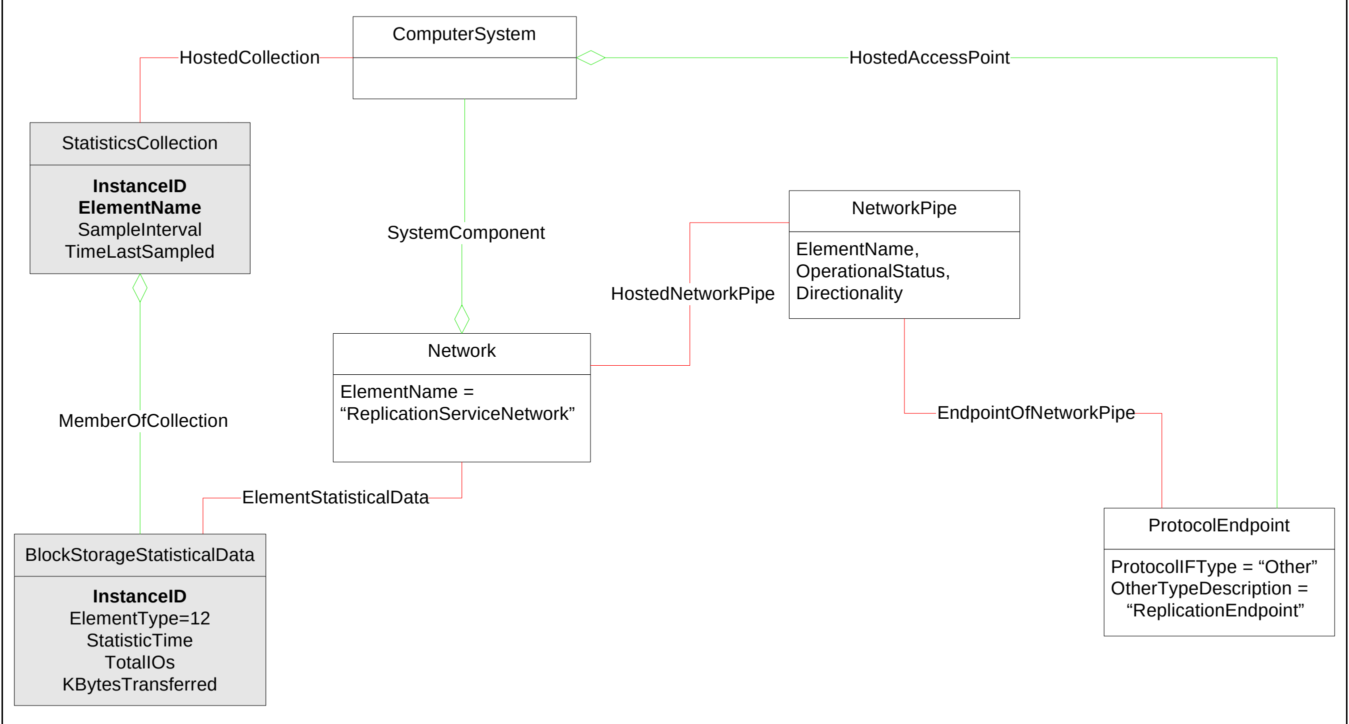


Figure 167: "Remote Mirrors Block Server Performance Instance Diagram" illustrates the class instances that would be supported if an Array also implemented the Remote Mirroring of the Copy Services Subprofile (and the Block Server Performance Subprofile). In this case, BlockStorageStatisticalData instances 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 the "8.2.8.11.9, "CIM Elements" section.

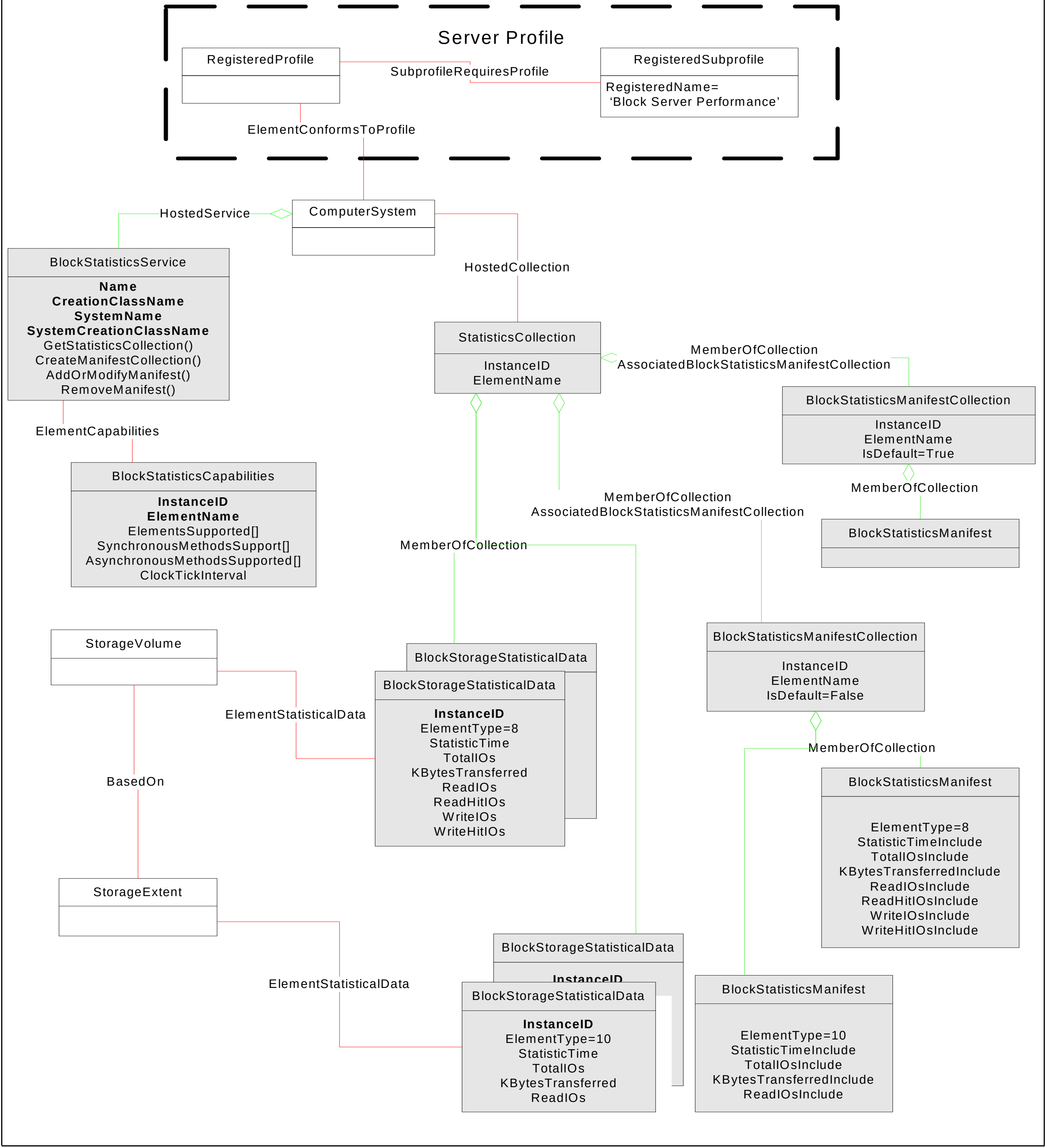
EXPERIMENTAL

8.2.8.11.1.14 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 8.2.8.11.1.7, "Default Manifest Collection".

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 168: "Block Server Performance Manifest Collections".

Figure 168: 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 168: "Block Server Performance Manifest Collections"). 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.

8.2.8.11.1.15 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.

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”

SynchronousMethodsSupported

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

AsynchronousMethodsSupported

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

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.

8.2.8.11.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.11.3 Cascading Considerations

Not applicable.

8.2.8.11.4 Supported Subprofiles and Packages

Table 1026: Supported Subprofiles for Block Server Performance

Registered Subprofile Names	Mandatory	Version
Multiple Computer System	No	1.1.0
Extent Composition	No	1.1.0
SPI Target Ports	No	1.1.0
FC Target Ports	No	1.1.0
iSCSI Target Ports	No	1.1.0
DA Target Ports	No	1.1.0
SPI Initiator Ports	No	1.1.0
FC Initiator Ports	No	1.1.0
iSCSI Initiator Ports	No	1.1.0
Disk Drive Lite	No	1.1.0
Copy Services	No	1.1.0

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.

8.2.8.11.5 Methods of the Profile

8.2.8.11.5.1 Extrinsic Methods of the Profile

8.2.8.11.5.1.1 Overview

The methods supported by this subprofile are summarized in Table 1027: “Creation, Deletion and Modification Methods in Block Server Performance Subprofile”, and detailed in the sections that follow it.

Table 1027: Creation, Deletion and Modification Methods in Block Server Performance Subprofile

Method	Created Instances	Deleted Instances	Modified Instances
GetStatisticsCollection	None	None	None
CreateManifestCollection	BlockStatisticsManifest-Collection AssociatedBlockStatis-ticsManifestCollection	None	None

Table 1027: Creation, Deletion and Modification Methods in Block Server Performance Subprofile (Continued)

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

8.2.8.11.5.1.2 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 this 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 this 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>
```

8.2.8.11.5.1.3 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" }

8.2.8.11.5.1.4 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.

8.2.8.11.5.1.5 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" }

8.2.8.11.5.2 Intrinsic Methods of the Profile

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

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).

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

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.

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.

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.

EXEC QUERY

This is one of the ways of retrieving statistics.

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 this 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 this 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 this 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 this 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

8.2.8.11.6 Client Considerations and Recipes

8.2.8.11.6.1 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 ($BlockManifest.IncludeReadHitIOTimeCounter)
     #numIncluded++
    if ($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>
         }

         // 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) {
            #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
        $Manifest.IncludeWriteHitIOTimeCounter = false
        } else if ($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]
    #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

8.2.8.11.6.2 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 its 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(),
         “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_LogicalDevice”,
      “Dependent”,
      “Antecedent”,
      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,
      #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

8.2.8.11.6.3 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”,
    “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 ($VolumeManifest.IncludeStatisticTime) {
     #StatisticTime = VolumeStats.StatisticTime
    } else {
     <ERROR! StatisticTime is a required property for Volumes and should
        be set to “true” in the default BlockManifest>
    }
    if ($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
           }
       }
```

8.2.8.11.6.4 Summary of Statistics Support by Element

Not all statistics properties are kept for all elements. Table 1028 illustrates the statistics properties that are kept for each of the metered elements.

Table 1028: Summary of Block 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 (LogicalDisk)	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
IOTime-Counter	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
ReadIOTime-Counter	O	O	O	N	N	N	O	N	O
ReadHitIO TimeCounter	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
WriteIOTime-Counter	O	O	O	N	N	N	O	N	O
WriteHitIO TimeCounter	O	O	O	N	N	N	O	N	N
KBytesWritten	O	O	O	O	N	N	O	N	O
IdleTime-Counter	N	N	N	O	O	N	O	O	O
MaintOp	N	N	N	N	N	N	N	O	O
MaintTime-Counter	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 1028:, "Summary of Block Statistics Support by Element"5: 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 an 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.

8.2.8.11.6.5 Formulas and Calculations

Table 1028:, "Summary of Block Statistics Support by Element" 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 1029: “Formulas and Calculations” 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 1029: Formulas and Calculations

Calculated Statistics	
New statistic	**Formula**
TimeInterval	delta StatisticTime
% utilization	100 * (delta StatisticTime - delta IdleTime)/ delta StatisticTime
I/O rate	delta TotalIOs / delta StatisticTime
I/O response time	delta IOTime / delta TotalIOs
Queue depth	delta I/O rate * delta I/O response time
Service Time	utilization / I/O rate
Wait Time	Response Time - Service Time
Average Read Size	delta KBytesRead / delta ReadIOs
Average Write Size	delta KBytesWritten / delta WriteIOs
% Read	100 * (delta ReadIOs / delta TotalIOs)
% Write	100 * (delta WriteIOs / delta TotalIOs)
% Hit	100 * ((delta ReadHitIOs + delta WriteHitIOs) / delta TotalIOs)

8.2.8.11.6.6 Block Server Performance Supported Capabilities Patterns

The Capabilities patterns summarized in Table 1030: “Block Server Performance Subprofile Supported Capabilities Patterns” are formally recognized by the Block Server Performance Subprofile of this version of SMI-S

Table 1030: 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

Table 1030: Block Server Performance Subprofile Supported Capabilities Patterns (Continued)

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).

8.2.8.11.7 Registered Name and Version

Block Server Performance version 1.1.0

8.2.8.11.8 CIM Server Requirements

Table 1031: CIM Server Requirements for Block Server Performance

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.11.9 CIM Elements

Table 1032: CIM Elements for Block Server Performance

Element Name	Description
Mandatory Classes	
CIM_AssociatedBlockStatisticsManifestCollection (8.2.8.11.9.1)	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_BlockStatisticsCapabilities (8.2.8.11.9.3)	This defines the statistics capabilities supported by the implementation of the profile.
CIM_BlockStatisticsManifest (8.2.8.11.9.4)	An instance of this class defines the statistics properties supported by the profile implementation for one element type.
CIM_BlockStatisticsManifestCollection (8.2.8.11.9.6)	An instance of this class defines the predefined collection of default block statistics manifests (one manifest for each element type).
CIM_BlockStatisticsService (8.2.8.11.9.8)	This is a Service that provides (optional) services of bulk statistics retrieval and manifest set manipulation methods.
CIM_BlockStorageStatisticalData (8.2.8.11.9.9)	This is a Subclass of CIM_StatisticalData for Block servers. It would be instantiated as specific block statistics for particular components.
CIM_ElementCapabilities (8.2.8.11.9.10)	This associates the BlockStatisticsCapabilities to the BlockStatisticsService.
CIM_ElementStatisticalData (8.2.8.11.9.11)	This associates a BlockStorageStatisticalData instance to the element for which the statistics are collected.
CIM_HostedCollection (8.2.8.11.9.12)	This would associate the StatisticsCollection to the top level system for the profile (e.g., array).
CIM_HostedService (8.2.8.11.9.13)	This associates the BlockStatisticsService to the ComputerSystem that hosts it.
CIM_MemberOfCollection (8.2.8.11.9.14)	This would associate all block statistics instances to the StatisticsCollection.
CIM_MemberOfCollection (8.2.8.11.9.15)	This would associate pre-defined Manifests to default manifest collection.
CIM_StatisticsCollection (8.2.8.11.9.17)	This would be an collection point for all Statistics that are kept for a Block Server.
Optional Classes	
CIM_AssociatedBlockStatisticsManifestCollection (8.2.8.11.9.2)	This is an association between the StatisticsCollection and a client defined manifest collection.
CIM_BlockStatisticsManifest (8.2.8.11.9.5)	An instance of this class defines the statistics properties of interest to the client for one element type.
CIM_BlockStatisticsManifestCollection (8.2.8.11.9.7)	An instance of this class defines one client defined collection of block statistics manifests (one manifest for each element type).
CIM_MemberOfCollection (8.2.8.11.9.16)	This would associate Manifests to client defined manifest collections.

8.2.8.11.9.1 CIM_AssociatedBlockStatisticsManifestCollection

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: true

Table 1033: SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Statistics		CIM_StatisticsCollection	The StatisticsCollection to which the manifest collection applies
ManifestCollection		CIM_BlockStatisticsManifestCollection	The manifest collection.

8.2.8.11.9.2 CIM_AssociatedBlockStatisticsManifestCollection

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(s): CreateManifestCollection
Modified By : Static
Deleted By : DeleteInstance
Class Mandatory: false

Table 1034: SMI Referenced Properties/Methods for CIM_AssociatedBlockStatisticsManifestCollection (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Statistics		CIM_StatisticsCollection	The StatisticsCollection to which the manifest collection applies
ManifestCollection		CIM_BlockStatisticsManifestCollection	The manifest collection.

8.2.8.11.9.3 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: true

Table 1035: SMI Referenced Properties/Methods for CIM_BlockStatisticsCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
ElementTypesSupported		uint16[]	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"}
SynchronousMethodsSupported		uint16[]	ValueMap { "2", "3", "4", "5", "6", "7", "8"}, Values {"Exec Query", "Indications", "QueryCollection", "GetStatisticsCollection", "Manifest Creation", "Manifest Modification", "Manifest Removal" }
ClockTickInterval		uint64	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 monotanically 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.
Optional Properties/Methods			
AsynchronousMethodsSupported		uint16[]	Not supported in current version of SMI-S.

8.2.8.11.9.4 CIM_BlockStatisticsManifest

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: true

Table 1036: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A Provider defined string that identifies the manifest in the context of the Default Manifest Collection.
InstanceID		string	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.

ElementType		uint16	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		boolean	
IncludeTotalIOs		boolean	
IncludeKBytesTransferred		boolean	
IncludeIOTimeCounter		boolean	
IncludeReadIOs		boolean	
IncludeReadHitIOs		boolean	
IncludeReadIOTimeCounter		boolean	
IncludeReadHitIOTimeCounter		boolean	
IncludeKBytesRead		boolean	
IncludeWriteIOs		boolean	
IncludeWriteHitIOs		boolean	
IncludeWriteIOTimeCounter		boolean	
IncludeWriteHitIOTimeCounter		boolean	
IncludeKBytesWritten		boolean	
IncludeIdleTimeCounter		boolean	
IncludeMaintOp		boolean	
IncludeMaintTimeCounter		boolean	

8.2.8.11.9.5 CIM_BlockStatisticsManifest

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 instance, AND a client must have created at least ONE instance of CIM_BlockStatisticsManifestCollection.

Created By : Extrinsic(s): AddOrModifyManifest
Modified By : Extrinsic(s): AddOrModifyManifest
Deleted By : Extrinsic(s): RemoveManifest
Class Mandatory: false

Table 1037: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ElementName		string	A Client defined string that identifies the manifest.
InstanceID		string	The instance Identification. Within the scope of the instantiating Namespace, InstanceID opaquely and uniquely identifies an instance of this class.

ElementType		uint16	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		boolean	
IncludeTotalIOs		boolean	
IncludeKBytesTransferred		boolean	
IncludeIOTimeCounter		boolean	
IncludeReadIOs		boolean	
IncludeReadHitIOs		boolean	
IncludeReadIOTimeCounter		boolean	
IncludeReadHitIOTimeCounter		boolean	
IncludeKBytesRead		boolean	
IncludeWriteIOs		boolean	
IncludeWriteHitIOs		boolean	
IncludeWriteIOTimeCounter		boolean	

Table 1037: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifest (Client defined)

Property	Flags	Type	Description & Notes
IncludeWriteHitIOTimeCounter		boolean	
IncludeKBytesWritten		boolean	
IncludeIdleTimeCounter		boolean	
IncludeMaintOp		boolean	
IncludeMaintTimeCounter		boolean	

8.2.8.11.9.6 CIM_BlockStatisticsManifestCollection

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: true

Table 1038: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Pre-defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	For the default manifest collection, this should be set to "DEFAULT".
IsDefault		boolean	Denotes whether or not this manifest collection is a provider defined default manifest collection. For the default manifest collection this is set to "true".

8.2.8.11.9.7 CIM_BlockStatisticsManifestCollection

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(s): CreateManifestCollection
Modified By : Static
Deleted By : DeleteInstance

Class Mandatory: false

Table 1039: SMI Referenced Properties/Methods for CIM_BlockStatisticsManifestCollection (Client defined)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	A client defined user-friendly name for the manifest collection. It is set during creation of the Manifest Collection through the ElementName parameter of the CreateManifestCollection method.
IsDefault		boolean	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".

8.2.8.11.9.8 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: true

Table 1040: SMI Referenced Properties/Methods for CIM_BlockStatisticsService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
Name		string	
Optional Properties/Methods			
GetStatisticsCollection()			Support for this method is optional. This method retrieves all statistics kept for the profile as directed by a manifest collection.

Table 1040: SMI Referenced Properties/Methods for CIM_BlockStatisticsService

Property	Flags	Type	Description & Notes
CreateManifestCollection()			Support for this method is optional. This method is used to create client defined manifest collections.
AddOrModifyManifest()			Support for this method is optional. This method is used to add or modify block statistics manifests in a client defined manifest collection.
RemoveManifests()			Support for this method is optional. This method is used to remove a block statistics manifest from a client defined manifest collection.

8.2.8.11.9.9 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 : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1041: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
StatisticTime		datetime	Time statistics table by object was last updated (Time Stamp in CIM 2.2 speci-fication format)
ElementType		uint16	This value is required AND current ver-sion 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 Com-puter System", "Back-end Computer System", "Front-end Port", "Back-end Port", "Volume", "Extent", "Disk Drive", "Arbitrary LUs" , "Remote Replica Group"}
TotalIOs		uint64	The cumulative count of I/Os for the object

Table 1041: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
KBytesTransferred		uint64	The cumulative count of data transferred in KBytes (1024bytes = 1KByte). Note: This is mandatory for the Top level computer system and Front-end Ports, but is optional for the component computer systems and Back-end Ports.
IOTimeCounter		uint64	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. Note: This is not SPECIFIED for CompositeExtents.
ReadIOs		uint64	The cumulative count of all reads. Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, but it is optional for the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports or CompositeExtents.
ReadHitIOs		uint64	The cumulative count of all read cache hits (Reads from Cache). Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, but it is optional for the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports, CompositeExtents or DiskDrives.
ReadIOTimeCounter		uint64	The cumulative elapsed time for all Read I/Os) for all cumulative Read I/Os. Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports or CompositeExtents.

Table 1041: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Property	Flags	Type	Description & Notes
ReadHitIOTimeCounter		uint64	The cumulative elapsed time for all Read I/Os read from cache for all cumulative Read I/Os. Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports, CompositeExtents or DiskDrives.
KBytesRead		uint64	The cumulative count of data read in KBytes (1024bytes = 1KByte). Note: This is not SPECIFIED for Ports or CompositeExtents.
WriteIOs		uint64	The cumulative count of all writes. Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, but it is optional for the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports or CompositeExtents.
WriteHitIOs		uint64	The cumulative count of Write Cache Hits (Writes that went directly to Cache without blocking). Note: This is mandatory for "Front-end" and "Peer" component ComputerSystems, but it is optional for the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports, CompositeExtents or DiskDrives.
WriteIOTimeCounter		uint64	The cumulative elapsed time for all Write I/Os for all cumulative Writes. Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports or CompositeExtents.

Table 1041: SMI Referenced Properties/Methods for CIM_BlockStorageStatisticalData

Property	Flags	Type	Description & Notes
WriteHitIOTimeCounter		uint64	The cumulative elapsed time for all Write I/Os written to cache for all cumulative Write I/Os. Note: This is optional for "Front-end" and "Peer" component ComputerSystems and the Top level computer system and not mandatory for "Back-end" component computer systems. Note: This is not SPECIFIED for Ports, CompositeExtents or DiskDrives.
KBytesWritten		uint64	The cumulative count of data written in KBytes (1024bytes = 1KByte). Note: This is not SPECIFIED for Ports or CompositeExtents.
IdleTimeCounter		uint64	The cumulative elapsed idle time using ClockTickInterval units (Cumulative Number of Time Units for all idle time in the array). Note: This is optional for "Back-end" component ComputerSystems and not mandatory for the Top level computer system and for "Front-end" and "Peer" other component computer systems. Note: This is not SPECIFIED for Front-end Ports.
MaintOp		uint64	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. Note: This is not SPECIFIED for ComputerSystems, Ports or StorageVolumes.
MaintTimeCounter		uint64	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. Note: This is not SPECIFIED for ComputerSystems, Ports or StorageVolumes.

8.2.8.11.9.10 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: true

Table 1042: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element (BlockStatisticsService)
Capabilities		CIM_Capabilities	The Capabilities instance associated with the element.

8.2.8.11.9.11 CIM_ElementStatisticalData

CIM_ElementStatisticalData is an association that relates a ManagedElement (StorageVolume, LogicalDisk, FCPort or 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 ManagedElement.

CIM_ElementStatisticalData is not subclassed from anything.

Created By : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 1043: SMI Referenced Properties/Methods for CIM_ElementStatisticalData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	A reference to a StorageVolume, LogicalDisk, FCPort, or ComputerSystem for which the Statistics apply
Stats		CIM_StatisticalData	A reference to the BlockStorageStatisticalData that hold the statistics for the managed element.

8.2.8.11.9.12 CIM_HostedCollection

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 indicate that the StatisticsCollection presents an aspect of the top level Computer System.

CIM_HostedCollection is subclassed from CIM_HostedDependency.

Created By : Static
Modified By : Static

Deleted By : Static
Class Mandatory: true

Table 1044: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The top level ComputerSystem of the profile
Dependent		CIM_SystemSpecificCollection	The StatisticsCollection

8.2.8.11.9.13 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: true

Table 1045: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting System.
Dependent		CIM_Service	The Service hosted on the System.

8.2.8.11.9.14 CIM_MemberOfCollection

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 : External
Modified By : Static
Deleted By : External
Class Mandatory: true

Table 1046: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Collect Statistics)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The StatisticsCollection
Member		CIM_ManagedElement	An individual Statistics Instance that is part of the set.

8.2.8.11.9.15 CIM_MemberOfCollection

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: true

Table 1047: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Collect pre-defined manifests)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The default manifest collection
Member		CIM_ManagedElement	The individual Manifest Instance that is part of the set.

8.2.8.11.9.16 CIM_MemberOfCollection

This use of MemberOfCollection is to Collect all Manifests instances in a client defined manifest collection.

Created By : Extrinsic(s): AddOrModifyManifest
Modified By : Static
Deleted By : Extrinsic(s): RemoveManifest
Class Mandatory: false

Table 1048: SMI Referenced Properties/Methods for CIM_MemberOfCollection (Collect client defined manifests)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	A client defined manifest collection
Member		CIM_ManagedElement	The individual Manifest Instance that is part of the set.

8.2.8.11.9.17 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: true

Table 1049: SMI Referenced Properties/Methods for CIM_StatisticsCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	
SampleInterval		datetime	Minimum recommended polling interval for an array, storage virtualizer system or volume manager. It is set by the provider and cannot be modified.

Table 1049: SMI Referenced Properties/Methods for CIM_StatisticsCollection

Property	Flags	Type	Description & Notes
TimeLastSampled		dateTime	Time statistics table by object was last updated (Time Stamp in SMI 2.2 specification format)

8.2.8.11.10 Related Standards

Table 1050: Related Standards for Block Server Performance

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.12 Copy Services Subprofile

This subprofile contains both standard and experimental content. Experimental sections, clauses and paragraphs are marked with beginning and ending tags. Most experimental content is associated with two new features: remote replication and delta snapshot management.

This version of the copy services subprofile maintains backward compatibility with the preceding version. See “Backward Compatibility” in 8.2.8.12.1 in the instrumentation section for a description of maintaining backward compatibility with the earlier version.

8.2.8.12.1 Description

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 "Managing Background Copy" in 8.2.8.12.5.

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. The subprofile includes optional management techniques such as space consumption limits for delta replicas based on the variable model.

EXPERIMENTAL

The Copy Services Subprofile provides the following SMI-S management disciplines summarized in Table 1051: “Copy Services Subprofile FCAPS Support”.

Table 1051: Copy Services Subprofile FCAPS Support

Level / FCAPS	Fault Mgmt	Configuration Mgmt	Accounting Mgmt	Performance Mgmt	Security Mgmt
Application					
File/Record					
Block		YES			
Connectivity	YES	YES			
Device					

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 169: "Copy Services Discovery".

A provider exposes one instance of StorageReplicationCapabilities for each CopyType 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.

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.

UnSyncAssoc: Creates an un-synchronized 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.

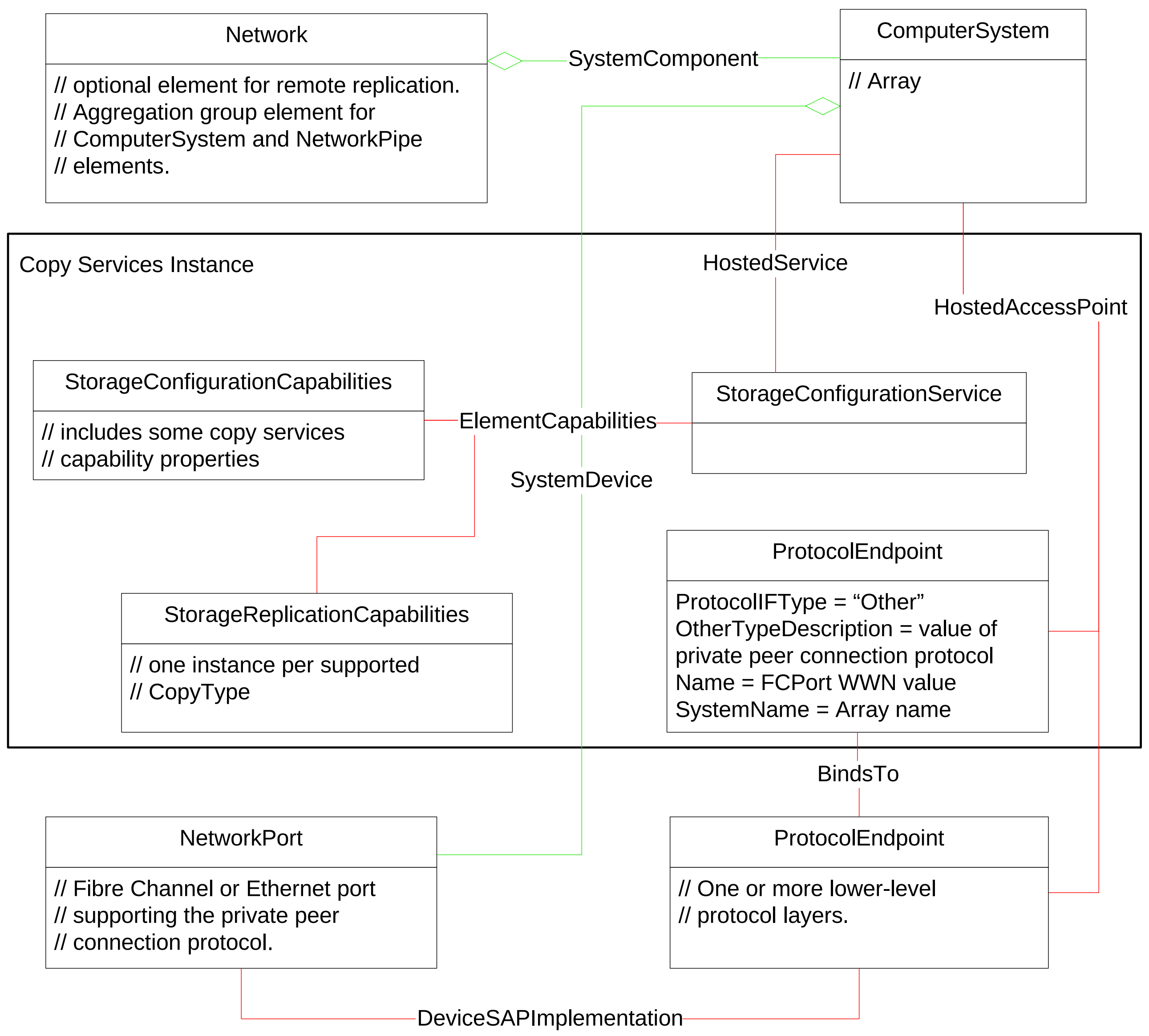
UnSyncUnAssoc: Creates an un-synchronized clone of the source element and does not maintain the source association after completing the copy operation.

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.

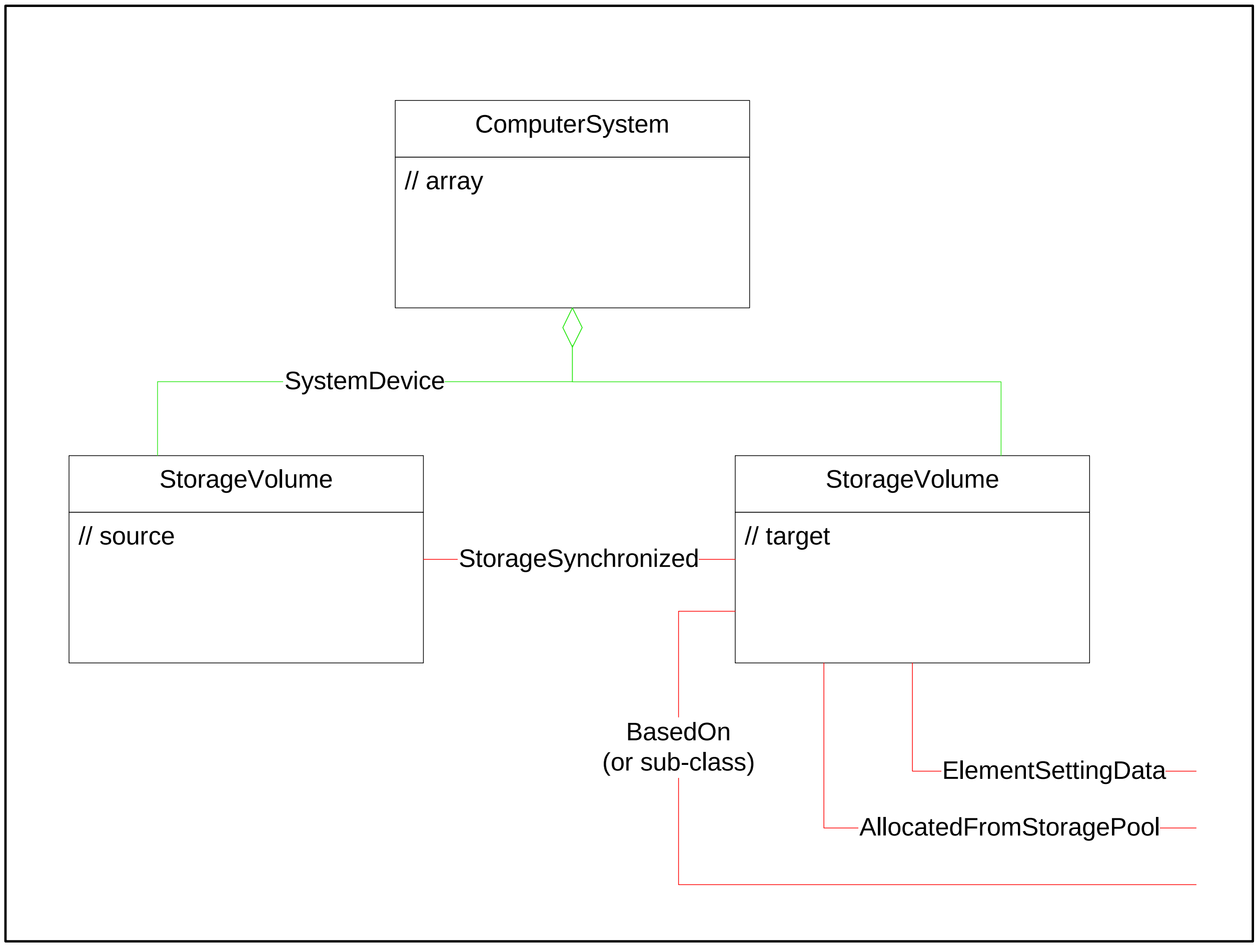
Figure 169: Copy Services Discovery


Network
// optional element for remote replication.
// Aggregation group element for
// ComputerSystem and NetworkPipe
// elements.
SystemComponent
ComputerSystem
// Array
Copy Services Instance
HostedService
HostedAccessPoint
StorageConfigurationCapabilities
// includes some copy services
// capability properties
ElementCapabilities
StorageConfigurationService
SystemDevice
ProtocolEndpoint
ProtocolIFType = “Other”
OtherTypeDescription = value of
private peer connection protocol
Name = FCPort WWN value
SystemName = Array name
StorageReplicationCapabilities
// one instance per supported
// CopyType
BindsTo
NetworkPort
// Fibre Channel or Ethernet port
// supporting the private peer
// connection protocol.
ProtocolEndpoint
// One or more lower-level
// protocol layers.
DeviceSAPImplementation

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 170: "Local Replica" shows the basic model of a local replica.

Figure 170: 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.

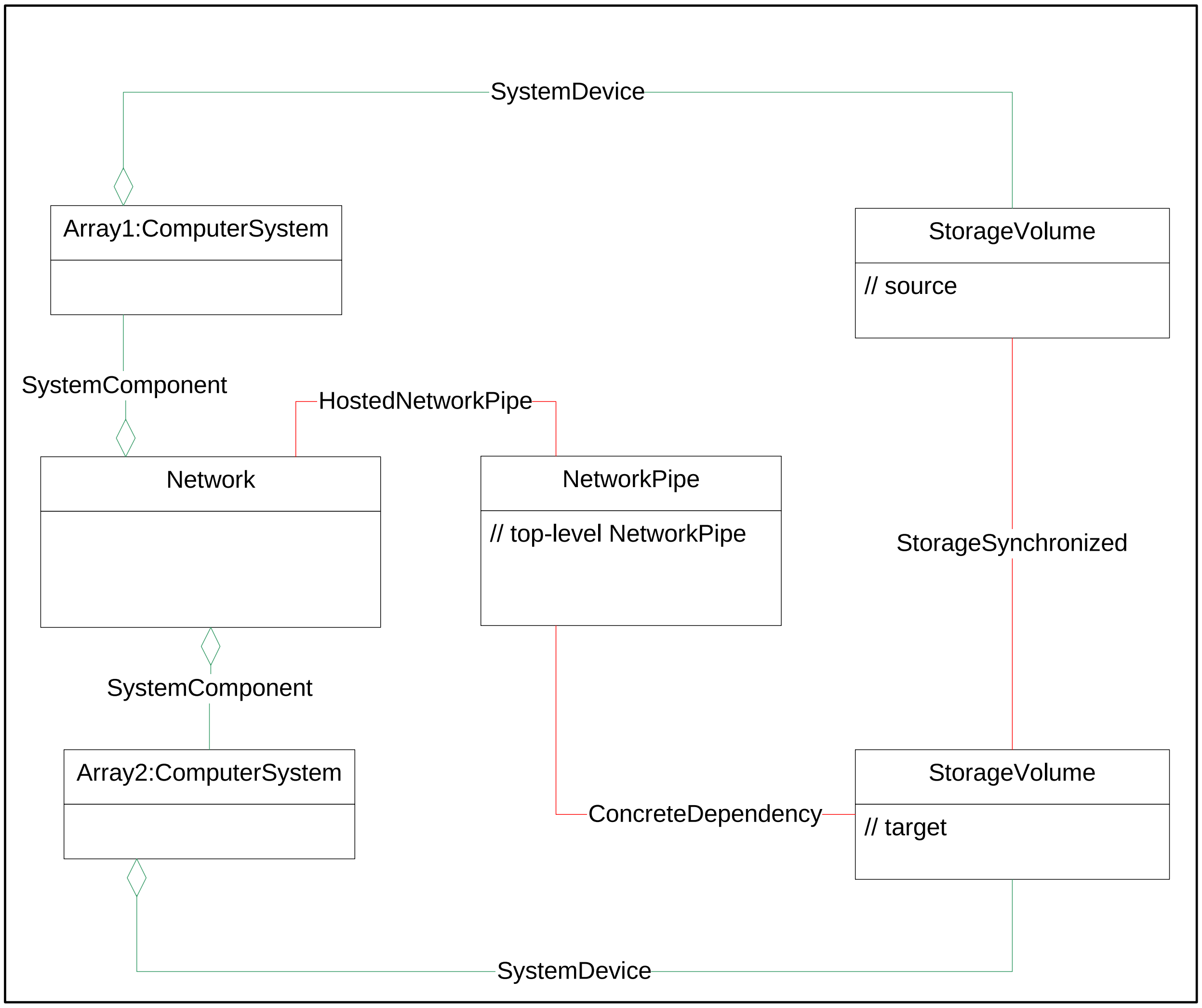
EXPERIMENTAL

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 NetworkPipe element as shown in Figure 171: "Remote Mirror Replica".

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.

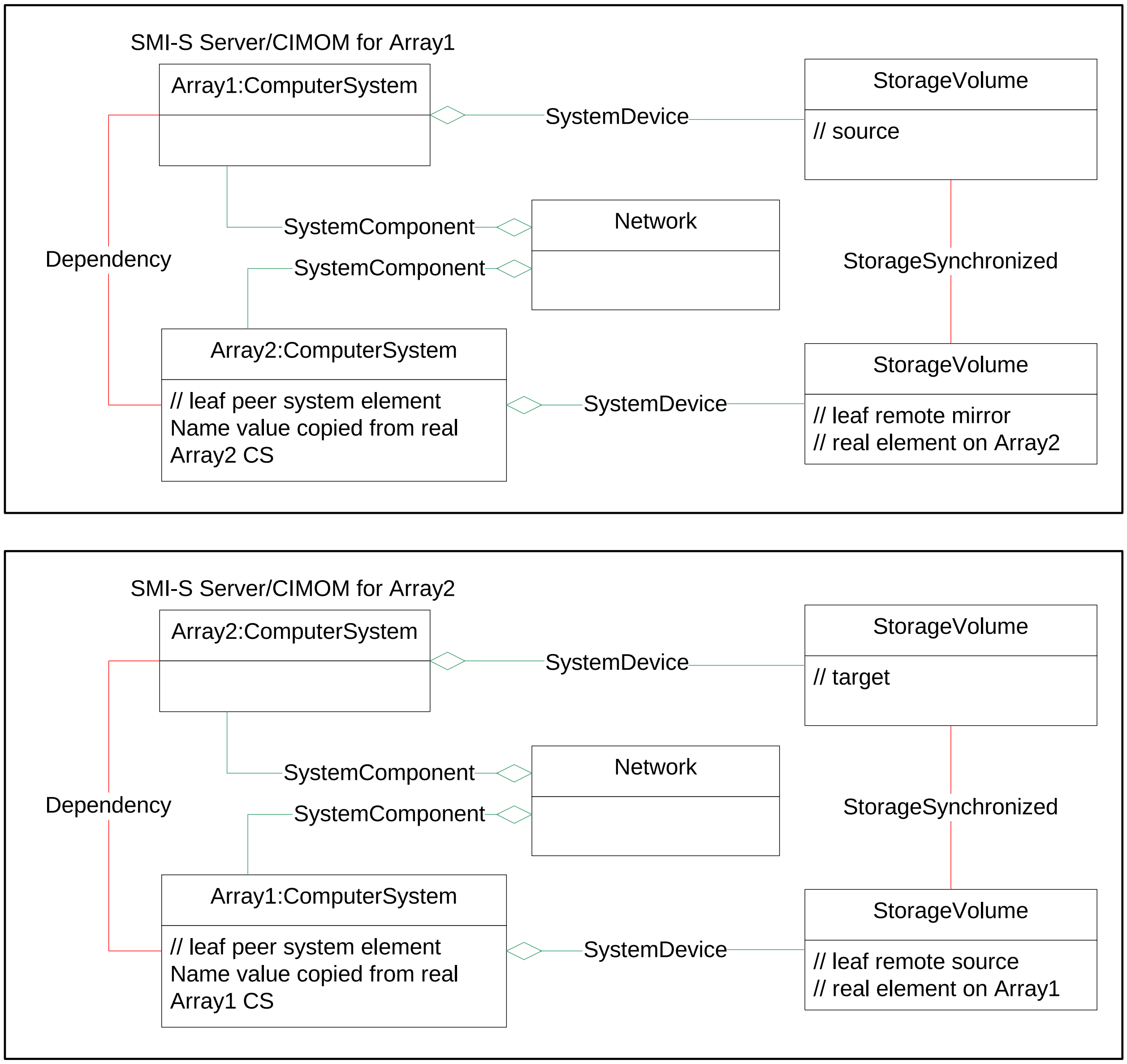
Figure 171: "Remote Mirror Replica" shows the basic model of a remote mirror replica when both source and target elements are controlled by one SMI-S server:

Figure 171: Remote Mirror Replica



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 172: "Cascading the Copy Services Subprofile".

Figure 172: Cascading the Copy Services Subprofile

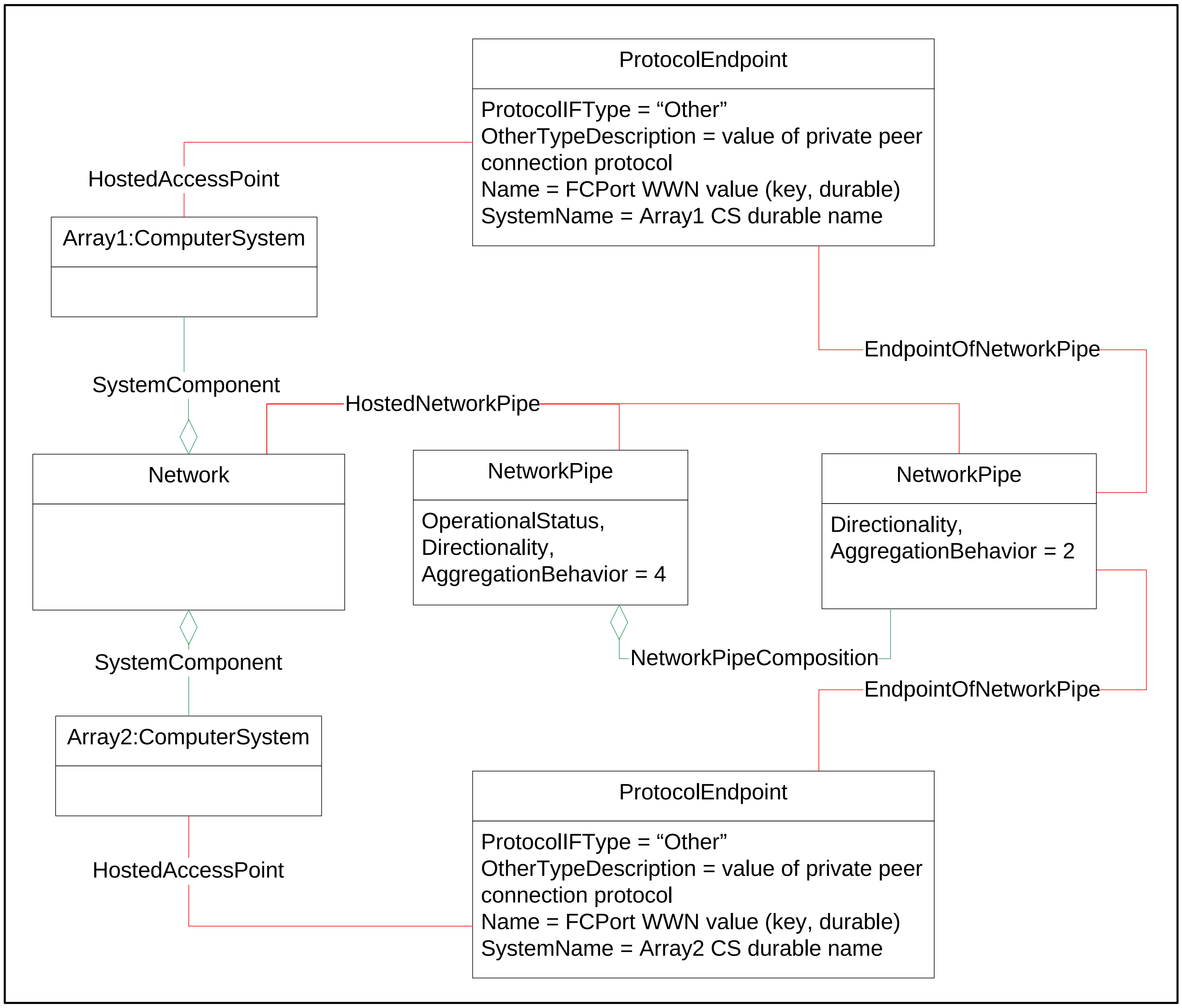


The provider ensures that leaf elements representing real instances of ComputerSystem, ProtocolEndpoint and an element such as StorageVolume are created in each CIMOM. Correlatable name properties and other key properties are copied from the real elements to the leaf elements. The provider shall ensure 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 NetworkPipe elements that identify connections. NetworkPipe instances can be static or can be created by extrinsic method invocation. All connections (NetworkPipe instances) and ComputerSystem instances supporting managed connections are associated to this Network element.

Figure 173: "Peer-to-Peer Connection" shows the complete model of a peer-to-peer connection:

Figure 173: Peer-to-Peer 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 must 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.

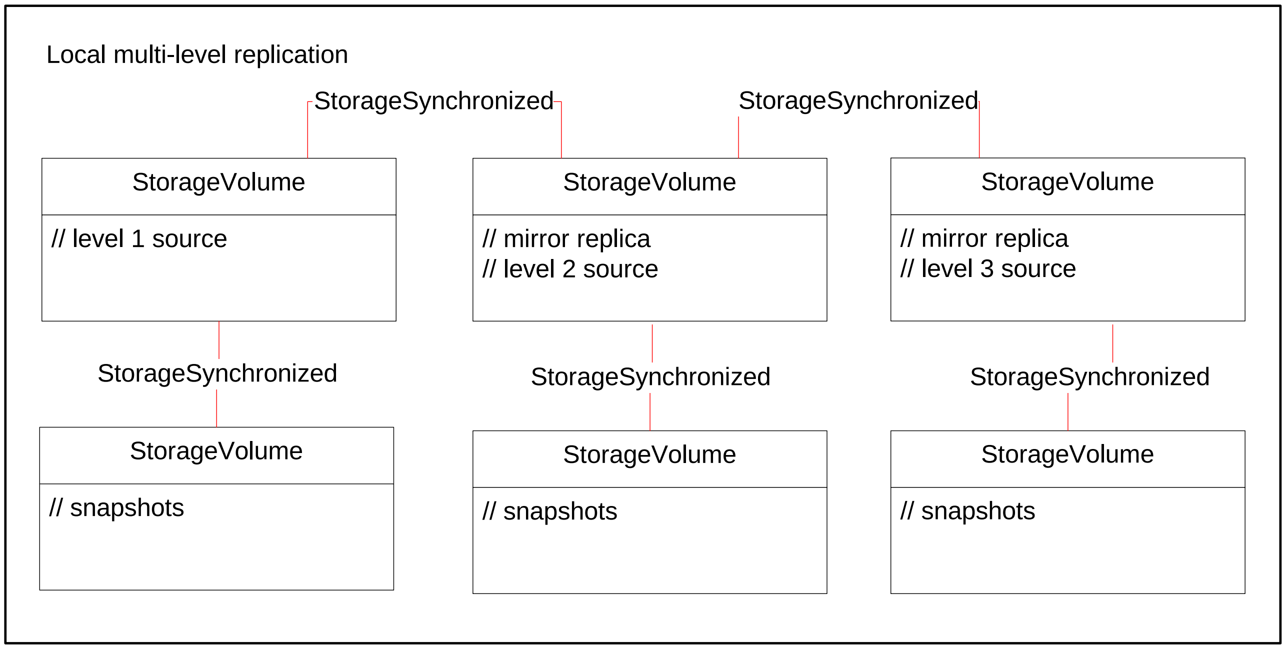
The subprofile allows use of either point-to-point or switched topologies for connections. Point-to-point connections are static connections that may be discovered and monitored by a client. Switched connections are dynamic connections and a client can manage the performance and availability characteristics 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 NetworkPipe composition. The top-level NetworkPipe 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 NetworkPipe 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

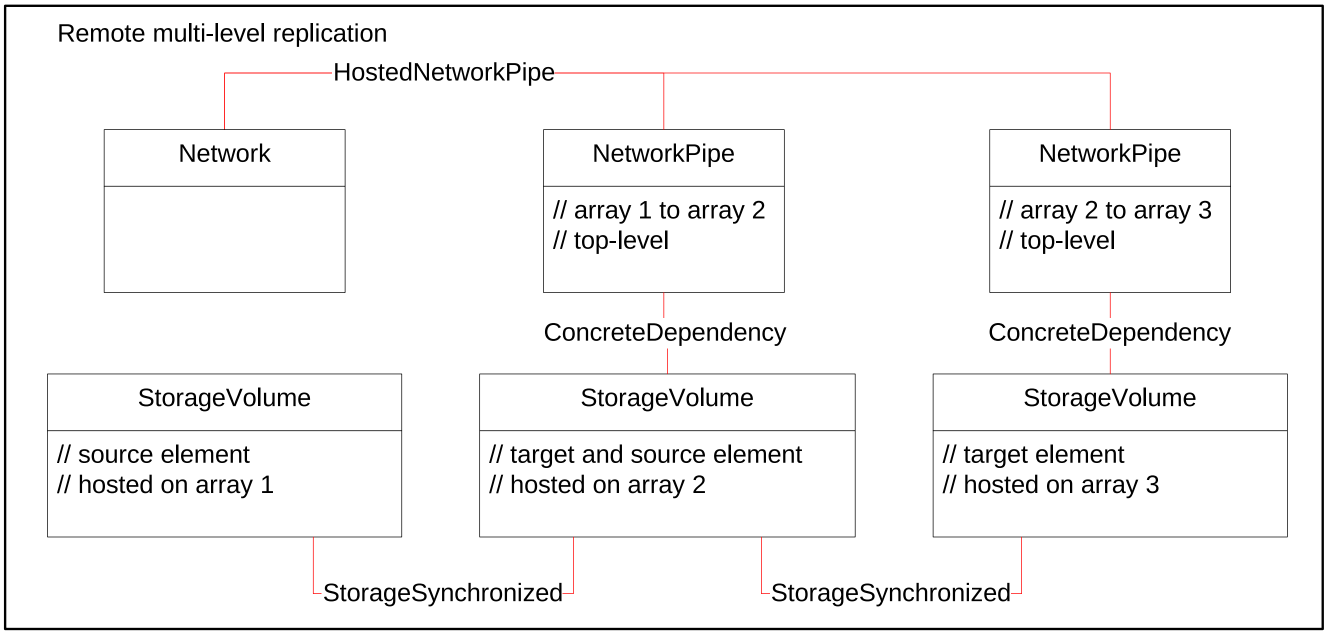
The subprofile supports both multiple replicas per associated source element and multi-level replication. Properties in StorageReplicationCapabilities allow the provider to indicate the maximum number of replicas for one source element and the maximum depth for multi-level replication. Figure 174: "Multi-Level Local Replication" and Figure 175: "Multi-Level Remote Replication" show the basic models for local multi-level replication and remote multi-level replication.

Figure 174: 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 175: "Multi-Level Remote Replication":

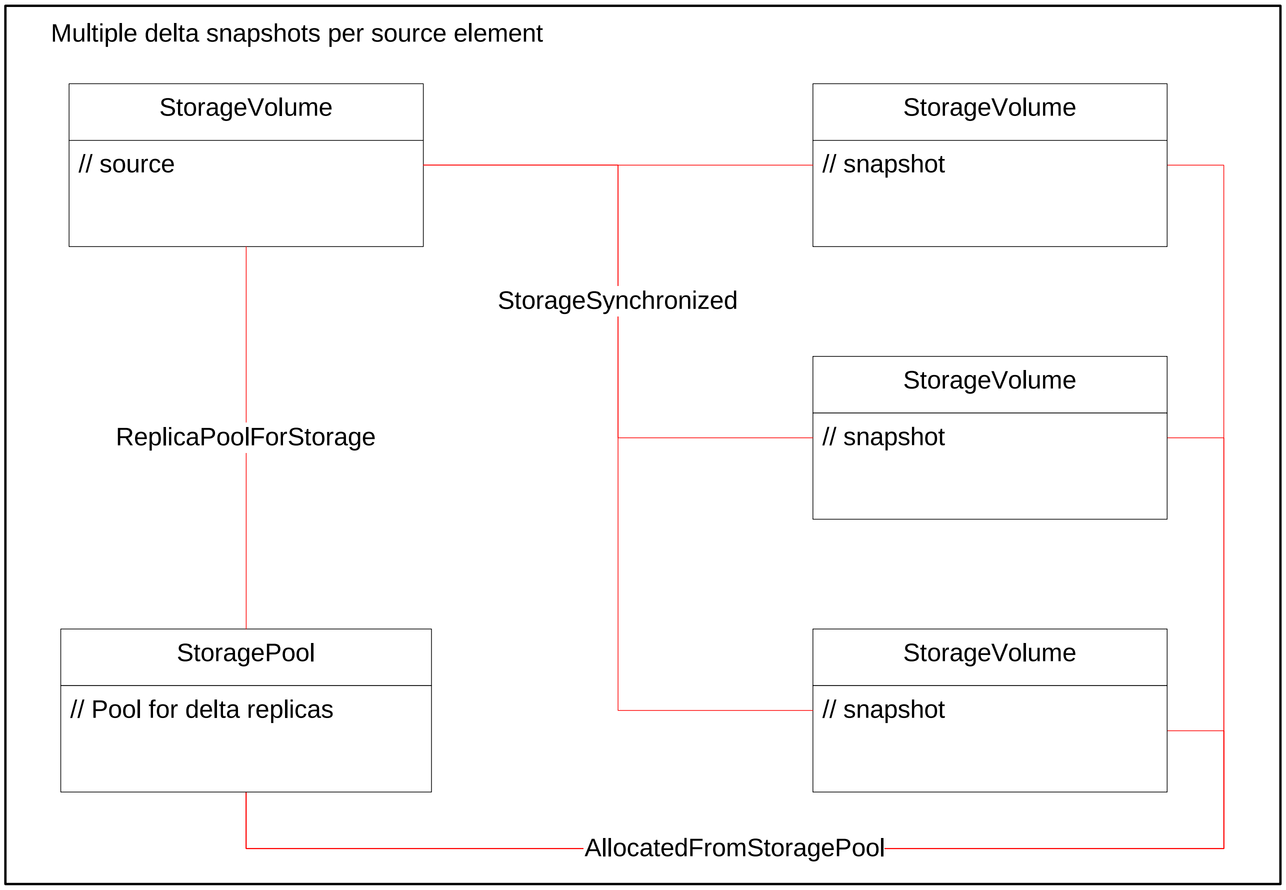
Figure 175: Multi-Level Remote Replication



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.

Figure 176: "Multi-Level Snapshots" shows the basic model of snapshots created as delta replicas.

Figure 176: Multi-Level Snapshots



EXPERIMENTAL

The optional capabilities are

- Specialized storage pools to contain delta replicas
- Space limits and warning thresholds
- Space reservation for delta replicas

These capabilities are described in detail 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.

EXPERIMENTAL

8.2.8.12.1.1 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.

8.2.8.12.1.2 Instrumentation Requirements

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.

8.2.8.12.1.3 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.

<u>State Management For Associated Replicas</u>

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 un-associated 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 1052: “Synchronization Operation Support Requirements” shows the classification.

Table 1052: 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 1053: “SyncState Values”.

Table 1053: SyncState Values

Synchronization State (SyncState 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	
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 HostAccesibleState[] 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 Table 1054. The table does not apply to CopyType “UnSyncUnAssoc”.

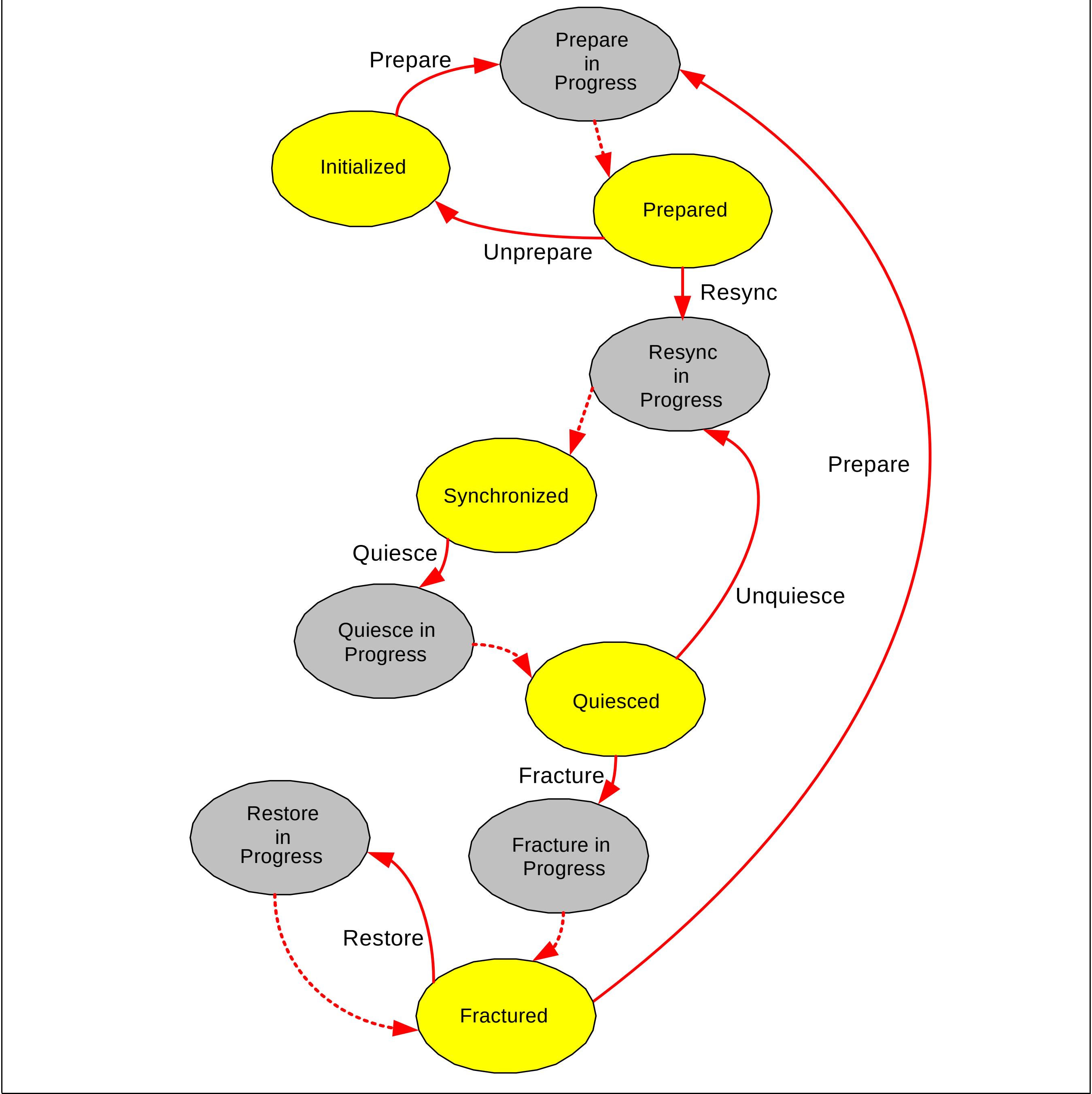
Table 1054: SyncMaintained and WhenSynced Properties

Synchronization State	SyncMaintained		WhenSynced	
	Sync/Async	UnSyncAssoc	Sync/Async	UnSyncAssoc
Initialized	True or False	True or False	Null	Date/Time frozen
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 177: "State Transitions for Mirrors" shows state transitions for mirrors:

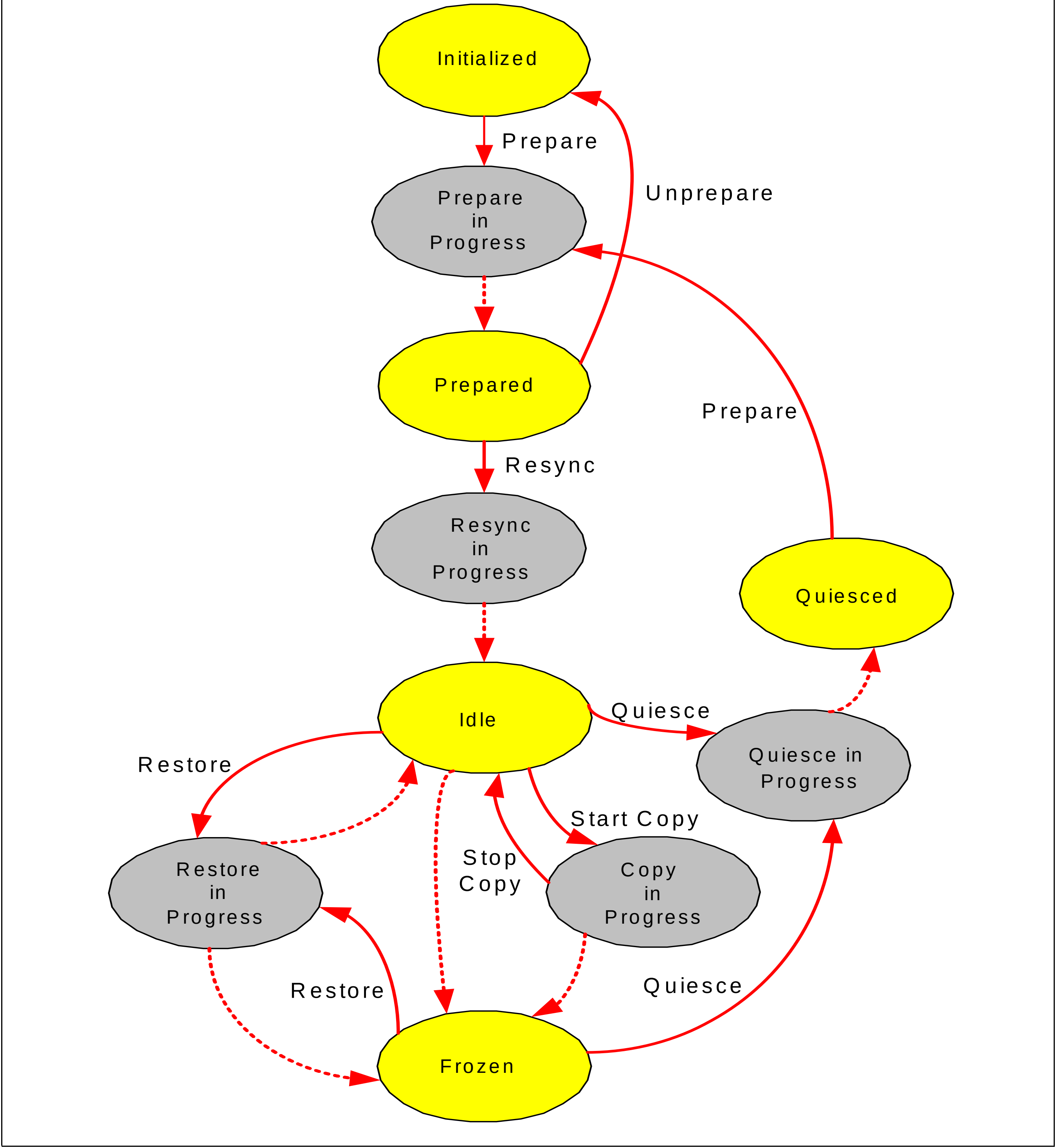
Figure 177: State Transitions for Mirrors



The replication state machine is entered as a result of invoking any of the copy services extrinsic methods that create a StorageSynchronized association. Exit the state machine by invoking the ModifySynchronization Detach operation.

Figure 178: "State Transitions for Snapshots" shows state transitions for snapshots:

Figure 178: State Transitions for Snapshots



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.

8.2.8.12.1.4 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

8.2.8.12.1.5 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 of the replication properties allow an initial value of “not applicable” if the provider does not use the property. The provider should set the SupportedSpecializedElements[] value list in StorageReplicationCapabilities to indicate which values of StorageSetting.IntendedUsage are supported by the provider.

A provider may require specialized pools to contain delta replicas, specialized elements as replica targets and specialized extents 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 elements and make them available for discovery by clients. In either case, the IntendedUsage property in StorageSetting shall be supported by the provider as part of the goal parameter for pool/element creation methods and pool search methods such as GetAvailableExtents.

When IntendedUsage is set for GetAvailableExtents, GetSupportedSizes or GetSupportedSizeRange, the value constrains the method provider. GetAvailableExtents should only return extents eligible to create specialized replica pools. GetSupportedSizes and GetSupportedSizeRange should only return non-null values if the selected pool is allowed to contain replica elements.

When IntendedUsage is set in the goal parameter for an element creation method, the value acts as an additional parameter indicating a special element subtype. The provider ensures that the required element type is created and IntendedUsage is retained in the persistent setting element associated with the new replica element.

8.2.8.12.1.6 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 failback.

The client managing a remote replication service shall discover both the source system and target system instances. The instance of StorageReplicationCapabilities for CopyType values “Sync” and “Async” 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

8.2.8.12.1.7 Backward Compatibility

A 1.1 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 1.1 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”.

8.2.8.12.1.8 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.

8.2.8.12.2 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 replicas are supported with either space limits or special pools with warning thresholds, certain alert indications may be generated by the provider:

- Alert indication when a delta replica reaches the space limit warning threshold.
- Alert indication when a delta replica attempts to exceed its space limit or is unable to consume space from its associated pool.
- Alert indication 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 1056: “Copy Services Alert Indications”.

Instance deletion indications as faults are supported for StorageSynchronized when a delta snapshot is automatically deleted for exceeding its space limit.

If remote replication is supported using managed connections, the provider shall generate instance modifications when OperationalStatus values change for a top-level NetworkPipe 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 Table 1055: “OperationalStatus Values for NetworkPipe”are supported for NetworkPipe.OperationalStatus[]:

Table 1055: OperationalStatus Values for NetworkPipe

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 [reserved for future use]
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 NetworkPipe 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. Alerts are generated for CIM_AllocatedFromStoragePool associations to indicate that a delta replica has consumed space either to a warning threshold level or to an allowable limit. The SpaceConsumed, SpaceLimit and SpaceLimitWarningThreshold can be analyzed to determine an appropriate response.

Table 1056: Copy Services Alert Indications

AlertingManaged Element	PerceivedSeverity	ProbableCause	ProbableCauseDescription
Replica pool association	Minor (4)	Threshold Crossed (52)	Delta replica at space warning threshold: SpaceConsumed/SpaceLimit
Replica pool association	Major (5)	Out of Memory (33)	Delta replica at space limit
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 1057: “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 1057: 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

8.2.8.12.3 Cascading Considerations

The Copy Services Subprofile is both a cascading subprofile and a leaf subprofile. Any remote replication provider that supports the cascading role shall also support the leaf role.

Cascading shall be supported by remote replication providers that allow separate SMI-S servers and CIMOMs for service access to the source system and the target system hosting remote replica elements.

A cascaded provider design does not eliminate the need for the client to interface to both SMI-S servers when managing remote replication operations. Cascading eliminates the need for the client to invoke a method twice in order to complete a single operation. The cascading provider discovers a leaf provider as necessary when the client invokes a method with cascading privileges. A cascading provider sets the RemoteReplicationServicePointAccess property to a value of “Source” or “Target” for instances of StorageReplicationCapabilities that support remote replication. This has two purposes:

- The client can determine the correct service access point for invocation of remote replication methods.
- The method provider can determine if it has the cascading role or the leaf role.

A cascading copy services provider supports stitching with leaf elements of the following types:

- ComputerSystem representing peer system instances.
- ProtocolEndpoint representing ports/paths assigned to peer-to-peer connections.
- StorageVolume representing source and target elements in a remote mirror pair.

Values of key properties, durable name properties and Correlatable ID properties are copied from the real elements to leaf elements at the time leaf elements are instantiated. A cascading provider shall ensure that state/status properties return the same value for leaf elements and corresponding real elements. Associations required by the subprofile for two real elements are also required between real and leaf elements. Three extrinsic methods of the subprofile have cascading privileges:

- AttachOrModifyReplica
- CreateOrModifyReplicationPipe
- ModifySynchronization

Leaf elements are created and deleted as a side effect when these three methods are invoked. These three methods are reflective. The client invokes the method to the cascading provider which, in turn, invokes the method to the leaf provider. If an error is detected by the leaf provider, the cascading provider shall reflect an instance of CIM_Error to the client when appropriate. Other extrinsic methods of the subprofile do not cascade.

The subprofile supports a single topology with two SMI-S servers providing service access to two peer systems. This topology allows both uni-directional and bi-directional connections between peers. The cascading relationship is identified with a dependency association between instances of ComputerSystem representing the cascading and leaf peers. Both SMI-S servers populate their CIMOM repositories with all of the cascading and leaf elements representing a connection or a remote mirror pair.

Cascading imposes the following limitations on a copy services provider:

1) If the provider supports job control for extrinsic methods, jobs shall only be created by a method provider operating in the cascading role.

2) Indications related to leaf elements shall only be generated by a provider operating in the cascading role for these elements.

EXPERIMENTAL

8.2.8.12.4 Supported Subprofiles and Packages

The Block Services Package 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 a two SMI-S server topology.

Copy services defines instance indications and alert indications using required and optional properties described in the Indications Subprofile.

8.2.8.12.5 Methods of this 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 1058 summarizes the extrinsic methods for replica creation and management.

Table 1058: Extrinsic Methods of ReplicationServices Subprofile

Method	Described in
ModifySynchronization()	Table 1059: “ModifySynchronization”
CreateReplica()	Table 1060: “CreateReplica Method”
AttachOrModifyReplica()	Table 1061: “AttachOrModifyReplica Method”
CreateReplicationBuffer()	Table 1062: “CreateReplicationBuffer Method”
CreateOrModifyReplicationPipe()	Table 1063: “CreateOrModifyReplicationPipe Method”

Table 1059: ModifySynchronization

Method: ModifySynchronization			
Errors: DRM24, MP2, DRM25			
Parameters:			
Qualifiers	Name	Type	Description/Values
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.

Table 1059: ModifySynchronization

Method: ModifySynchronization			
Errors: DRM24, MP2, DRM25			
Parameters:			
Qualifiers	Name	Type	Description/Values
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 8.2.8.12.1.2 "Instrumentation Requirements". 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.

Table 1060: 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	
IN, REQ	CopyType	uint16	Copy type created: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc

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.
- All CopyType values may be supported.

If TargetSettingGoal is not supplied by the client, the provider generates a default StorageSetting element for the replica. If TargetPool is not supplied by the client, the provider selects a pool to contain the created replica element. If “job started' is returned, a Target Element reference may or may not be returned by the provider. 8.2.8.12.1.2 "Instrumentation Requirements" explains when a reference to the new replica element is available to the client.

EXPERIMENTAL

Table 1061: AttachOrModifyReplica Method

Method: AttachOrModifyReplica			
Errors: DRM25, DRM26, DRM27, MP5, MP7			
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
IN, EmInst	Goal	string	Setting element as an embed-ded instance.
IN	ReplicationPipe	NetworkPipe REF	

Method: AttachReplica			
Errors: DRM25, DRM26, DRM27, MP5, MP7			
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 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.

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.

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 1062: 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 replication NetworkPipe.
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 NetworkPipe. 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 buffer element is created with a BasedOn association to the extent and consumes the full extent. 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 1063: 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, EmInst	Goal	string	Setting element as an embed-ded instance. Reserved for future use.
IN, OUT	ReplicationPipe	NetworkPipe REF	

Method notes:

Creates a NetworkPipe 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 NetworkPipe 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.

EXPERIMENTAL

8.2.8.12.6 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 CopyType/replica type combination supported by the provider. Types “Sync” and “Async” are used to manage both local and remote mirrors. Type “UnSyncAssocFull” is used to manage full size snapshots and type “UnSyncAssocDelta” is used to manage delta snapshots. Type “UnSyncUnAssoc” is used to create a clone that becomes an independent storage element when finished. 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

Instances of StorageReplicationCapabilities for a specific CopyType value may indicate support for both local and remote replication. The value lists for SupportedSynchronousActions[] and SupportedAsynchronousActions[] should include multiple values indicating all of the local and remote replication capabilities that are supported. The client should understand that many of the properties in the capabilities instances return value lists indicating multiple capabilities.

The PersistentReplicasSupported property in each instance of StorageReplicationCapabilities is set to “true” if the client can manage replicas as elements that persist across system reset events and power off events.

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

8.2.8.12.6.1 Managing Peer-to-peer Connections

Remote replication is supported if CopyType “Sync” or “Async” is supported and any of the remote replication operations are in the list of supported actions for either of these CopyType values. 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” or both support “Async”.
- There shall be a SupportedSynchronousActions[] and SupportedAsynchronousActions[] value match. Both shall support the same set of remote 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 --> NetworkPipe --> 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 NetworkPipe 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 NetworkPipe 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 173: "Peer-to-Peer Connection".

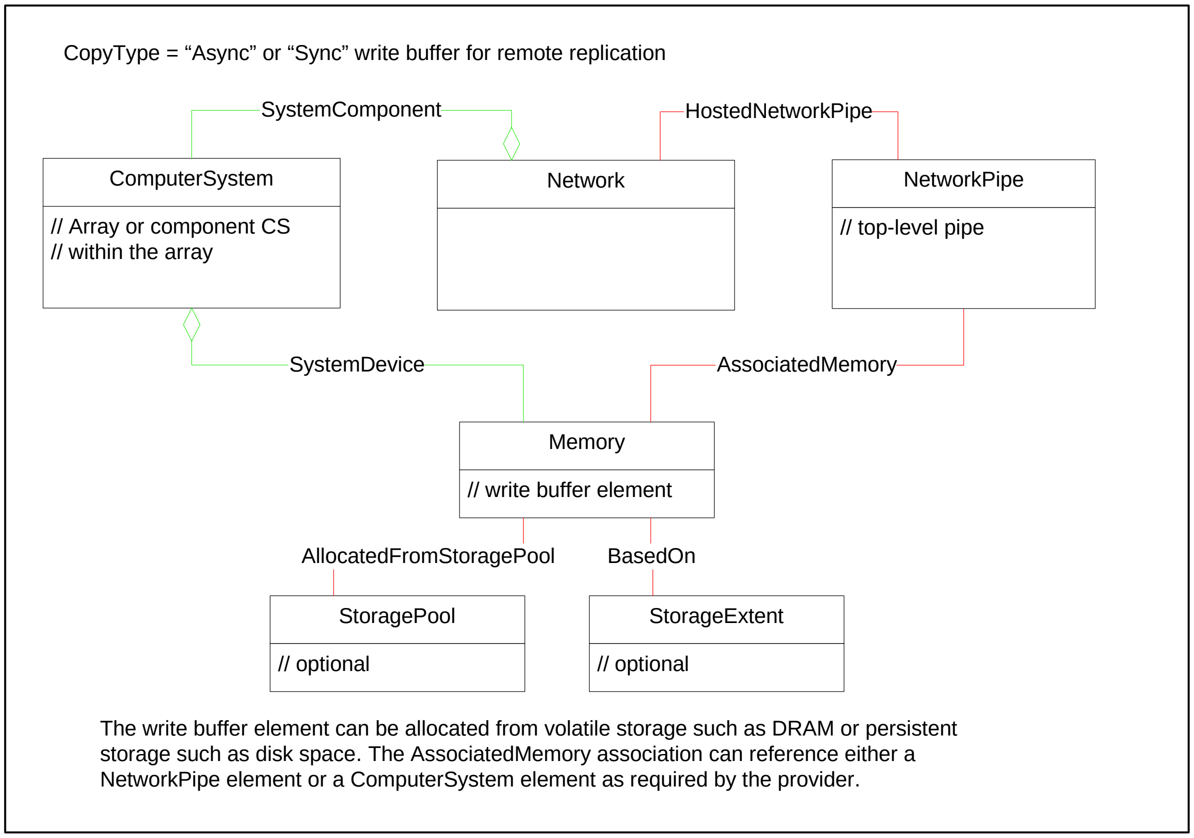
Step 4: A write ahead buffer may be required by one or both peers as indicated by the RemoteBufferSupported property in StorageReplicationCapabilities. Four properties in StorageReplicationCapabilities indicate how to manage remote buffers:

- 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 NetworkPipe element.
- RemoteBufferElementType indicates if the client supplies a reference to a concrete extent, a pool or neither as the container element for the buffer.

If the client supplies a concrete extent, 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.

Figure 179: Remote Replication Buffer


CopyType = “Async” or “Sync” write buffer for remote replication
SystemComponent
HostedNetworkPipe
ComputerSystem
// Array or component CS
// within the array
Network
NetworkPipe
// top-level pipe
SystemDevice
AssociatedMemory
Memory
// write buffer element
AllocatedFromStoragePool
BasedOn
StoragePool
// optional
StorageExtent
// optional
The write buffer element can be allocated from volatile storage such as DRAM or persistent storage such as disk space. The AssociatedMemory association can reference either a NetworkPipe element or a ComputerSystem element as required by the provider.

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 be invoking

DeleteInstance for a top-level NetworkPipe. The entire NetworkPipe composition is deleted along with corresponding associations. Deletion fails if any replica pairs remain associated with the connection.

EXPERIMENTAL

8.2.8.12.6.2 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 SupportedSpecializedElements[] property in StorageReplicationCapabilities indicates which values of IntendedUsage 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 GetAvailableExtents, 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.

8.2.8.12.6.3 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 target candidates to a specialized set of elements if the IntendedUsage property of StorageSetting is supported. 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 IntendedUsage to one of the values “local mirror”, “remote mirror”, “delta snapshot” or “full snapshot”. Set other replication setting property values as desired. Refer to the “Creating and Managing Snapshots” section in 8.2.8.12.6 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. If the setting element associated with the volume has the necessary IntendedUsage value and the volume is not presently a replica target (no existing StorageSynchronized associations), the client can screen and possibly select the volume as a new replica target. Note: if the provider does not support element specialization for replicas, there is no other way presently defined in the subprofile for screening existing volumes as candidates. If the target is to become a remote mirror, the selected pool shall be hosted on the peer system containing the target in this step and subsequent steps.
4) If a candidate becomes a delta replica and the provider supports element modification, the client can change the values of the delta reservation properties before invoking AttachOrModifyReplica. The client should ensure that all of the goal properties are first set to the same values as the values in the existing StorageSetting element associated to the candidate volume. Next, modify only the delta reservation properties to the values required by the client. Pass the completed instance as the Goal parameter when AttachOrModifyReplica is invoked.

5) 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 extents. As in step 2, set IntendedUsage and all of the other replication setting properties to the desired 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 shall be zero when the element is created.

EXPERIMENTAL

8.2.8.12.6.4 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
- “Shared” – a single shared pool is required for all delta replicas. If the pool already exists, it is associated to StorageConfigurationService with a ReplicaPoolForStorage association.
- “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.

The client may create the pool from another pool or from a set of extents as allowed by the provider. The StorageConfigurationCapabilities.SupportedStoragePoolFeatures[] property indicates the options for pool creation and modification. If the provider supports warning thresholds and space limits, the recommended approach is to create the pool from small extents. This allows the pool size to be increased by adding extents when a pool space warning is indicated. A provider that supports warning thresholds is also likely to support pool modification so that the pool size can be increased.

Calculate a size value for the pool. Select a candidate container pool and create a goal setting instance. Set IntendedUsage to “Delta Pool”. If the pool is created within a pool, invoke GetSupportedSizes or GetSupportedSizeRange to verify that a pool of the required size can be created.If the pool is created from extents, select a set of candidate extents from the candidate pool using the GetAvailableExtents method. If a provider supports the IntendedUsage property, all candidates should have the same IntendedUsage value. Finally, invoke CreateOrModifyStoragePool to create the pool. Refer to 8.2.8.10, "Block Services Package" for recipes that show this sequence of operations. If new component extents shall be created, set IntendedUsage to “Delta Pool Component” and create a set of extents with the required size. Refer to the Block Services recipes for element creation and set the created element type to “Storage Extent”. The pool size can be increased following a pool alert indication. Use the block services pool modification recipe(s) supported by the provider.

The delta replica pool is automatically associated to the appropriate managed element by the provider. A ReplicaPoolForStorage association to the StorageConfigurationService is created during the CreateOrModifyStoragePool operation. If the pool is “exclusive”, the association antecedent is modified to reference the source element during the first CreateReplica operation that refers to the pool. The ReplicaPoolForStorage association only identifies the association of the specialized pool to elements that may consume space in the pool. The AllocatedFromStoragePool association is used to manage and monitor space consumption by individual snapshot elements. If warning thresholds are supported, the client may invoke ModifyInstance to modify the value of StoragePool.LowSpaceWarningThreshold.

The provider may optionally provision the special pool with a set of virtual devices before returning completion status to the provider. These virtual devices are subsequently be used as AttachOrModifyReplica target elements. This allows the provider to maintain a higher degree of control over replica properties and the maximum number of replicas. 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 should not depend on the capacity relationship formulas specified in Block Services, Extent Mapping and Extent Conservation. The standard capacity relationship is:

```
TotalManagedSpace = RemainingManagedSpace + SUM(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. Additionally, if a snapshot service provider allows multiple snapshots to share a consumed block, it is difficult for a client to predict the space consumption rate for the pool. 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. The client should delegate much of the capacity management role to the provider using the following techniques:

Client: set LowSpaceWarningThreshold for the pool if the provider supports warning thresholds.
Provider: generate an alert indication at the threshold point.

Client: set DeltaReservationMin for a new snapshot if provider supports space reservation.
Provider: return “Failed” to method invocation if insufficient space to create the snapshot.

The provider is responsible for maintaining accurate values of RemainingManagedSpace and SpaceConsumed when multiple snapshots share a consumed block.

Extent mapping and extent conservation are not supported for elements created in a specialized delta replica pool.

EXPERIMENTAL

8.2.8.12.6.5 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 and is equivalent to a snapshot. 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: locate a candidate pool eligible to contain a new mirror or locate an element in the pool eligible to be a replica target. 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.

Step 2: 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 IntendedUsage to either “Local mirror” or “Remote mirror”.

Step 3: screen the candidate pool or the elements contained in the pool. The client shall provide a replica size value for the screening operation. Normally, this is the same size value as the source element. The generated setting created in step 2 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” in 8.2.8.12.6. Select a returned volume based on best fit or some other appropriate filter. Invoke GetSupportedSizes or GetSupportedSizeRange if CreateReplica is used. Proceed to step 4 if an eligible pool or extent 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 extent. Note: a client may elect to bypass screening and require a user to manually select a candidate pool or target element.

Step 4: 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 8.2.8.12.1.2 "Instrumentation Requirements". 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 AttachRemoteReplica Refer to the Instrumentation Requirements section under 8.2.8.12.1.6 "Provider Configurations for Remote Replication".

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.

8.2.8.12.6.6 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

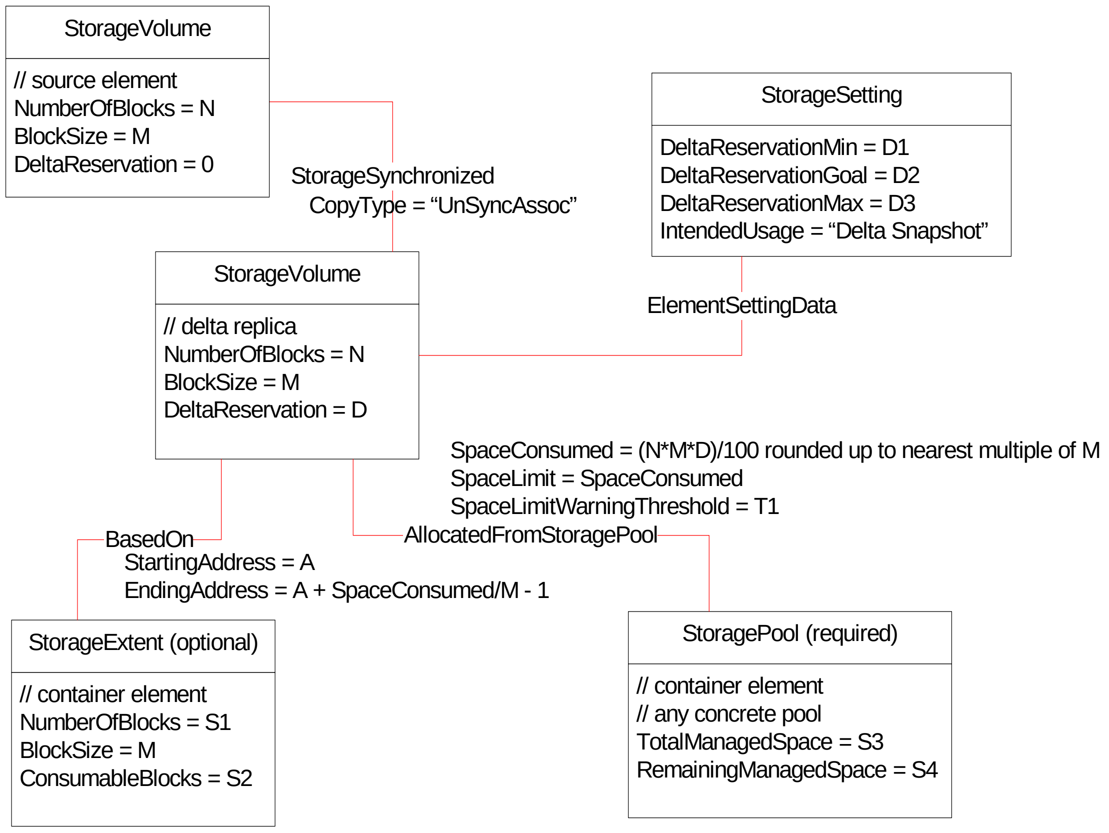
8.2.8.12.6.7 Creating and Managing Snapshots

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. Separate instances of StorageReplicationCapabilities are used to manage full size snapshots and delta snapshots:

- Full size: SupportedSynchronizationType = “UnSyncAssoc-Full”
- Delta: SupportedSynchronizationType = “UnSyncAssoc-Delta”

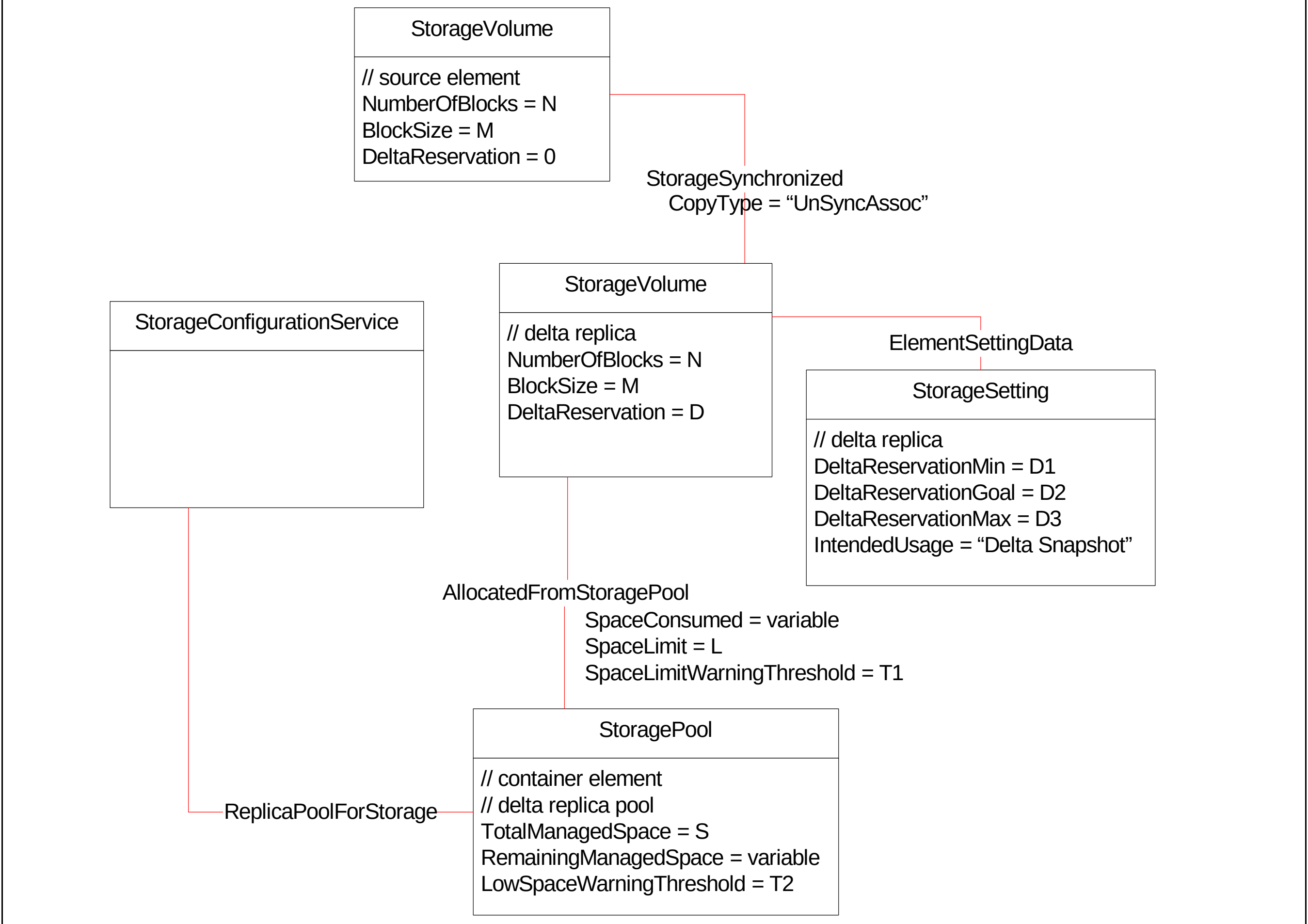
Snapshot providers may deploy either a fixed space consumption model or a variable space consumption model for snapshot replicas. A full size replica always uses a fixed space consumption model. A delta replica may use either a fixed or a variable model. Replica elements based on the variable model shall be created in special pools for delta replicas. A provider indicates support for special pools by including the value “Delta Replica Pool” in SupportedSpecializedElements[]. 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 180: "Fixed Space Consumption" and Figure 181: "Variable Space Consumption" show the fixed and variable space consumption models for delta snapshots:

Figure 180: Fixed Space Consumption



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 not used for full size snapshots. SpaceLimit and SpaceLimitWarningThreshold are not used for snapshots with fixed space consumption.

Figure 181: Variable Space Consumption



The instances of StorageReplicationCapabilities for “UnSyncAssoc-Delta” and “UnSyncAssoc-Full” may use the patterns detailed in Table 1064: “Patterns Supported for StorageReplicationCapabilities”.

Table 1064: Patterns Supported for StorageReplicationCapabilities

SupportedSynchronizationType	Supported…Actions[n]	DeltaReplicaPoolAccess	Space Consumption
UnSyncAssoc-Delta	“Local Replica Attachment”	Any pool or extent	Fixed
UnSyncAssoc-Delta	“Local Replica Creation”	Any pool or extent	Fixed
UnSyncAssoc-Delta	“Local Replica Attachment”	Shared or Exclusive	Variable
UnSyncAssoc-Delta	“Local Replica Creation”	Shared or Exclusive	Variable
UnSyncAssoc-Full	“Local Replica Attachment”	n/a	Fixed
UnSyncAssoc-Full	“Local 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 created as a separate step before the client begins creating delta replicas. 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 creating a special pool for delta replicas is described in "Creating and Managing Pools for Delta Replicas" in 8.2.8.12.5. 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 extents 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 IntendedUsage to either “Full snapshot” or “Delta snapshot”.

If the provider indicates SpaceReservationSupported “true”, 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:

DeltaReservationMin <= DeltaReservationGoal <= 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. The delta reservation properties are always used when CreateReplica is invoked. The properties are only used by AttachOrModifyReplica when the input target element is a virtual volume with a SpaceConsumed value of zero. 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. If the provider supports a variable space consumption model, DeltaReservationGoal should be set to a value that is the best estimate of space consumption as a function of the source element volatility and the replica retention period. All three of the delta reservation properties are set to the same value if the client wants to guarantee a specific amount of space is reserved. The provider determines the actual amount of space reserved within the range requested by the client. If the provider cannot satisfy the minimum reservation request, the client receives an error response indicating resource limit exceeded. If the request is satisfied, the provider sets SpaceConsumed to reflect the initial amount of space reserved for the snapshot. The DeltaReservation property for the snapshot storage element is set to the ratio of snapshot SpaceConsumed to source SpaceConsumed.

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 extents 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” in 8.2.8.12.5 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 extent. Be sure to use a Size value of zero 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 values from the goal parameter apply to the new replica. If a delta replica is created, the NumberOfBlocks and BlockSize values of the source element are assigned to the target. If space limits apply, the initial value of AllocatedFromStoragePool.SpaceLimit is set to a default value determined by the provider and SpaceConsumed may not exceed Spacelimit. SpaceLimitWarningThreshold is set to an initial value of SpaceLimitWarningThresholdDefault.

The properties listed in Table 1065: “Space Consumption Properties” are used to monitor and manage space consumption for delta replicas using a variable space consumption pattern.

Table 1065: 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
StorageExtent.DeltaReservation: valid for target elements. Value 0 to 100 set by CreateReplica and AttachOrModifyReplica method providers.	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
AllocatedFromStoragePool.SpaceLimitWarningThreshold: valid if provider supports replica warning thresholds.	constant	yes
AllocatedFromStoragePool.SpaceLimit: valid if provider supports space limits for replicas in special delta replica pools.	constant	yes
StorageSetting.DeltaReservationMin: Minimum space reserved when space reservation is supported.	constant	yes (goal)
StorageSetting.DeltaReservationMax: Maximum space reserved when space reservation is supported.	constant	yes (goal)
StorageSetting.DeltaReservationGoal: Client goal for space reserved when space reservation is supported.	constant	yes (goal)

The properties listed in Table 1066: “Space Consumption Properties, Fixed Pattern” are used to monitor and manage space consumption for delta replicas using a fixed space consumption pattern.

Table 1066: 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 and AttachOrModifyReplica method providers.	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.	constant	yes (goal)
StorageSetting.DeltaReservationMax: Value is % of source element size that is maximum fixed size. Used only with CreateReplica method.	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.	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 ModifyInstance intrinsic method can be invoked to manage space limits for delta snapshots when SpaceLimitSupported is “true”. The client should set the SpaceLimitWarningThreshold and SpaceLimit properties to the desired values in the AllocatedFromStoragePool association to the delta replica. The client should also listen for all alert indications defined for space limit management,

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.

A group of delta replicas may be incrementally dependent from the oldest to the newest. The newest element in a group is always independent. If the IncrementalDeltasSupported property has a value of “true”, then “Resync” and “Detach” operations should only be invoked for the oldest element in a group.

8.2.8.12.6.8 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 full size snapshots or delta snapshots without space limits. 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

8.2.8.12.6.9 Recipes

The Copy Services recipes show usage of all methods used to manage local and remote replication. There is at least one invocation of each method. The ModifySynchronization method has many variations. Preceding discussion in the client considerations section explains usage of all ModifySynchronization operations not shown in recipes. The set of recipes references recipes in block services but does not duplicate recipe logic from this or other subprofiles. The recipes assume that the client subscribes to all provider-supplied filters for instance creation, modification and deletion indications. The recipes assume that the client does all setup for job control such as waiting for job completion and checking job completion results.

Recipes supporting remote replication require the client to discover the SAP for both source and target hosting systems such as arrays. Extrinsic method invocation is directed to the correct SAP based on the value of the RemoteReplicationServicePointAccess property in a StorageReplicationCapabilities instance.

Recipes that create replicas all follow three basic patterns that cover most known providers:

- Locate an existing element to attach as the replica target.
- Create a new replica element in a target pool.
- Create a new storage element then attach the element as the replica target.

Much of the recipe logic may seem similar to Block Services recipes. Copy Services recipes have two fundamental differences:

- The IntendedUsage property in a StorageSetting goal or instance is used to manage special purpose pools, component extents or target replica elements.
- Component extents with special purposes may be created by extrinsic method invocation.

8.2.8.12.6.9.1 Establish Peer-to-Peer Connection

```
// NAME: Establish Peer-to-Peer Connection
// FILE: CopyServicesSP_Recipe1of7
//
// 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 and
// 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 NetworkPipe composition with one top-level
// NetworkPipe element and one or more second-level NetworkPipe elements
// corresponding to the number of supplied endpoint pairs.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// Provider supports remote replication using CopyType “Async” or “Sync”
// $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
// #ConnectionName is a string value assigned to the ElementName
// property of the new top-level pipe
```

```
//
// OUTPUT: $Pipe is a reference to the new top-level NetworkPipe.
//

// Setup control variables for the recipe.
$L[] = Associators(
     $SourceSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$SourceSCS = $L[0]
$L[] = Associators(
     $TargetSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$TargetSCS = $L[0]
$SourceSRC, $TargetSRC = null
$SRC[] = Associators(
         // locate StorageReplicationCapabilities for source array
     $SourceSCS->,
     “CIM_ElementCapabilities”,
     “CIM_StorageReplicationCapabilities”,
     null, null, false, false, null)
for #i in $SRC[]
{
     if (($SRC[#i].SupportedSynchronizationType == 2) ||
                                    ($SRC[#i].SupportedSynchronizationType == 3))
     { // found “Async” or “Sync”
         if (contains(9, $SRC[#i].SupportedSynchronousActions[]))
         { // found “NetworkPipe Creation”
             $SourceSRC = $SRC[#i]
             break
         }
     }
}
$SRC[] = Associators(
         // locate StorageReplicationCapabilities for target array
     $TargetSCS->,
     “CIM_ElementCapabilities”,
     “CIM_StorageReplicationCapabilities”,
     null, null, false, false, null)
for #i in $SRC[]
{
     if (($SRC[#i].SupportedSynchronizationType == 2) ||
                                    ($SRC[#i].SupportedSynchronizationType == 3))
     { // found “Async” or “Sync”
```

```
        if (contains(9, $SRC[#i].SupportedSynchronousActions[]))
        { // found “NetworkPipe Creation”
            $TargetSRC = $SRC[#i]
            break
        }
    }
}
#protocol = $SourceSRC.PeerConnectionProtocol // all endpoints must match

// Verify that the peer systems are compatible for remote replication
if (($SourceSRC == null) || ($TargetSRC == null)
{
     <error: peer connections not supported>
}
if (($TargetSRC.PeerConnectionProtocol != #protocol) ||
    ($SourceSRC.BidirectionalConnectionsSupported !=
         $TargetSRC.BidirectionalConnectionsSupported) ||
    ($SourceSRC.RemoteReplicationServicePointAccess !=
         $TargetSRC.RemoteReplicationServicePointAccess))
{
     <error: peer systems not compatible>
}

// All verification checks passed. Invoke CreateOrModifyReplicationPipe
// at the right SAP to establish the connection.
%InArguments[“PipeElementName”] = #ConnectionName
%InArguments[“SourceSystem”] = $SourceSystem->
%InArguments[“TargetSystem”] = $TargetSystem->
%InArguments[“SourceEndpoint”] = $SourceEnd->[]
%InArguments[“TargetEndpoint”] = $TargetEnd->[]
%InArguments[“Goal”] = null
if ($SourceSRC.RemoteReplicationServicePointAccess != 4)
{ // invoke to source system service
     #r = InvokeMethod(
         $SourceSCS->,
         “CreateOrModifyReplicationPipe”,
         %InArguments,
         %OutArguments)
} else
{ // invoke to target system service
     #r = InvokeMethod(
         $TargetSCS->,
         “CreateOrModifyReplicationPipe”,
         %InArguments,
         %OutArguments)
}
if (#r != 0)
```

```
{
     <error: failed to establish connection>
}

// Reference to the new top-level pipe.
$Pipe-> = %OutArguments[“ReplicationPipe”]

// Recipe complete. Connection is established 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.
```

8.2.8.12.6.9.2 Create Local Mirror Or Clone

```
// NAME: Create Local Mirror Or Clone
// FILE: CopyServicesSP_Recipe2of7
//
// DESCRIPTION: Create a new, local mirror replica element or a clone
// element. Client indicates CopyType “Sync”, “Async” or “UnSyncUnAssoc”.
// The replica element or clone element is fully copied from the source
// element and is hosted on the same array as the source element.
// The recipe supports both CreateReplica and AttachOrModifyReplica
// extrinsic methods. If the attach method is supported, search for a
// target element to attach. If no target element is found, search for
// a target pool, create an element in the pool and attach the new
// element. If the create method is supported, search for a target pool
// and create the new replica in the pool.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SCC is the instance of CIM_StorageConfigurationCapabilities
// controlling the recipe.
// $SRC is the instance of CIM_StorageReplicationCapabilities
// controlling the recipe.
// $SRC is for SupportedSynchronizationType “Sync”, “Async” or “UnSyncUnAssoc”.
// $SCS is the instance of CIM_StorageConfigurationService controlling
// the recipe.
// $System is the instance of CIM_ComputerSystem identifying the array.
// $SV is the instance of CIM_StorageVolume identifying the
// replica source.
//

//
// MakeGoalParameter subroutine. Creates a modifiable goal parameter and
// sets the IntendedUsage value. Caller must delete the goal parameter
// when no longer needed. Caller can pass $ModSetting as EmbeddedInstance
// or can pass REF to repository copy.
//
```

```
sub uint8 MakeGoalParameter (IN $Pool, IN #IntendedUsage, OUT $ModSetting)
{
$CL[] = Associators(
     $Pool->,
     “CIM_ElementCapabilities”,
     “CIM_StorageCapabilities”,
     null, null, false, false, null)
$Capabilities = $CL[0]
%InArguments[“SettingType”] = 3 // Goal
#g = InvokeMethod(
     $Capabilities->,
     “CreateSetting”,
     %InArguments,
     %OutArguments)
if (#g != 0)
{
     < error: cannot create setting element>
}
$GenSetting-> = %OutArguments[“NewSetting”]
$ModSetting = GetInstance( // make local client copy
     $GenSetting->,
     false, false, false, null)
$ModSetting.IntendedUsage = #IntendedUsage
$ModSetting-> = $ModSetting.getObjectPath()
ModifyInstance( // update in CIMOM repository
     $ModSetting->,
     false, null)
} // end of MakeGoalParameter

//
// DeleteGoal subroutine corresponding to above MakeGoalParameter.
//
sub uint8 DeleteGoal (IN $ModSetting)
{
$ModSetting-> = $ModSetting.getObjectPath()
DeleteInstance ($ModSetting->)
}

//
// SizeCheck subroutine. Return 0 if element of requested size and type
// can be created in the pool. Return 1 if size not supported. Searches
// for size or range greater than or equal to the requested size. A
// client could also be designed to search for an exact match.
//
sub uint8 SizeCheck
            (IN $Pool, IN #ElementType, IN #ElementSize, IN $ModSetting)
{
```

```
%InArguments[“ElementType”] = #ElementType
%InArguments[“Goal”] = $ModSetting.getObjectPath()
#c = InvokeMethod(
     $Pool->,
     “GetSupportedSizes”,
     %InArguments,
     %OutArguments)
if (#c == 0)
{ // method supported -- check size
     #sizes[] = %OutArguments[“Sizes”]
     for #k in #sizes[]
     {
        if (#ElementSize <= #sizes[#k])
        {
           return 0
        }
     }
} else // try next method
{
     #c = InvokeMethod(
        $Pool->,
        “GetSupportedSizeRange”,
        %InArguments,
        %OutArguments)
     if (#c != 0)
     {
        return 1
     }
     #max = %OutArguments[“MaximumVolumeSize”]
     #min = %OutArguments[“MinimumVolumeSize”]
     if ((#ElementSize >= #min) && (#ElementSize <= #max))
     {
        return 0
     }
}
return 1
} // end of SizeCheck

//
// FindTargetElementOrPool subroutine. Searches the selected pool for a
// target element of the correct size eligible for the intended usage.
// Sets $TV if target element is found and returns 0. If no target element
// is found but selected pool allows the intended usage, returns 1.
// Otherwise, returns 2. If 1 returned, $ModSetting retained for use
// by caller.
//
sub uint8 FindTargetElementOrPool (IN $Pool, IN #IntendedUsage, IN #Size,
                      OUT $TV, OUT $ModSetting)
{
$Vols[] = Associators(
     $Pool->,
     “CIM_AllocatedFromStoragePool”,
     “CIM_StorageVolume”,
     null, null, false, false, null)
for #i in $Vols[]
{
     // Volume is a candidate if not a replica source or target
     // and the IntendedUsage value is a match. Client sets
      // IntendedUsage to zero if provider does not support
      // element specialization.
     $Refs[] = ReferenceNames(
        $Vols[#i].getObjectPath(),
        “CIM_StorageSynchronized”,
        null)
     if ($Refs[].size() == 0) // element is not already a replica
     {
        if (($Vols[#i].NumberOfBlocks * $Vols[#i].BlockSize) >= #Size)
        {
           $SL[] = Associators(
              $Vol[#i].getObjectPath(),
              “CIM_ElementSettingData”,
              “CIM_StorageSetting”,
              null, null, false, false,
              “IntendedUsage”)
           $VolSetting = $SL[0]
           if ($VolSetting.IntendedUsage == #IntendedUsage)
           {
              $TV = $Vols[#i]
              return 0
           }
        }
     }
}

// Did not find target element. Search for a target pool.

// Create a setting element for the search and perhaps element creation.
#r = &MakeGoalParameter ($Pool, #IntendedUsage, $ModSetting)

// Search pool trying the first method.
#r = &SizeCheck ($Pool, 2, #Size, $ModSetting)
if (#r == 0)
{
     return 1 // size supported in pool
```

```
}

// Pool is not a candidate
#d = &DeleteGoal ($ModSetting)
return 2
} // *** end of FindTargetElementOrPool subroutine

// Main section of recipe

#CrEl, #AttRep, #CrRep = false
if ((contains(3, $SCC.SupportedStorageElementFeatures[])) &&
     (contains(6, $SCC.SupportedStorageElementFeatures[]) ||
      contains(7, $SCC.SupportedStorageElementFeatures[])))
{
     #CrEl = true // CreateOrModifyElementFromStoragePool supported?
}
if (contains(2, $SRC.SupportedSynchronousActions[] ||
     contains(2, $SRC.SupportedAsynchronousActions[])
{
     #CrRep = true // CreateReplica method supported
}
if (contains(6, $SRC.SupportedSynchronousActions[] ||
     contains(6, $SRC.SupportedAsynchronousActions[])
{
     #AttRep = true // AttachOrModifyReplica method supported
}

// Step 1: find a target element and/or a target pool for the local
// mirror. Request same usable size as source element
#Size = $SV.NumberOfBlocks * $SV.BlockSize
#IntendedUsage = 0 // “Not specialized” indicated for a clone or a
                   // provider not supporting specialized elements.
if (($SRC.SupportedSynchronizationType != 6) && // “UnSyncUnAssoc”
    (contains(5, $SRC.SupportedSpecializedElements[])) // “Local Mirror”
{
     #IntendedUsage = 5 // “Local mirror” indicated for a local mirror
}
$PoolList[] = Associators(
     $System->,
     “CIM_HostedStoragePool”,
     “CIM_StoragePool”,
     null, null, false, false, null)
for #ii in $PoolList[]
{
     $Pool = $PoolList[#ii]
     #rr = &FindTargetElementOrPool
                       ($Pool, #IntendedUsage, #Size, $TV, $ModSetting)
```

```
          if (#rr != 2)
          {
             break
          }
     }
     if (#rr == 2)
     {
          <error: cannot create a local mirror/no target element or pool>
     }

     // Step 2: if $TV returned as target and #AttRep is true, invoke
     // the AttachOrModifyReplica method.
     if ((#rr == 0) && #AttRep)
     {
          %InArguments[“SourceElement”] = $SV->
          %InArguments[“TargetElement”] = $TV->
          %InArguments[“CopyType”] = $SRC.SupportedSynchronizationType
          %InArguments[“ReplicationPipe”] = null // for local mirror or clone
          #r = InvokeMethod(
             $SCS->,
             “AttachOrModifyReplica”,
             %InArguments,
             %OutArguments)
          if (#r != 0 && #r != 4096)
          {
             <error: attach 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 (!contains(2, $Job.OperationalStatus[])
             {
                <error: attach job failed, stop and examine CIM_Error>
             }
          }
          if ($SRC.SupportedSynchronizationType != 6) // not a clone
          { // locate new StorageSynchronized association for a mirror
             $SL[] = References(
                $TV->,
                “CIM_StorageSynchronized”,
                “SyncedElement”,
                false, false, null)
             $SS = $SL[0]
```

```
            if ($SS.SyncState != $SRC.InitialReplicationState)
            {
                <wait for $SS.SyncState instance mod indication>
                $SS = GetInstance( // refresh the SyncState value
                    $SS->,
                    false, false, false, null)
            }
        }
        // stop recipe if step 2 was executed.
    }

    // Step 3: if a target pool was returned and #CrRep is true, invoke
    // the CreateReplica method.
    if ((#rr == 1) && #CrRep)
    {
        %InArguments[“SourceElement”] = $SV->
        %InArguments[“CopyType”] = $SRC.SupportedSynchronizationType
        %InArguments[“TargetSettingGoal”] = $ModSetting.getObjectPath()
        %InArguments[“TargetPool”] = $Pool->
        #r = InvokeMethod(
            $SCS->,
            “CreateReplica”,
            %InArguments,
            %OutArguments)
        #d = &DeleteGoal ($ModSetting)
        if (#r != 0 && #r != 4096)
        {
            <error: create 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 (!contains(2, $Job.OperationalStatus[])
            {
                <error: create job failed, stop and examine CIM_Error>
            }
            $TL[] = Associators(
                $Job->,
                “CIM_AffectedJobElement”,
                “CIM_StorageVolume”,
                null, null, false, false, null)
            $TV = $TL[0]
        } else
```

```
          {
             $TV-> = %OutArguments[“TargetElement”]
             $TV = GetInstance(
                $TV->,
                false, false, false, null)
          }
          if ($SRC.SupportedSynchronizationType != 6) // not a clone
          { // locate new StorageSynchronized association for a mirror
             $SL[] = References(
                $TV->,
                “CIM_StorageSynchronized”,
                “SyncedElement”,
                false, false, null)
             $SS = $SL[0]
             if ($SS.SyncState != $SRC.InitialReplicationState)
             {
                <wait for $SS.SyncState instance mod indication>
                $SS = GetInstance( // refresh the SyncState value
                   $SS->,
                   false, false, false, null)
             }
          }
          // stop recipe if step 3 was executed.
     }

     // Step 4: if a target pool was returned and #AttRep is true, invoke
     // the CreateOrModifyElementFromStoragePool method followed by the
     // AttachOrModifyReplica method.
     if ((#rr == 1) && #AttRep && #CrEl)
     {
          %InArguments[“ElementType”] = 2 // StorageVolume
          %InArguments[“Goal”] = $ModSetting
          %InArguments[“Size”] = #Size
          %InArguments[“InPool”] = $Pool->
          %InArguments[“TheElement”] = null
          #r = InvokeMethod(
             $SCS->,
             “CreateOrModifyElementFromStoragePool”,
             %InArguments,
             %OutArguments)
          #d = &DeleteGoal ($ModSetting)
          if (#r != 0 && #r != 4096)
          {
             <error: element creation 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 (!contains(2, $Job.OperationalStatus[])
    {
        <error: creation job failed, stop and examine CIM_Error>
    }
    $TL[] = Associators(
        $Job->,
        “CIM_AffectedJobElement”,
        “CIM_StorageVolume”,
        null, null, false, false, null)
    $TV = $TL[0]
} else
{
    $TV-> = %OutArguments[“TheElement”]
    $TV = GetInstance(
        $TV->,
        false, false, false, null)
}
%InArguments[“SourceElement”] = $SV->
%InArguments[“TargetElement”] = $TV->
%InArguments[“CopyType”] = $SRC.SupportedSynchronizationType
%InArguments[“ReplicationPipe”] = null // for local mirror or clone
#r = InvokeMethod(
    $SCS->,
    “AttachOrModifyReplica”,
    %InArguments,
    %OutArguments)
if (#r != 0 && #r != 4096)
{
    <error: attach 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 (!contains(2, $Job.OperationalStatus[])
    {
        <error: attach job failed, stop and examine CIM_Error>
    }
}
```

```
          if ($SRC.SupportedSynchronizationType != 6) // not a clone
          { // locate new StorageSynchronized association for a mirror
             $SL[] = References(
                $TV->,
                “CIM_StorageSynchronized”,
                “SyncedElement”,
                false, false, null)
             $SS = $SL[0]
             if ($SS.SyncState != $SRC.InitialReplicationState)
             {
                <wait for $SS.SyncState instance mod indication>
                $SS = GetInstance( // refresh the SyncState value
                   $SS->,
                   false, false, false, null)
             }
          }
     } else
     {
          <error: cannot create a local mirror/cannot create a target element>
          #d = &DeleteGoal ($ModSetting)
     } // end of step 4.

     // End of recipe. If successful, $TV is an instance of the local mirror
     // or clone and $SS is an instance of the StorageSynchronized association
     // to the mirror.
```

8.2.8.12.6.9.3 Create Remote Mirror Or Clone

```
     // NAME: Create Remote Mirror Or Clone
     // FILE: CopyServicesSP_Recipe3of7
     //
     // DESCRIPTION: Create a new, remote mirror replica element or a clone
     // element. Client indicates CopyType “Sync”, “Async” or “UnSyncUnAssoc”.
     // The replica element or clone element is fully copied from the source
     // element. The recipe supports CreateReplica and AttachOrModifyReplica
     // extrinsic methods. If the attach method is supported, search for a
     // target element to attach. If no target element is found, search for a
     // target pool, create an element in the pool and attach the new element.
     // If the create method is supported, search for a target pool and
     // create the new replica in the pool. The replica source and replica
     // target are hosted target are hosted on different arrays.
     //
     // PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
     //
     // $SourceSystem is the instance of CIM_ComputerSystem for the source
     // array. $TargetSystem is the instance of CIM_ComputerSystem identifying
     // the target array.
```

```
// $SV is the instance of CIM_StorageVolume identifying the replica
// source hosted by $SourceSystem.
// $Pipe is an optional instance of CIM_NetworkPipe identifying a
// peer-to-peer connection between $SourceSystem and $TargetSystem.
// If supplied, $Pipe must reference a top-level pipe in a 2-level
// NetworkPipe composition.
// Client must create a replica buffer, if required, prior to
// creation of any remote replica pairs.
// #CopyType indicates “Sync”, “Async” or “UnSyncUnAssoc”.
//
// Calls the FindTargetElementOrPool subroutine in the “Create Local
// Mirror Or Clone” recipe.
//

if (#CopyType != 2 || #CopyType != 3 || #CopyType != 6)
{
     <error: invalid CopyType parameter for remote mirror creation>
}

// Locate instances of StorageReplicationCapabilities for source and
// target arrays corresponding to #CopyType
$SRS_S, $SRS_T = null
$SL[] = Associators (
     $SourceSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$SCS_S = $SL[0]
$SL[] = Associators (
     $SCS_S->,
     “CIM_ElementCapabilities”,
     “CIM_StorageReplicationCapabilities”,
     null, null, false, false, null)
for #i in $SL[]
{
     if ($SL[#i].SupportedSynchronizationType == #CopyType)
     {
        $SRS_S = $SL[#i]
        break
     }
}

$SL[] = Associators (
     $TargetSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
```

```
$SCS_T = $SL[0]
$SL[] = Associators (
     $SCS_T->,
     “CIM_ElementCapabilities”,
     “CIM_StorageReplicationCapabilities”,
     null, null, false, false, null)
for #i in $SL[]
{
     if ($SL[#i].SupportedSynchronizationType == #CopyType)
     {
        $SRS_T = $SL[#i]
        break
     }
}

if ($SRS_S == null || $SRS_T == null)
{
     <error: requested CopyType not supported>
}

// Verify that source and target arrays are compatible
if (($SRS_S.RemoteReplicationServicePointAccess !=
     $SRS_T.RemoteReplicationServicePointAccess) ||
    ($SRS_S.PeerConnectionProtocol !=
     $SRS_T.PeerConnectionProtocol))
{
     <error: source and target arrays not compatible>
}
if (($SRS_S.MaximumPeerConnections > 0) && ($Pipe == null))
{
     <error: connection required before creating remote mirrors>
}

// Use RemoteReplicationServicePointAccess to select method SAP.
if ($SRS_S.RemoteReplicationServicePointAccess == 4)
{
     $SCS = $SCS_T // invoke replication methods to target provider
     $SRS = $SRS_T
} else
{
     $SCS = $SCS_S // invoke to source array provider
     $SRS = $SRS_S
}

// Always reference StorageConfigurationCapabilities for target array
$SL[] = Associators (
     $SCS_T->,
     “CIM_ElementCapabilities”,
     “CIM_StorageConfigurationCapabilities”,
     null, null, false, false, null)
$SCC = $SL[0]

// Remaining steps are nearly identical for local & remote mirror creation
#CrEl, #AttRep, #CrRep = false
if ((contains(3, $SCC.SupportedStorageElementFeatures[])) &&
     (contains(6, $SCC.SupportedStorageElementFeatures[]) ||
      contains(7, $SCC.SupportedStorageElementFeatures[])))
{
     #CrEl = true // CreateOrModifyElementFromStoragePool supported
}
if (contains(3, $SRC.SupportedSynchronousActions[] ||
     contains(3, $SRC.SupportedAsynchronousActions[])
{
     #CrRep = true // CreateReplica method supported
}
if (contains(7, $SRC.SupportedSynchronousActions[] ||
     contains(7, $SRC.SupportedAsynchronousActions[])
{
     #AttRep = true // AttachOrModifyReplica method supported
}

// Step 1: find a target element and/or a target pool for the remote
// mirror or clone. Search on target array. Request same usable size as
// source element
#Size = $SV.NumberOfBlocks * $SV.BlockSize
#IntendedUsage = 0 // “Not specialized” indicated for a clone or a
                   // provider not supporting specialized elements.
if (($SRC.SupportedSynchronizationType != 6) && // “UnSyncUnAssoc”
    (contains(4, $SRC.SupportedSpecializedElements[])) // “Remote Mirror”
{
     #IntendedUsage = 4 // “Remote mirror” indicated for a remote mirror
}
$PoolList[] = Associators (
     $TargetSystem->,
     “CIM_HostedStoragePool”,
     “CIM_StoragePool”,
     null, null, false, false, null)
for #ii in $PoolList[]
{
     $Pool = $PoolList[#ii]
     #rr = &FindTargetElementOrPool
                 ($Pool, #IntendedUsage, #Size, $TV, $ModSetting)
     if (#rr != 2)
     {
        break
     }
}
if (#rr == 2)
{
     <error: cannot create a remote mirror/no target element or pool>
}

// Step 2: if $TV returned as target and #AttRep is true, invoke
// the AttachOrModifyReplica method.
if ((#rr == 0) && #AttRep)
{
     %InArguments[“SourceElement”] = $SV->
     %InArguments[“TargetElement”] = $TV->
     %InArguments[“CopyType”] = $SRC.SupportedSynchronizationType
     %InArguments[“ReplicationPipe”] = $Pipe->
     #r = InvokeMethod(
        $SCS->,
        “AttachOrModifyReplica”,
        %InArguments,
        %OutArguments)
     if (#r != 0 && #r != 4096)
     {
        <error: attach failed, stop recipe 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 (!contains(2, $Job.OperationalStatus[])
        {
            <error: attach job failed, stop and examine CIM_Error>
        }
     }
     if ($SRC.SupportedSynchronizationType != 6) // not a clone
     { // locate new StorageSynchronized association if mirror created
        $SL[] = References(
            $TV->,
            “CIM_StorageSynchronized”,
            “SyncedElement”,
            false, false, null)
        $SS = $SL[0]
        if ($SS.SyncState != $SRC.InitialReplicationState)
        {
```

```
           <wait for $SS.SyncState instance mod indication>
           $SS = GetInstance( // refresh the SyncState value
              $SS->,
              false, false, false, null)
        }
     }
     // stop recipe if step 2 was executed.
}

// Step 3: if a target pool was returned and #CrRep is true, invoke
// the CreateReplica method. This option for remote mirrors and clones
// requires either a single SMI server as the SAP for the source and
// target arrays or cascading support by the method provider.
if ((#rr == 1) && #CrRep)
{
     %InArguments["SourceElement"] = $SV->
     %InArguments["CopyType"] = $SRC.SupportedSynchronizationType
     %InArguments["TargetSettingGoal"] = $ModSetting.getObjectPath()
     %InArguments["TargetPool"] = $Pool->
     #r = InvokeMethod(
        $SCS->,
        "CreateReplica",
        %InArguments,
        %OutArguments)
     #d = &DeleteGoal ($ModSetting)
     if (#r != 0 && #r != 4096)
     {
        <error: create 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 (!contains(2, $Job.OperationalStatus[])
        {
           <error: create job failed, stop and examine CIM_Error>
        }
        $TL[] = Associators(
           $Job->,
           "CIM_AffectedJobElement",
           "CIM_StorageVolume",
           null, null, false, false, null)
        $TV = $TL[0]
     } else
     {
        $TV-> = %OutArguments[“TargetElement”]
        $TV = GetInstance(
                $STV->,
                false, false, false, null)
     }
     if ($SRC.SupportedSynchronizationType != 6) // not a clone
     { // locate new StorageSynchronized association if mirror created
        $SL[] = References(
           $TV->,
           “CIM_StorageSynchronized”,
           “SyncedElement”,
           false, false, null)
        $SS = $SL[0]
        if ($SS.SyncState != $SRC.InitialReplicationState)
        {
           <wait for $SS.SyncState instance mod indication>
           $SS = GetInstance( // refresh the SyncState value
              $SS->,
              false, false, false, null)
        }
     }
     // stop recipe if step 3 was executed.
}

// Step 4: if a target pool was returned and #AttRep is true, invoke
// the CreateOrModifyElementFromStoragePool method followed by the
// AttachOrModifyReplica method.
if ((#rr == 1) && #AttRep && #CrEl)
{
     %InArguments[“ElementType”] = 2 // StorageVolume
     %InArguments[“Goal”] = $ModSetting.getObjectPath()
     %InArguments[“Size”] = #Size
     %InArguments[“InPool”] = $Pool->
     %InArguments[“TheElement”] = null
     #r = InvokeMethod(
        $SCS_T->,
        “CreateOrModifyElementFromStoragePool”,
        %InArguments,
        %OutArguments)
     #d = &DeleteGoal ($ModSetting)
     if (#r != 0 && #r != 4096)
     {
        <error: element 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 (!contains(2, $Job.OperationalStatus[])
        {
            <error: creation job failed, stop and examine CIM_Error>
        }
        $TL[] = Associators(
            $Job->,
            “CIM_AffectedJobElement”,
            “CIM_StorageVolume”,
            null, null, false, false, null)
        $TV = $TL[0]
    } else
    {
        $TV-> = %OutArguments[“TheElement”]
        $TV = GetInstance(
                $STV->,
                false, false, false, null)
    }
    %InArguments[“SourceElement”] = $SV->
    %InArguments[“TargetElement”] = $TV->
    %InArguments[“CopyType”] = $SRC.SupportedSynchronizationType
    %InArguments[“ReplicationPipe”] = $Pipe->
    #r = InvokeMethod(
        $SCS->,
        “AttachOrModifyReplica”,
        %InArguments,
        %OutArguments)
    if (#r != 0 && #r != 4096)
    {
        <error: attach 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 (!contains(2, $Job.OperationalStatus[])
        {
            <error: attach job failed, stop recipe examine CIM_Error>
        }
    }
```

```
          if ($SRC.SupportedSynchronizationType != 6) // not a clone
          { // locate new StorageSynchronized association if mirror created
             $SL[] = References(
                $TV->,
                “CIM_StorageSynchronized”,
                “SyncedElement”,
                false, false, null)
             $SS = $SL[0]
             if ($SS.SyncState != $SRC.InitialReplicationState)
             {
                <wait for $SS.SyncState instance mod indication>
                $SS = GetInstance( // refresh the SyncState value
                   $SS->,
                   false, false, false, null)
             }
          }
     } else
     {
          <error: cannot create a remote mirror/cannot create a target element>
          #d = &DeleteGoal ($ModSetting)
     } // end of step 4.

     // End of recipe. If successful, $TV is an instance of the remote mirror
     // or clone and $SS is an instance of the StorageSynchronized association
     // to the mirror.
```

8.2.8.12.6.9.4 Create Snapshot

The Create Snapshot recipe has a long section showing how to create a special purpose storage pool that may contain only delta replicas. These specialized pools may be shared or exclusive, created from special extents or within another pool and may be populated with virtual storage elements that do not consume space until attached to a source element as a replica.

```
     // NAME: Create Snapshot
     // FILE: CopyServicesSP_Recipe4of7
     //
     // DESCRIPTION: Create a snapshot of a source element. The snapshot is a
     // PIT image. Client indicates CopyType “UnSyncAssoc”.
     // The replica element is hosted on the same array as
     // the source element. The recipe supports both CreateReplica and
     // AttachOrModifyReplica extrinsic methods. If the attach method is
     // supported, search for a target element to attach. If no target element
     // is found, search for a target pool, create an element in the pool and
     // attach the new element. If the create method is supported, search for
     // a target pool and create the new replica in the pool.
     //
     // Delta replicas may require a special storage pool as a container. The
     // recipe searches for an existing special pool. If a special pool is not
     // found, one is created.
     //
```

```
// Snapshots are created with a setting element that requires the
// DeltaReservationMin, DeltaReservationGoal and DeltaReservationMax
// properties to have assigned values.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SCC: instance of CIM_StorageConfigurationCapabilities controlling
// the recipe.
// $SRC: instance of CIM_StorageReplicationCapabilities controlling
// the recipe.
// $SRC is for copy type “UnSyncAssoc-Delta” or “UnSyncAssoc-Full”.
// $SCS is the instance of CIM_StorageConfigurationService controlling
//  the recipe.
// $System is the instance of CIM_ComputerSystem identifying the array.
// $SV is the the replica source instance of CIM_StorageVolume.
// #UseDeltaReservationDefault: if false, user supplies 3 variables:
// #DRMin: user-assigned value for DeltaReservationMin.
// #DRGoal: user-assigned value for DeltaReservationGoal.
// #DRMax: user-assigned value for DeltaReservationMax.
// If a special delta replica pool may be created, user
// supplies 3 variables:
// #PoolSize: requested size of the special pool.
// #ComponentSize: size of component extents. Multiple of #PoolSize.
// #NumVirtVols: number of 0MB virtual volumes to create in the pool.
//

// CreateSpecialUsageExtent subroutine. Creates a component extent for
// the special purpose indicated by #IntendedUsage:
//  3: component for special delta replica storage pool
//  8: component for remote replication buffer
//
sub uint8 CreateSpecialUsageExtent
                        (IN $Pool, IN #IntendedUsage, IN #Size, OUT $TX)
{
// Create a setting element for the component creation.
#r = &MakeGoalParameter ($Pool, #IntendedUsage, $ModSetting)
// Create the component extent in $Pool
%InArguments[“ElementType”] = 3 // StorageExtent
%InArguments[“Goal”] = $ModSetting.getObjectPath()
%InArguments[“Size”] = #Size
%InArguments[“InPool”] = $Pool->
%InArguments[“TheElement”] = null
#r = InvokeMethod(
     $SCS->,
     “CreateOrModifyElementFromStoragePool”,
     %InArguments,
     %OutArguments)
```

```
#d = &DeleteGoal ($ModSetting)
if (#r != 0 && #r != 4096)
{
     <error: element creation 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 (!contains(2, $Job.OperationalStatus[])
     {
        <error: creation job failed, stop and examine CIM_Error>
     }
     $L[] = Associators(
        $Job->,
        “CIM_AffectedJobElement”,
        “CIM_StorageExtent”,
        null, null, false, false, null)
     $TX = $L[0]
} else
{
     $TX-> = %OutArguments[“TheElement”]
     $TX = GetInstance(
        $TX->,
        false, false, false, null)
}
return 0
} // *** end of CreateSpecialUsageExtent subroutine

// Main section of recipe

// Control variables based on configuration and replication capabilities.
#CrEl, #AttRep, #CrRep, #CrPoolFrExt, #CrPoolFrPool, #CrExt = false
if (contains(3, $SCC.SupportedStorageElementFeatures[]) &&
     (contains(6, $SCC.SupportedStorageElementFeatures[]) ||
      contains(7, $SCC.SupportedStorageElementFeatures[]))
{
     // CreateOrModifyElementFromStoragePool supported for volumes
     #CrEl = true
}
if (contains(2, $SCC.SupportedStorageElementFeatures[]) &&
     (contains(6, $SCC.SupportedStorageElementFeatures[]) ||
      contains(7, $SCC.SupportedStorageElementFeatures[]))
{
     // CreateOrModifyElementFromStoragePool supported for extents
     #CrExt = true
}
if (contains(2, $SCC.SupportedSynchronousActions[]) ||
     contains(2, $SCC.SupportedAsynchronousActions[]))
{
     if (contains(3, $SCC.SupportedStoragePoolFeatures[]) ||
         contains(4, $SCC.SupportedStoragePoolFeatures[]))
     {
        #CrPoolFrPool = true
     }
     if (contains(2, $SCC.SupportedStoragePoolFeatures[]))
     {
        #CrPoolFrExt = true
     }
}
if (contains(2, $SRC.SupportedSynchronousActions[]) ||
     contains(2, $SRC.SupportedAsynchronousActions[]))
{
     #CrRep = true // CreateReplica method supported
}
if (contains(6, $SRC.SupportedSynchronousActions[]) ||
     contains(6, $SRC.SupportedAsynchronousActions[]))
{
     #AttRep = true // AttachOrModifyReplica method supported
}
#IntendedUsage = 0 // assume specialized elements not supported
if ($SRC.SupportedSynchronizationType == 4) // “UnSyncAssoc-Full”
{
      if (contains(6, $SupportedSpecializedElements[]))
     {
        #IntendedUsage = 6 // “Full Snapshot”
     }
} else // must be “UnSyncAssoc-Delta”
{
     if (contains(7, $SupportedSpecializedElements[]))
     {
        #IntendedUsage = 7 // “Delta Snapshot”
     }
}
#drpa = $SRC.DeltaReplicaPoolAccess

// Step 1: Find the special delta replica pool if it already exists.
// Create a special pool if required and it does not exist.
if (#drpa != 2)
{ // a special delta replica pool is required
     $PoolList[] = Associators( // an exclusive pool for source element?
         $SV->,
         “CIM_ReplicaPoolForStorage”,
         “CIM_StoragePool”,
         null, null, false, false, null)
     if ($PoolList[].size() == 0)
     { // an exclsuive pool does not exist. Look for a shared pool.
         $PoolList[] = Associators( // a shared pool for the service?
             $SCS->,
             “CIM_ReplicaPoolForStorage”,
             “CIM_StoragePool”,
             null, null, false, false, null)
     }
     if ($PoolList[].size() == 0) // Special pool does not exist
     { // Find a container pool and component extents for a new pool

// Search all hosted storage pools. Find the pool with the most remaining
// capacity as well as the pool with a set of component extents with
// either the best match to #PoolSize or the most component space.
         #poolspace, #maxsize = 0
         #bestsize = 0xFFFFFFFF
         $MostSP, $BestSP, $MaxSP = null
         $BestXL[], $MaxXL[] = null
         $SPL[] = Associators( // all hosted storage pools
             $System->,
             “CIM_HostedStoragePool”,
             “CIM_StoragePool”,
             null, null, false, false, null)
         for #i in $SPL[]
         {
             // Make setting for GetAvailableExtents Goal parameter
             $CP = $SPL[#i]
             #rr = &MakeGoalParameter ($CP, 3, $ModSetting)
             %InArguments[“Goal”] = $ModSetting.getObjectPath()
             #rr = InvokeMethod(
               $CP->,
               “GetAvailableExtents”,
               %InArguments,
               %OutArguments)
             if (#rr == 1)
             {
               break // method not supported
             }
             if (#rr != 0)
             {
               <error: method invocation failed>
             }
             $SPX->[] = %OutArguments[“AvailableExtents”]
            #k, #xmax, #xbest = 0
            $CXL[] = null
            for #j in $SPX->[]
            {
               $SX = GetInstance(
                  $SPX->[#j],
                  false, false, false, null)
               #xsize = $SX.NumberOfBlocks * $SX.BlockSize
               if (#xbest < #PoolSize)
               {
                  #xbest = #xbest + #xsize
                  $CXL[#k] = $SX
                  #k++
               }
               #xmax = #xmax + 1
            }
            if (#xmax > #maxsize)
            { // current pool has largest collected extent size
               $MaxXL[] = $CXL[]
               $MaxSP = $CP
               #maxsize = #xmax
            }
            if ((#xbest >= #PoolSize) && (#xbest < #bestsize))
            { // current pool provides best fit collected extents
               $BestXL[] = $CXL[]
               $BestSP = $CP
               #bestsize = #xbest
            }
            // done analyzing component extents.
            #d = &DeleteGoal ($ModSetting)
            // Make a setting for the GetSupported_ methods
            #rr = &MakeGoalParameter ($CP, 2, $ModSetting)
            // will current pool allow creating #PoolSize pool?
            #rr = &SizeCheck ($CP, 2, #PoolSize, $ModSetting)
            if (#rr == 0) // current pool supports #PoolSize
            {
               if($CP.RemainingManagedSpace > #poolspace)
               { // current pool has most remaining space
                  $MostSP = $CP
                  #poolspace = $CP.RemainingManagedSpace
               }
            }
         } // done analyzing the current pool
      } // done searching all pools

//
// Finished searching all pools for best candidate to contain a new delta
// replica pool. May need to create one or more component extents before
// creating the new pool. Will create the new pool based on one of these
// policies:
//   1. DeltaReplicaPoolAccess is “Shared” and #maxsize >= #PoolSize.
//      Will make the new pool as large as possible from the located set
//      of extents.
//   2. DeltaReplicaPoolAccess is “Shared” and #maxsize <= #PoolSize.
//      Will create additional extents of #ComponentSize for the best
//      fit >= #PoolSize.
//   3. DeltaReplicaPoolAccess is “Exclusive” and #bestsize >= #PoolSize.
//      Will make the new pool a best fit >= #PoolSize from $BestXL[].
//   4. DeltaReplicaPoolAccess is “Exclusive” and #bestsize < #PoolSize.
//      Will create additional extents of #ComponentSize for the best
//      fit >= #PoolSize.
//   5. Cannot create special pool from a set of extents. Create from an
//      eligible pool.
//
     $CP, $CX[] = null
     if (#CrPoolFrExt)
     {
        if (((#drpa == 3) && (#bestsize < #PoolSize)) ||
          ((#drpa == 4) && (#maxsize < #PoolSize) && (#maxsize > 0)))
        { // policies 2 and 4: best fit from new and existing extents
           if (#CrExt)
           {
              $CP = $BestSP
              #inext = $BestXL[].size()
              #ilast = #inext +
                            ((#PoolSize - #bestsize)/#ComponentSize) + 1
                    // start by using the existing eligible extents
              $CX[] = $BestXL[]
              while (#inext < #ilast)
              { // add additional new extents as needed
                 #rr = &CreateSpecialUsageExtent
                                   ($CP, 2, #ComponentSize, $TX)
                 $CX[#inext] = $TX
                 #inext++
              }
           }
        } else
        {
           if (#drpa == 4)
           { // policy 1: largest possible shared pool
              $CP = $MaxSP
              $CX[] = $MaxXL[]
           } else
           { // policy 3: best fit from existing extents
```

```
                $CP = $BestSP
                $CX[] = $BestXL[]
            }
        }
    }
    if ($CP == null && #CrPoolFrPool) // policy 5:
    { // No component extents available.
       // May be possible to create in a pool.
        $CP = $MostSP
    }

// Create the specialized delta replica pool. Will be created from either
// a located set of component extents or in an eligible pool.
    if ($CP == null)
    {
        <error: cannot create a delta replica pool>
    }
    if ($CX[].size() > 0)
    { // setup to create from component extents
        %InArguments["InExtents"] = $CX->[]
        %InArguments["InPools"] = null
    } else // setup to create in a pool
    {
        %InArguments["InExtents"] = null
        $InPools->[0] = $CP->
        %InArguments["InPools"] = $InPools->[]
    }
    #rr = &MakeGoalParameter ($CP, 2, $ModSetting)
    %InArguments["Goal"] = $ModSetting.getObjectPath()
    %InArguments["Size"] = #PoolSize
    #rr = InvokeMethod (
        $SCS->,
        "CreateOrModifyStoragePool",
        %InArguments,
        %OutArguments)
    #d = &DeleteGoal ($ModSetting)
    if (#r != 0 && #r != 4096)
    {
        <error: pool creation failed, stop; 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 (!contains(2, $Job.OperationalStatus[])
            {
               <error: creation job failed, examine CIM_Error>
            }
            $L[] = Associators(
               $Job->,
               “CIM_AffectedJobElement”,
               “CIM_StoragePool”,
               null, null, false, false, null)
            $Pool = $L[0]
         } else
         {
            $Pool-> = %OutArguments[“Pool”]
            $Pool = GetInstance(
               $Pool->,
               false, false, false, null)
         }

   // Populate the delta replica pool with virtual volumes if requested.
         if ((#NumVirtVols > 0) && (#IntendedUsage == 7))
         {
            #i = 0
            while (#i < #NumVirtVols)
            {
                     // goal for usage=delta snapshot
               #rr = &MakeGoalParameter ($Pool, 7, $ModSetting)
               %InArguments[“ElementType”] = 2 // StorageVolume
               %InArguments[“Goal”] = $ModSetting.getObjectPath()
               %InArguments[“Size”] = 0
               %InArguments[“InPool”] = $Pool->
               %InArguments[“TheElement”] = null
               #r = InvokeMethod(
                  $SCS->,
                  “CreateOrModifyElementFromStoragePool”,
                  %InArguments,
                  %OutArguments)
               #d = &DeleteGoal ($ModSetting)
               if (#r != 0 && #r != 4096)
               {
                  <error: element 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 (!contains(2, $Job.OperationalStatus[])
                {
                   <error: creation job failed,
                                                  examine CIM_Error>
                }
             }
          #i++
          }
       }
    } // finished locating or creating a delta replica pool

    // Step 2: find a target element and/or a target pool for the snapshot.
    // If a special pool is used for delta replicas, $Pool is already set.
    if (#drpa != 2)
    {
       #Size = 0 // May have size=0 virtual volumes available
    } else
    {
        // Request same size as source element
       #Size = $SV.NumberOfBlocks * $SV.BlockSize
       $PoolList[] = Associators (
          $System->,
          “CIM_HostedStoragePool”,
          “CIM_StoragePool”,
          null, null, false, false, null)
    }
    for #ii in $PoolList[]
    {
       $Pool = $PoolList[#ii]
       if (#Size != 0)
       {
          $L[] = Associators(
             $Pool->,
             “CIM_ElementCapabilities”,
             “CIM_StorageCapabilities”,
             null, null, false, false, null)
          $Capabilities = $L[0]
          // adjust #Size value based on DeltaReservationDefault
          // or DeltaReservationGoal.
          if (!#UseDeltaReservationDefault)
          {
             if (($Capabilities.DeltaReservationMin
                                                   <= #DRMin) &&
                   (#DRMin <= #DRGoal) && (#DRGoal <= #DRMax) &&
                 (#DRMax <= $Capabilities.DeltaReservationMax))
             {
                #Size = (#Size * #DRGoal) / 100
             } else
             {
                <error: invalid Delta Reservation values>
             }
          } else
          {
             #Size = (#Size *
                            $Capabilities.DeltaReservationDefault) / 100
          }
       }
       #rr = &FindTargetElementOrPool ($Pool, #IntendedUsage, #Size, $TV)
       if (#rr != 2)
       {
          break
       }
    }
    if (#rr == 2)
    {
       <error: cannot create a snapshot/no target element or pool>
    }

    // Step 3: Modify the setting element so all of the DeltaReservation
    // properties are set to valid values. This setting will be used to
    // create a new snapshot element or modify the setting properties for
    // an existing target element that becomes a snapshot.
    if (!#UseDeltaReservationDefault)
    {
       if (($ModSetting.DeltaReservationMin <= #DRMin) &&
            (#DRMin <= #DRGoal) && (#DRGoal <= #DRMax) &&
            (#DRMax <= $ModSetting.DeltaReservationMax))
       {
          $ModSetting.DeltaReservationMin = #DRMin
          $ModSetting.DeltaReservationGoal = #DRGoal
          $ModSetting.DeltaReservationMax = #DRMax
       } else
       {
          <error: invalid Delta Reservation values>
       }
    }


    // Step 4: if $TV returned as target and #AttRep is true, invoke
    // the AttachOrModifyReplica method.
    if ((#rr == 0) && #AttRep)
```

```
{
     %InArguments[“SourceElement”] = $SV->
     %InArguments[“TargetElement”] = $TV->
     %InArguments[“CopyType”] = 4
     %InArguments[“Goal”] = $ModSetting
     %InArguments[“ReplicationPipe”] = null // for snapshot
     #r = InvokeMethod(
        $SCS->,
        “AttachOrModifyReplica”,
        %InArguments,
        %OutArguments)
     if (#r != 0 && #r != 4096)
     {
        <error: attach failed, stop recipe 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 (!contains(2, $Job.OperationalStatus[])
        {
           <error: creation job failed, stop and examine CIM_Error>
        }
     }

     // locate new StorageSynchronized association if snapshot created
     $L[] = References(
        $TV->,
        “CIM_StorageSynchronized”,
        “SyncedElement”,
        false, false, null)
     $SS = $L[0]
     if ($SS.SyncState != $SRC.InitialReplicationState)
     {
        <wait for $SS.SyncState instance modification indication>
        $SS = GetInstance( // refresh SyncState value
           $SS->,
           false, false, false, null)
     }
// stop recipe if step 4 was executed.
}

// Step 5: if a target pool was returned and #CrRep is true, invoke
// the CreateReplica method.
```

```
if ((#rr == 1) && #CrRep)
{
     %InArguments[“SourceElement”] = $SV->
     %InArguments[“CopyType”] = 4
     %InArguments[“TargetSettingGoal”] = $ModSetting.getObjectPath()
     %InArguments[“TargetPool”] = $Pool->
     #r = InvokeMethod(
        $SCS->,
        “CreateReplica”,
        %InArguments,
        %OutArguments)
     #d = &DeleteGoal ($ModSetting)
     if (#r != 0 && #r != 4096)
     {
        <error: attach 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 (!contains(2, $Job.OperationalStatus[])
        {
           <error: creation job failed, stop and examine CIM_Error>
        }
        $L[] = Associators(
           $Job->,
           “CIM_AffectedJobElement”,
           “CIM_StorageVolume”,
           null, null, false, false, null)
        $TV = $L[0]
     } else
     {
        $TV-> = %OutArguments[“TheElement”]
        $TV = GetInstance(
           $TV->,
           false, false, false, null)
     }
     // locate new StorageSynchronized association if snapshot created
     $SS = References(
        $TV->,
        “CIM_StorageSynchronized”,
        “SyncedElement”,
        false, false, null)
     if ($SS.SyncState != $SRC.InitialReplicationState)
     {
        <wait for $SS.SyncState instance modification indication>
        $SS = GetInstance( // refresh SyncState value
           $SS->,
           false, false, false, null)
     }
// stop recipe if step 5 was executed.
}

// Step 6: if a target pool was returned and #AttRep is true, invoke
// the CreateOrModifyElementFromStoragePool method followed by the
// AttachOrModifyReplica method.
if ((#rr == 1) && #AttRep && #CrEl)
{
     %InArguments[“ElementType”] = 2 // StorageVolume
     %InArguments[“Goal”] = $ModSetting.getObjectPath()
     %InArguments[“Size”] = #Size // was calculated earlier
     %InArguments[“InPool”] = $Pool->
     %InArguments[“TheElement”] = null
     #r = InvokeMethod(
        $SCS->,
        “CreateOrModifyElementFromStoragePool”,
        %InArguments,
        %OutArguments)
     #d = &DeleteGoal ($ModSetting)
     if (#r != 0 && #r != 4096)
     {
        <error: element 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 (!contains(2, $Job.OperationalStatus[])
        {
           <error: creation job failed, stop and examine CIM_Error>
        }
        $L[] = Associators(
           $Job->,
           “CIM_AffectedJobElement”,
           “CIM_StorageVolume”,
           null, null, false, false, null)
        $TV = $L[0]
     } else
```

```
        {
           $TV-> = %OutArguments[“TheElement”]
           $TV = GetInstance(
              $TV->,
              false, false, false, null)
        }
        %InArguments[“SourceElement”] = $SV->
        %InArguments[“TargetElement”] = $TV->
        %InArguments[“CopyType”] = 4
        %InArguments[“Goal”] = null // already applied setting
        %InArguments[“ReplicationPipe”] = null // for snapshot
        #r = InvokeMethod(
           $SCS->,
           “AttachOrModifyReplica”,
           %InArguments,
           %OutArguments)
        if (#r != 0 && #r != 4096)
        {
           <error: attach 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 (!contains(2, $Job.OperationalStatus[])
           {
              <error: creation job failed, stop and examine CIM_Error>
           }
        }
        // locate new StorageSynchronized association if snapshot created
        $L[] = References(
           $TV->,
           “CIM_StorageSynchronized”,
           “SyncedElement”,
           false, false, null)
        $SS = $L[0]
        if ($SS.SyncState != $SRC.InitialReplicationState)
        {
           <wait for $SS.SyncState instance modification indication>
           $SS = GetInstance( // refresh SyncState value
              $SS->,
              false, false, false, null)
        }
     } else
```

```
{
     <error: cannot create a snapshot/cannot create a target element>
     #d = &DeleteGoal ($ModSetting)
} // end of step 6.

// End of recipe. If successful, $TV is an instance of StorageVolume
// representing the snapshot. $SS is an instance of StorageSynchronized
// associating the snapshot to its source element.
```

8.2.8.12.6.9.5 Modify Replica

```
// NAME: Modify Replica
// FILE: CopyServicesSP_Recipe5of7
//
// 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 provider fails the
// operation and returns a CIM_Error instance if an invalid state
// transition is attempted.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SourceSystem and $TargetSystem are instances of ComputerSystem for
// the arrays hosting the source volume and target volume respectively.
// Will be the same if replication is local and different if replication
// is remote.
// $SourceVolume and $TargetVolume are instances of StorageVolume
// representing the replica source and replica target respectively on
// their hosting arrays.
// #operation is the modify operation to be executed.
//

//
// SetupServiceAndCapabilities subroutine: locate the service and
// capabilities instances used to invoke modify operations. Determine
// the SAP for modify operations. Locate the StorageSynchronized
// association for the replica pair.
//
sub uint8 SetupServiceAndCapabilities (IN $SourceSystem, IN $SourceVolume,
     IN $TargetSystem, IN $TargetVolume, OUT $SCS, OUT $SRC, OUT $StgSync)
{
// locate the modify SAP for the replica. If this is a local replica, the
// SAP is always the source host.
// For a remote replica, StorageReplicationCapabilities indicates if
// the SAP is the source or the target host.
     $L[] = References(
        $TargetVolume->,
        "CIM_StorageSynchronized",
        "SyncedElement",
        false, false,
        {"CopyType", "SyncState"})
     If ($L[].size() != 1)
     {
        <error: volume is not an associated replica target>
     }
     $StgSync = $L[0]
     $L[] = Associators(
            // locate StorageConfigurationService for source array
        $SourceSystem->,
        "CIM_HostedService",
        "CIM_StorageConfigurationService",
        null, null, false, false, null)
     $SCS = $L[0]
     $L[] = Associators(
            // locate StorageReplicationCapabilities for the service
        $SCS->,
        "CIM_ElementCapabilities",
        "CIM_StorageReplicationCapabilities",
        null, null, false, false, null)
     #CT = $StgSync.CopyType
     if ($StgSync.CopyType == 5) // adjust before comparison loop
        // SupportedSynchronizationType defines UnSyncUnAssoc as 6 while
        // CopyType defines it as 5. Refer to MOF file.
     { // (2|3|4|5) -> (2|3|4|6)
        #CT++
     }
     for #i in $L[]
     {
        if ($L[#i].SupportedSynchronizationType == #CT)
        {
           $SRC = $L[#i]
           break
        }
     }
// $SCS and $SRC are set for source array. Should they be set for
// the target array instead?
     if ($SourceSystem.Name != $TargetSystem.Name)
      // Name values != implies remote
     { // if remote, is SAP the target array?
        if ($SRC.RemoteReplicationServicePointAccess == 4)
        { // switch $SCS and $SRC to target array
           $L[] = Associators(
                  // locate StorageConfigurationService for target
               $TargetSystem->,
               "CIM_HostedService",
               "CIM_StorageConfigurationService",
               null, null, false, false, null)
           $SCS = $L[0]
           $L[] = Associators(
                  // locate StorageReplicationCapabilities
               $SCS->,
               "CIM_ElementCapabilities",
               "CIM_StorageReplicationCapabilities",
               null, null, false, false, null)
           for #i in $L[]
           {
               if ($L[#i].SupportedSynchronizationType == #CT)
               {
                   $SRC = $L[#i]
                   break
               }
           }
        } else
        { // SAP is source array. Need StorageSynchronized
          // instance in source SMI-S server/CIMOM.
           if ($SRC.RemoteReplicationServicePointAccess == 3)
           { // switch $StgSync to source array CIMOM
               $L[] = Associators(
                   $SourceVolume->,
                   "CIM_StorageSynchronized",
                   "CIM_StorageVolume",
                   "SystemElement",
                   "SyncedElement",
                   false, false,
                   "Name")
               for #i in $L[]
               {
                   if ($L[#i].Name == $TargetVolume.Name)
                   {
                       $LL[] = References(
                           $L[#i].getObjectPath(),
                           "CIM_StorageSynchronized",
                           "SyncedElement",
                           false, false,
                           {"CopyType", "SyncState"})
                       $StgSync = $LL[0]
                       break
                   }
               }
           }
```

```
            }
        }
        return 0
    } // end of subroutine

    // Main section of recipe
    #r = &SetupServiceAndCapabilities($SourceSystem, $SourceVolume,
                        $TargetSystem,$TargetVolume, $SCS, $SRC, $StgSync)
    if (!contains(#operation, $SRC.SupportedModifyOperations[]))
    {
        <error: requested ModifySynchronization operation unsupported>
    }
    %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 (!contains(2, $Job.OperationalStatus[]) // is it "OK"?
        {
            <error: modify job failed, stop and examine CIM_Error>
        }
    }
    if (#operation != 2)
    // if not "detach" get a fresh copy of StorageSynchronized
    {
        $StgSync = GetInstance(
            $StgSync->,
            false, false, false,
            {"CopyType", "SyncState"})
         // list of replica "...In Progress" association states
        #InProgress = {3,5,7,8,10,15}
        if (contains($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,
                {"CopyType", "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.
```

8.2.8.12.6.9.6 Delete Replica

```
// NAME: Delete Replica
// FILE: CopyServicesSP_Recipe6of7
//
// DESCRIPTION: An associated replica target element is deleted and its
// consumed space is returned to the containing storage pool. If the
// provider supports detach operations, the StorageSynchronized
// association will be detached/deleted before the replica element
// is deleted. The recipe includes safety checks to ensure that the
// replica target is not in use when the delete request is received.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// $SourceSystem and $TargetSystem are instances of ComputerSystem for
// the arrays hosting the source volume and target volume respectively.
// Will be the same if replication is local and different if replication
// is remote.
// $SourceVolume and $TargetVolume are instances of StorageVolume
// representing the replica source and replica target respectively on
// their hosting arrays.
//
// Recipe uses the SetupServiceAndCapabilities subroutine in the
// Modify Replica recipe.
//

// Main section of recipe

#r = &SetupServiceAndCapabilities($SourceSystem, $SourceVolume,
                    $TargetSystem,$TargetVolume, $SCS, $SRC, $StgSync)
// Verify that ReturnToStoragePool is supported.
$L[] = Associators(
     $TargetSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$TargetSAP = $L[0]
$L[] = Associators(
     $TargetSAP->,
     “CIM_ElementCapabilities”,
     “CIM_StorageConfigurationCapabilities”,
     null, null, false, false, null)
$TargetSCC = $L[0]
if ((!contains(6, $TargetSCC.SupportedSynchronousActions[])) &&
    (!contains(6, $TargetSCC.SupportedAsynchronousActions[])))
{
     <error: storage element return operation not supported>
}

// The safety checks are illustrative of possible client best
// practices and should not be included in CTP test cases.
// Do two safety checks before deleting the element
$Refs[] = ReferenceNames(
     $TargetVolume->,
     “CIM_ProtocolControllerForUnit”,
     null)
if ($Refs[].size() > 0)
{
     <error: replica target is exposed for host access>
}
$Refs[] = ReferenceNames(
     $TargetVolume->,
     “CIM_StorageSynchronized”,
     “SystemElement”)
if ($Refs[].size() > 0)
{
     <error: replica target is also a replica source>
}

// If provider supports detach operation, detach before deleting element.
if (contains(2, $SRC.SupportedModifyOperations[]))
{
     %InArguments[“Synchronization”] = $StgSync->
     %InArgument[“Operation”] = 2 // detach
     #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 mod indication for job completion>
        $Job = GetInstance(
            $Job->,
            false, false, false, null)
        if (!contains(2, $Job.OperationalStatus[]) // is it “OK”?
        {
            <error: modify job failed, stop and examine CIM_Error>
        }
     }
}
// End of safety checks. The remainder of the recipe is required for CTP.

// Delete the replica target element
%InArguments[“TheElement”] = $TargetVolume->
#r = InvokeMethod(
     $TargetSAP->,
     “ReturnToStoragePool”,
     %InArguments,
     %OutArguments)
if (#r != 0 && #r != 4096)
{
     <error: return failed, stop recipe and examine CIM_Error>
     // if the error MessageID is MP3, “Property Not Found”,
     // the replica may have been implicitly deleted by the
     // preceding Detach operation.
}
if (#r == 4096)
{
     $Job-> = %OutArguments[“Job”]
     <wait for instance modification indication for job completion>
     $Job = GetInstance(
        $Job->,
        false, false, false, null)
     if (!contains(2, $Job.OperationalStatus[]) // is it “OK”?
     {
        <error: modify job failed, stop recipe and examine CIM_Error>
     }
}

// end of recipe.
```

8.2.8.12.6.9.7 Create Remote Replication Buffer

```
// NAME: Create Remote Replication Buffer
// FILE: CopyServicesSP_Recipe7of7
//
// 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 three
// properties in the instance of StorageReplicationCapabilities:
// RemoteBufferElementType, RemoteBufferHost and RemoteBufferLocation.
// The outputs are references $SourceBuffer and $TargetBuffer.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTIONS:
//
// Provider supports remote replication using CopyType “Async” or “Sync”
// $SourceSystem is a top-level or leaf ComputerSystem for source array
// $TargetSystem is a top-level or leaf ComputerSystem for target array
// $SourcePipe is a NetworkPipe representing the source/target connection.
// $TargetPipe is a NetworkPipe representing the target/source connection.
// $SourcePipe and $TargetPipe may reference the same instance if the
// source and target arrays share a single SMI-S server/CIMOM. Both must
// reference a top-level pipe.
// $_Pipe references may be null unless RemoteBufferHost == “Pipe”.
// #BufferSize indicates extent size if RemoteBufferElementType
//      = “InExtent”.
//
// This recipe uses subroutines &MakeGoalParameter, &SizeCheck and
// &CreateSpecialUsageExtent in prior recipes in the subprofile.
//

// Setup control variables for the recipe.
$L[] = Associators(
     $SourceSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$SourceSCS = $L[0]
$L[] = Associators(
     $TargetSystem->,
     “CIM_HostedService”,
     “CIM_StorageConfigurationService”,
     null, null, false, false, null)
$TargetSCS = $L[0]
$SRC[] = Associators(
     $SourceSCS->,
     “CIM_ElementCapabilities”,
     “CIM_StorageReplicationCapabilities”,
     null, null, false, false, null)
$ReplicaSRC = null
for #i in $SRC[]
{
     if (($SRC[#i].SupportedSynchronizationType == 2) ||
          ($SRC[#i].SupportedSynchronizationType == 3))
          // CopyType “Async” or “Sync”
     { // action 8 is “Buffer Creation”
        if ((contains(8, $SRC[#i].SupportedSynchronousActions[])) ||
            (contains(8, $SRC[#i].SupportedAsynchronousActions[])))
        {
           $ReplicaSRC = $SRC[#i]
           break
        }
     }
}
if ($ReplicaSRC == null)
{
     <error: buffer creation not supported>
}
#host = $ReplicaSRC.RemoteBufferHost // top-level CS, leaf CS or pipe
#location = $ReplicaSRC.RemoteBufferLocation // source, target or both
#container = $ReplicaSRC.RemoteBufferElementType
                        // buffer is extent, element in a pool or opaque

// Locate pools to contain the replication buffers
$SourcePool, $TargetPool = null
if (#container != 2)
{
     #BufferSize = 0
}
if (#container != 0) // need pool unless “not specified”
{
     if ((#location == 2) || (#location == 4))
     { // need pool on source array
        $PoolList[] = Associators (
           $SourceSystem->,
           “CIM_HostedStoragePool”,
           “CIM_StoragePool”,
           null, null, false, false, null)
        for #i in $PoolList[]
        {
           $Pool = $PoolList[#i]
           // Make a setting for the GetSupported_ methods
           #r = &MakeGoalParameter ($Pool, 8, $ModSetting)
           // will current pool allow creating a replication buffer?
           #r = &SizeCheck ($Pool, 3, #BufferSize, $ModSetting)
```

```
                #d = &DeleteGoal ($ModSetting)
                if (#r == 0) // current pool supports replication buffers
                {
                    $SourcePool = $Pool
                    break
                }
            }
            if ($SourcePool == null)
            {
                <error: eligible pool not found>
            }
        }
        if ((#location == 3) || (#location == 4))
        { // need pool on target array
            $PoolList[] = Associators (
                $TargetSystem->,
                “CIM_HostedStoragePool”,
                “CIM_StoragePool”,
                null, null, false, false, null)
            for #i in $PoolList[]
            {
                $Pool = $PoolList[#i]
                // Make a setting for the GetSupported_ methods
                #r = &MakeGoalParameter ($Pool, 8, $ModSetting)
                // will pool allow creating a replication buffer?
                #r = &SizeCheck ($Pool, 3, #BufferSize, $ModSetting)
                #d = &DeleteGoal ($ModSetting)
                if (#r == 0) // pool supports replication buffers?
                {
                    $TargetPool = $Pool
                    break
                }
            }
            if ($TargetPool == null)
            {
                <error: eligible pool not found>
            }
        }
    }

    // Create component extents if required
    $SourceExtent->, $TargetExtent-> = null
    if (#container == 2)
    {
        if ((#location == 2) || (#location == 4))
        { // need extent on source array
            #r = &CreateSpecialUsageExtent
```

```
                         ($SourcePool, 8, #BufferSize, $SourceExtent)
            $SourcePool-> = null
        }
        if ((#location == 3) || (#location == 4))
        { // need extent on target array
            #r = &CreateSpecialUsageExtent
                         ($SourcePool, 8, #BufferSize, $TargetExtent)
            $TargetPool-> = null
        }
    }

    // Create the replication buffers
    if ((#location == 2) || (#location == 4)) // need buffer on source array
    {
        if (#host == 4) // buffer associated to connection pipe
        {
            %InArguments[“Host”] = $SourcePipe->
        } else
        {
            %InArguments[“Host”] = $SourceSystem->
        }

        %InArguments[“TargetElement”] = $SourceExtent->
        %InArguments[“TargetPool”] = $SourcePool->
        #r = InvokeMethod(
            $SourceSCS->,
            “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 (!contains(2, $Job.OperationalStatus[])
            {
                <error: creation job failed, stop and examine CIM_Error>
            }
            $L[] = Associators(
                $Job->,
                “CIM_AffectedJobElement”,
                “CIM_Memory”,
                null, null, false, false, null)
            $SourceBuffer-> = $L->[0]
        } else
        {
            $SourceBuffer-> = %OutArguments[“ReplicaBuffer”]
        }
    }
    if ((#location == 3) || (#location == 4)) // need buffer on target array
    {
        if (#host == 4) // buffer associated to connection pipe
        {
            %InArguments[“Host”] = $TargetPipe->
        } else
        {
            %InArguments[“Host”] = $TargetSystem->
        }

        %InArguments[“TargetElement”] = $TargetExtent->
        %InArguments[“TargetPool”] = $TargetPool->
        #r = InvokeMethod(
            $TargetSCS->,
            “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 (!contains(2, $Job.OperationalStatus[])
            {
                <error: creation job failed, stop and examine CIM_Error>
            }
            $L[] = Associators(
                $Job->,
                “CIM_AffectedJobElement”,
                “CIM_Memory”,
                null, null, false, false, null)
            $TargetBuffer-> = $L->[0]
        } else
```

```
          {
             $TargetBuffer-> = %OutArguments[“ReplicaBuffer”]
          }
     } // end of recipe
```

8.2.8.12.7 Registered Name and Version

Copy Services version 1.1.0

8.2.8.12.8 CIM Server Requirements

Table 1067: CIM Server Requirements for Copy Services

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.12.9 CIM Elements

Table 1068: CIM Elements for Copy Services

Element Name	Description
Mandatory Classes	
CIM_AllocatedFromStoragePool (8.2.8.12.9.1)	Base definition is in Block Services Package.
CIM_ComputerSystem (8.2.8.12.9.4)	Base definition is in Array Profile. Adds associations used for stitched remote peer connections.
CIM_ElementCapabilities (8.2.8.12.9.6)	
CIM_HostedService (8.2.8.12.9.9)	
CIM_LogicalDisk (8.2.8.12.9.10)	Base definition in Volume Management Profile. Refer to description of CIM_StorageVolume in this subprofile.
CIM_StorageCapabilities (8.2.8.12.9.17)	Base definition is in Block Services Package.
CIM_StorageConfigurationCapabilities (8.2.8.12.9.18)	Base definition is in Block Services Package. Adds two properties.
CIM_StorageConfigurationService (8.2.8.12.9.19)	Base definition is in Block Services Package. Methods are described in the Extrinsic Methods clause.
CIM_StoragePool (8.2.8.12.9.21)	Base definition is in Block Services Package.
CIM_StorageReplicationCapabilities (8.2.8.12.9.22)	A set of properties that describe the capabilities of a copy services provider.
CIM_StorageSetting (8.2.8.12.9.23)	Base definition is in Block Services Package.
CIM_StorageSynchronized (8.2.8.12.9.24)	Associates replica target element to a source element.
CIM_StorageVolume (8.2.8.12.9.25)	Base definition is in Array Profile. Adds associations used when the element is a replica source or a replica target.
Optional Classes	
CIM_AssociatedMemory (8.2.8.12.9.2)	Associates remote replication buffer to a peer system element or a replication pipe.
CIM_BasedOn (8.2.8.12.9.3)	May be used for replica buffer element created from a concrete extent.
CIM_ConcreteDependency (8.2.8.12.9.5)	Associates remote replica target element to a peer-to-peer connection.
CIM_EndpointOfNetworkPipe (8.2.8.12.9.7)	Associates peer-to-peer ProtocolEndpoint to lower-level NetworkPipe.
CIM_HostedNetworkPipe (8.2.8.12.9.8)	Associates replication NetworkPipe to replication Network domain element.
CIM_Memory (8.2.8.12.9.11)	Write ahead buffer element for a remote replication service.
CIM_Network (8.2.8.12.9.12)	Specialized peer-to-peer network for a remote replication service.
CIM_NetworkPipe (8.2.8.12.9.13)	Identifies peer-to-peer connection for a remote replication service.
CIM_NetworkPipeComposition (8.2.8.12.9.14)	Associates one top-level NetworkPipe to one or more lower-level pipes.
CIM_ProtocolEndpoint (8.2.8.12.9.15)	Base definition is in Fabric Profile. Special purpose endpoint that controls a peer-to-peer connection.

Table 1068: CIM Elements for Copy Services

Element Name	Description
CIM_ReplicaPoolForStorage (8.2.8.12.9.16)	Associates special storage pool for delta replicas to StorageConfigurationService or to a source element.
CIM_StorageExtent (8.2.8.12.9.20)	Concrete extent components for delta replica pools or remote replication buffers.
CIM_SystemComponent (8.2.8.12.9.26)	Associates peer system element to remote replication network domain.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StorageSynchronized	All instance creation indications for StorageSynchronized.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_StorageSynchronized	All instance deletion indications for StorageSynchronized.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageSynchronized AND SourceInstance.CIM_StorageSynchronized::SyncState <> PreviousInstance.CIM_StorageSynchronized::SyncState	CQL - CQL -- Synchronization state transition for a replica association.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_StorageSynchronized AND SourceInstance.SyncState <> PreviousInstance.SyncState	Deprecated WQL - Deprecated WQL -- Synchronization state transition for a replica association.
Optional Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_Memory	All instance creation indications for replication buffers.
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_Memory	All instance deletion indications for replication buffers.
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_NetworkPipe AND SourceInstance.CIM_NetworkPipe::OperationalStatus <> PreviousInstance.CIM_NetworkPipe::OperationalStatus	Change of peer-to-peer connection operational status.
SELECT * FROM CIM_AlertIndication WHERE OwnerEntity = "SNIA" AND AlertingManagedElement ISA CIM_StoragePool	Remaining pool space either below warning threshold set for the pool or there is no remaining space in the pool.
SELECT * FROM CIM_AlertIndication WHERE OwnerEntity = "SNIA" AND AlertingManagedElement ISA CIM_AllocatedFromStoragePool	Delta replica space consumed has either reached the warning threshold or has exceeded the space limit set for the replica.

8.2.8.12.9.1 CIM_AllocatedFromStoragePool

SpaceLimit and SpaceLimitWarningThreshold are only applicable when the dependent storage element is a delta replica with SpaceConsumed limits enforced by the provider.

Class Mandatory: true

Table 1069: SMI Referenced Properties/Methods for CIM_AllocatedFromStoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StoragePool	
Dependent		CIM_LogicalElement	
Optional Properties/Methods			
SpaceLimit	M	uint64	Space limit for a delta replica. If space limits are enforced, SpaceConsumed cannot exceed this value.
SpaceLimitWarningThreshold	M	uint16	Percentage of SpaceLimit. An indication is generated when SpaceConsumed reaches or exceeds the warning threshold level.

8.2.8.12.9.2 CIM_AssociatedMemory

Associates remote replication buffer to a peer system element or a replication pipe.
Created By : Extrinsic(s): CreateReplicationBuffer
Modified By : Extrinsic(s): Not modifiable
Deleted By : DeleteInstance
Class Mandatory: false

Table 1070: SMI Referenced Properties/Methods for CIM_AssociatedMemory

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Memory	
Dependent		CIM_LogicalElement	

8.2.8.12.9.3 CIM_BasedOn

May be used for replica buffer element created from a concrete extent.
Class Mandatory: false

Table 1071: SMI Referenced Properties/Methods for CIM_BasedOn

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageExtent	
Dependent		CIM_StorageExtent	

8.2.8.12.9.4 CIM_ComputerSystem

Base definition is in Array Profile. Adds associations used for stitched remote peer connections.
Class Mandatory: true
No specified properties or methods.

8.2.8.12.9.5 CIM_ConcreteDependency

Associates remote replica target element to a peer-to-peer connection.
Created By : Extrinsic(s): CreateReplica, AttachReplica, AttachOrModifyReplica
Deleted By : Extrinsic(s): ModifySynchronization

Class Mandatory: false

Table 1072: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Top-level NetworkPipe identifying the connection.
Dependent		CIM_ManagedElement	Remote replica target element.

8.2.8.12.9.6 CIM_ElementCapabilities

Class Mandatory: true

Table 1073: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	
Capabilities		CIM_Capabilities	

8.2.8.12.9.7 CIM_EndpointOfNetworkPipe

Associates peer-to-peer ProtocolEndpoint to lower-level NetworkPipe.
Class Mandatory: false

Table 1074: SMI Referenced Properties/Methods for CIM_EndpointOfNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ServiceAccessPoint	
Dependent		CIM_NetworkPipe	
Optional Properties/Methods			
SourceOrSink		uint16	Indicates endpoint for source or target host when the connection is uni-directional. Values: 0: Unknown 1: Source 2: Target (sink) 3: Not applicable

8.2.8.12.9.8 CIM_HostedNetworkPipe

Associates replication NetworkPipe to replication Network domain element.
Created By : Extrinsic(s): CreateOrModifyReplicationPipe
Deleted By : DeleteInstance
Class Mandatory: false

Table 1075: SMI Referenced Properties/Methods for CIM_HostedNetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_Network	

Table 1075: SMI Referenced Properties/Methods for CIM_HostedNetworkPipe

Property	Flags	Type	Description & Notes
Dependent		CIM_NetworkPipe	

8.2.8.12.9.9 CIM_HostedService

Class Mandatory: true

Table 1076: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	
Dependent		CIM_Service	

8.2.8.12.9.10 CIM_LogicalDisk

Base definition in Volume Management Profile. Refer to description of CIM_StorageVolume in this subprofile.
Class Mandatory: true
No specified properties or methods.

8.2.8.12.9.11 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(s): CreateReplicationBuffer
Deleted By : DeleteInstance
Class Mandatory: false

Table 1077: SMI Referenced Properties/Methods for CIM_Memory

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
DeviceID		string	
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	

8.2.8.12.9.12 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.

Class Mandatory: false

Table 1078: SMI Referenced Properties/Methods for CIM_Network

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	

Table 1078: SMI Referenced Properties/Methods for CIM_Network

Property	Flags	Type	Description & Notes
Name		string	
NameFormat		string	
Optional Properties/Methods			
ElementName		string	

8.2.8.12.9.13 CIM_NetworkPipe

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(s): CreateOrModifyReplicationConnection
Modified By : Extrinsic(s): CreateOrModifyReplicationConnection
Deleted By : DeleteInstance
Class Mandatory: false

Table 1079: SMI Referenced Properties/Methods for CIM_NetworkPipe

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName	MN	string	
OperationalStatus		uint16[]	Described in Health and Fault Management clause.
AggregationBehavior		uint16	Identifies pipe as top-level or lower-level: 2: lower-level pipe with two endpoints 4: top-level pipe
Directionality		uint16	Indicates bi-directional or uni-directional connection. Values: 0: unknown 2: bi-directional 3: uni-directional

8.2.8.12.9.14 CIM_NetworkPipeComposition

Associates one top-level NetworkPipe to one or more lower-level pipes.
Created By : Extrinsic(s): CreateOrModifyReplicationPipe
Deleted By : DeleteInstance
Class Mandatory: false

Table 1080: SMI Referenced Properties/Methods for CIM_NetworkPipeComposition

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_NetworkPipe	
PartComponent		CIM_NetworkPipe	

8.2.8.12.9.15 CIM_ProtocolEndpoint

Base definition is in Fabric Profile. Special purpose endpoint that controls a peer-to-peer connection.
Class Mandatory: false

Table 1081: SMI Referenced Properties/Methods for CIM_ProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ProtocolIFType		uint16	Value is always "Other".
OtherTypeDescription		string	String identifying the peer-to-peer connection protocol.

8.2.8.12.9.16 CIM_ReplicaPoolForStorage

Associates special storage pool for delta replicas to StorageConfigurationService or to a source element.
Class Mandatory: false

Table 1082: SMI Referenced Properties/Methods for CIM_ReplicaPoolForStorage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_EnabledLogicalElement	
Dependent		CIM_StoragePool	

8.2.8.12.9.17 CIM_StorageCapabilities

Base definition is in Block Services Package.
Class Mandatory: true

Table 1083: SMI Referenced Properties/Methods for CIM_StorageCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
DeltaReservationMin		uint16	Refer to property descriptions for CIM_StorageSetting class.
DeltaReservationMax		uint16	
DeltaReservationDefault		uint16	Initial value for CIM_StorageSetting.DeltaReservationGoal.

8.2.8.12.9.18 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.

Class Mandatory: true

Table 1084: SMI Referenced Properties/Methods for CIM_StorageConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SupportedAsynchronousActions	N	uint16[]	Identify replication methods using job control. Values: 8: Replica Creation 9: Replica Modification 10: Replica Attachment
SupportedSynchronousActions	N	uint16[]	Identify replication methods not using job control. Values: 8: Replica Creation 9: Replica Modification 10: Replica Attachment
SupportedStorageElementTypes		uint16[]	Storage element types that can be replicated. Values: 2: Storage Volume 4: Logical Disk
SupportedCopyTypes		uint16[]	CopyType values: 2: Async 3: Sync 4: UnSyncAssoc 5: UnSyncUnAssoc
InitialReplicationState		uint16	The initial SyncState when replica creation is completed. Values: 2: Initialized 3: Prepared 4: Synchronized

8.2.8.12.9.19 CIM_StorageConfigurationService

Base definition is in Block Services Package. Methods are described in the Extrinsic Methods clause.
Class Mandatory: true

Table 1085: SMI Referenced Properties/Methods for CIM_StorageConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ModifySynchronization()			
Optional Properties/Methods			
CreateOrModifyReplicationPipe()			
CreateReplicationBuffer()			
CreateReplica()			
AttachReplica()			Defined for 1.0.2 downward compatibility. Behaves as AttachOrModifyReplica without Goal and replicationPipe parameters.
AttachOrModifyReplica()			

8.2.8.12.9.20 CIM_StorageExtent

The Copy Services Subprofile supports explicit creation of concrete extents for two specialized uses. Concrete extents may be used as components for a specialized pool that will only contain delta replicas. Concrete extents may be used as an input parameter for CreateReplicationBuffer.

Created By : Extrinsic(s): CreateOrModifyElementFromStoragePool, CreateOrModifyElementFromElements
Modified By : Extrinsic(s): CreateOrModifyElementFromStoragePool, CreateOrModifyElementFromElements
Deleted By : Extrinsic(s): ReturnToStoragePool
Class Mandatory: false

Table 1086: SMI Referenced Properties/Methods for CIM_StorageExtent (Copy Services)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	

8.2.8.12.9.21 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.

Class Mandatory: true

Table 1087: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Optional Properties/Methods			
LowSpaceWarningThreshold	M	uint16	Percentage of TotalManagedSpace triggering an alert indication. When RemainingManagedSpace reaches or falls below this percentage, the indication is generated.

8.2.8.12.9.22 CIM_StorageReplicationCapabilities

This class defines all of the capability properties for a replication service. A provider must supply one instance for each CopyType value supported.

Class Mandatory: true

Table 1088: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SupportedSynchronizationType		uint16	Provider must supply one instance of this class for each supported CopyType value. Values: 2: Async 3: Sync 4: UnSyncAssoc-Full 5: UnSyncAssoc-Delta 6: UnSyncUnAssoc
SupportedAsynchronousActions	N	uint16[]	Identify replication methods using job control. Values: 2: Local Replica Creation 3: Remote Replica Creation 4: Local Replica Modification 5: Remote Replica Modification 6: Local Replica Attachment 7: Remote Replica Attachment 8: Buffer Creation
SupportedSynchronousActions	N	uint16[]	Identify replication methods not using job control. Values: 2: Local Replica Creation 3: Remote Replica Creation 4: Local Replica Modification 5: Remote Replica Modification 6: Local Replica Attachment 7: Remote Replica Attachment 8: Buffer Creation 9: NetworkPipe Creation
InitialReplicationState		uint16	The initial SyncState when replica creation is completed. Values: 2: Initialized 3: Prepared 4: Synchronized 5: Idle

Table 1088: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Property	Flags	Type	Description & Notes
SupportedModifyOperations		uint16[]	Identify ModifySynchronization operations supported for this CopyType. Values: 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		uint16	Host access restrictions. Values: 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
HostAccessibleState		uint16[]	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
PersistentReplicasSupported		boolean	true: replicas persist during power off or reset false: not persistent. May be in invalid state after power off or reset
MaximumReplicasPerSource		uint16	Maximum replicas of all types allowed for one source element.
MaximumLocalReplicationDepth		uint16	Volume A mirrors Volume B mirrors Volume C to this maximum allowable depth.

Table 1088: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Property	Flags	Type	Description & Notes
Optional Properties/Methods			
SupportedSpecializedElements		uint16[]	Identify specialized element types supported. Values: 2: Delta replica pool 3: Delta pool component extent 4: Remote mirror 5: Local mirror 6: Full snapshot 7: Delta snapshot 8: Replication buffer
SpaceLimitSupported		boolean	Only valid for CopyType "UnSyncAssoc-Delta": true: space limits are enforced false: not enforced
SpaceReservationSupported		boolean	Only valid for CopyType "UnSyncAssoc-Delta": true: space reserved at creation false: no space reserved at creation
LocalMirrorSnapshotSupported		boolean	Only valid for CopyType "Sync" and "Async": true: local mirror replicas can be snapshot source element false: local mirrors cannot be snapshot source
RemoteMirrorSnapshotSupported		boolean	Only valid for CopyType "Sync" and "Async": true: remote mirror replicas can be snapshot source element false: remote mirrors cannot be snapshot source
IncrementalDeltasSupported		boolean	Only valid for CopyType "UnSyncAssoc-Delta": true: all delta replicas associated with a source element are incrementally dependent. Only the oldest replica may be deleted or resynced false: not dependent
BidirectionalConnectionsSupported		boolean	Only valid for CopyType "Sync" and "Async" remote replicas: true: connections are bi-directional false: connections are uni-directional
MaximumPortsPerConnection		uint16	Maximum ports assigned to one peer-to-peer connection.
MaximumConnectionsPerPort		uint16	Maximum peer-to-peer connections for one port.
MaximumPeerConnections		uint16	Maximum peer-to-peer connections for one system element.

Table 1088: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Property	Flags	Type	Description & Notes
MaximumRemoteReplication-Depth		uint16	
InitialSynchronizationDefault		uint16	Refer to CIM_StorageSetting.InitialSyn-chronization
ReplicationPriorityDefault		uint16	Refer to CIM_StorageSetting.Replica-tionPriority
LowSpaceWarningThresholdDe-fault		uint8	Default value for LowSpaceWarningTh-reshold. Percentage value between 0 and 100.
SpaceLimitWarningThresholdDe-fault		uint8	Default value for SpaceLimitWarn-ingThreshold. Percentage value between 0 and 100.
RemoteReplicationService-PointAccess		uint16	Primary SAP for a remote replication service. Values: 2: Not specified 3: Source hosting system 4: Target hosting system 5: Proxy service
AlternateReplicationService-PointAccess		uint16	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
DeltaReplicaPoolAccess		uint16	Indicates if a specialized pool is required as a container for delta repli-cas. Values: 2: Any pool may contain delta replicas 3: Exclusive special pool per source element 4: Shared special pool for all source elements
RemoteBufferElementType		uint16	Indicates target container element type for remote replication buffers. Values: 2: Not specified by client 3: Storage pool 4: Storage extent
RemoteBufferHost		uint16	Indicates hosting element for replica-tion buffers. Values: 2: Associated to top-level system ele-ment 3: Associated to a ComponentCS sys-tem element 4: Associated to a NetworkPipe ele-ment

Table 1088: SMI Referenced Properties/Methods for CIM_StorageReplicationCapabilities

Property	Flags	Type	Description & Notes
RemoteBufferLocation		uint16	Indicates location for replication buffers. Values: 2: Source systems only 3: Target systems only 4: Both source and target require a buffer
RemoteBufferSupported		uint16	Indicates if a peer-to-peer connection may use a remote replication buffer. Values: 0: Not supported 2: Required 3: Optional
UseReplicationBufferDefault		uint16	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
PeerConnectionProtocol		string	Opaque string identifying the peer-to-peer transport protocol supported by the hosting system.

8.2.8.12.9.23 CIM_StorageSetting

Base definition is in Block Services Package.
Class Mandatory: true

Table 1089: SMI Referenced Properties/Methods for CIM_StorageSetting

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
DeltaReservationMin	M	uint8	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	uint8	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	uint8	Goal for space reserved for a delta replica at time of creation. Value 0 to 100 is a percentage of the source element size.
PersistentReplica	M	boolean	True indicates the replica persists during power off or reset.

Table 1089: SMI Referenced Properties/Methods for CIM_StorageSetting

Property	Flags	Type	Description & Notes
IntendedUsage	M	uint16	Indicates that a storage element is created for a specialized use. Values: 0: Not specialized 2: Special pool for delta replica elements 3: Component extent for delta replica pool 4: Remote mirror target element 5: Local mirror target element 6: Full size snapshot element 7: Delta snapshot element 8: Remote replication buffer element
Optional Properties/Methods			
IncrementalDeltas	M	boolean	True indicates delta replicas for source are incrementally dependent in the order created.
UseReplicationBuffer	M	uint16	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	uint16	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
ReplicationPriority	M	uint16	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

8.2.8.12.9.24 CIM_StorageSynchronized

Associates replica target element to a source element.
Created By : Extrinsic(s): CreateReplica, AttachReplica, AttachOrModifyReplica
Modified By : Extrinsic(s): ModifySynchronization
Deleted By : Extrinsic(s): ModifySynchronization
Class Mandatory: true

Table 1090: SMI Referenced Properties/Methods for CIM_StorageSynchronized

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemElement		CIM_ManagedElement	Replica source element.

Table 1090: SMI Referenced Properties/Methods for CIM_StorageSynchronized

Property	Flags	Type	Description & Notes
SyncedElement		CIM_ManagedElement	Replica target element.
WhenSynced	N	datetime	If the replica is a PIT image, this value is the date/time created.
SyncMaintained		boolean	Boolean indicating whether synchronization is maintained.
CopyType		uint16	Type of association between source and target. Values: 2: Sync 3: Async 4: UnSyncAssoc 5: UnSyncUnAssoc
SyncState		uint16	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
Optional Properties/Methods			
ReplicaType		uint16	Informational property describing the type of replication. Values: 0: Not specified 2: Full Copy 3: Before Delta 4: After Delta 5: Log
CopyPriority	M	uint16	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

8.2.8.12.9.25 CIM_StorageVolume

A StorageVolume element that is an associated replica source or target requires a StorageSynchronized association. The StorageSetting element associated to a replica target may use several additional properties not applicable to an independent storage element. If the element is a remote replica target, an AttachedElement association may be used to stitch the element to the top-level pipe of a managed connection.

Created By : Extrinsic(s): Target may be created by CIM_StorageConfigurationService.CreateReplica

Class Mandatory: true

Table 1091: SMI Referenced Properties/Methods for CIM_StorageVolume (Array Profile)

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Optional Properties/Methods			
DeltaReservation		uint8	Actual space reserved percentage if the created element is a delta replica . Value 0 to 100 is a percentage of the source element size.

8.2.8.12.9.26 CIM_SystemComponent

Associates peer system element to remote replication network domain.
Class Mandatory: false

Table 1092: SMI Referenced Properties/Methods for CIM_SystemComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_ManagedSystemElement	

8.2.8.12.10 Related Standards

Table 1093: Related Standards for Copy Services

Specification	Revision	Organization
CIM Infrastructure	2.3	DMTF
CIM Operations over HTTP	1.2	DMTF
CIM Schema	2.11.0	DMTF

DEPRECATED

8.2.8.13 Disk Drive Subprofile (DEPRECATED)

The functionality of the Disk Drive Subprofile has been subsumed by the 8.2.8.14, "Disk Drive Lite Subprofile".

The Disk Drive Subprofile is defined in IIS24775-2006, *Storage Managemen*t.

DEPRECATED

8.2.8.14 Disk Drive Lite Subprofile

8.2.8.14.1 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). 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.

8.2.8.14.1.1 Base model

A disk drive is modeled as a MediaAccessDevice (DiskDrive) linked to a 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 is connected by a Realizes association. The model can optionally contain SoftwareIdentity that contains information about the firmware and is linked by a DeviceSoftware association.

8.2.8.14.1.2 Associations to external classes

The Disk Drive subprofile ties into the rest of the system via a number of key associations.

- ConcreteComponent - To associate an extent exported by the Disk Drive to the StoragePool that it is part of.
- BasedOn - To associate an extent exported by the Disk Drive to another (higher level) extent (or a Volume).
- Container - To associate the physical package of the disk drive to the physical package of the system.
- SystemDevice - To scope the Disk to the system containing it.
- ProtocolControllerAccessUnit - To link the Disk to system port it is accessed through.

8.2.8.14.1.3 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

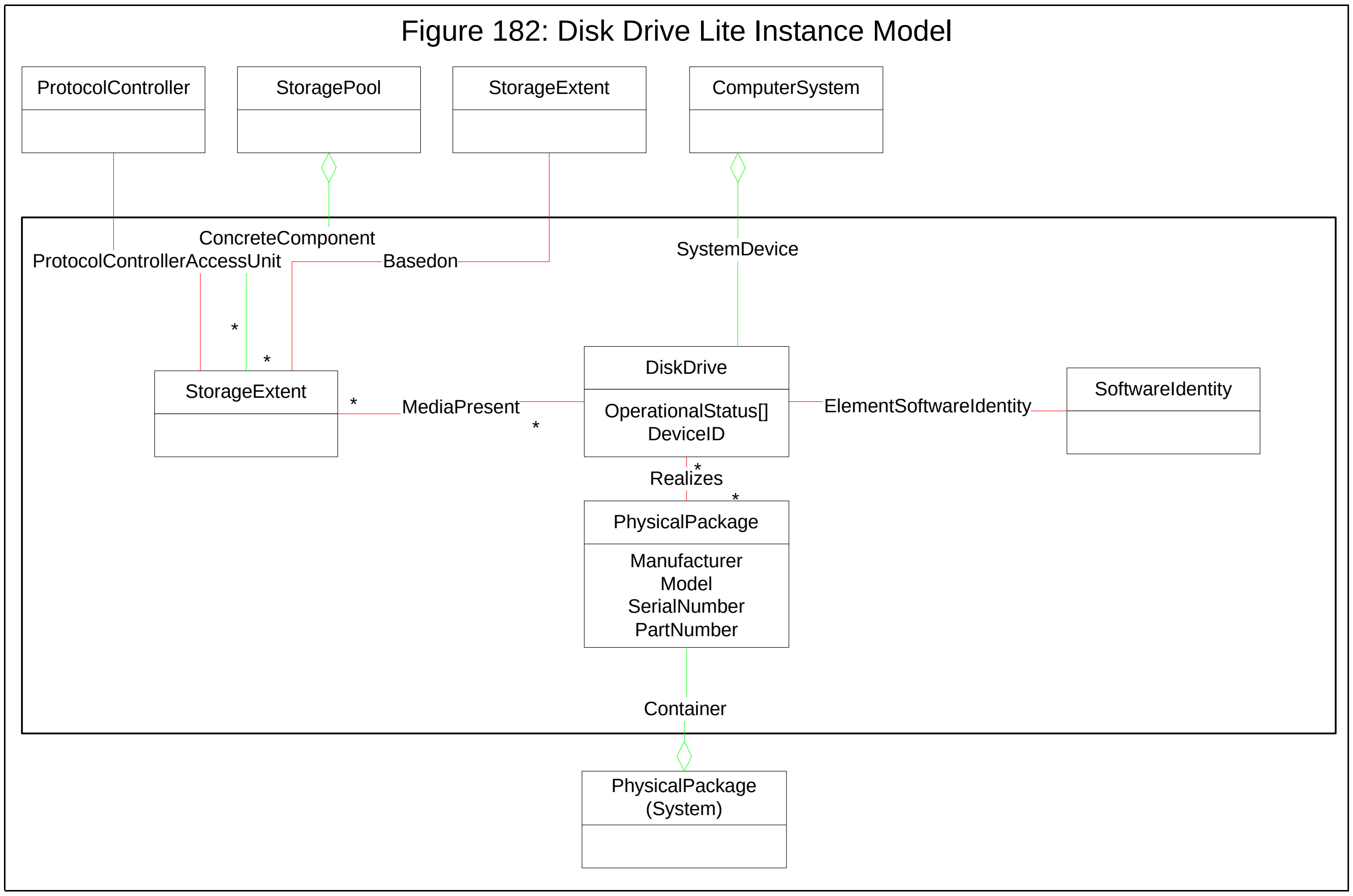


Figure 182: "Disk Drive Lite Instance Model" illustrates a sample instance diagram.

Durable Names and Correlatable IDs of the Profile

None.

Required External Associations

When implementing the Disk Drive subprofile, the ConcreteComponent association to StoragePools is mandatory (because LogicalStorage is required in the Profile).

The Container association to a higher level physical package is also mandatory (because the PhysicalPackage for the System is required). However, in the case of the Container association, it is possible that the Disk Drive PhysicalPackage is not directly contained in the System PhysicalPackage. It shall be possible for a client to traverse the container associations from the System PhysicalPackage to the Disk Drive PhysicalPackage, even if the client is required to go through intermediate steps (that is, intermediate physical packages).

The ProductParentChild association from the disk drive product to the higher level product is also mandatory. It is not necessary for the System Product to be the next level up the ProductParentChild association, but it may be.

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.Health and Fault Management

8.2.8.14.2 Health and Fault Management Considerations

The DiskDrive.OperationalStatus contains the overall status of the disk, summarized in Table 1094, “OperationalStatus for a Disk”.

Table 1094: OperationalStatus for a Disk

OperationalStatus	Description
OK	Disk Drive is on line
Error	Disk Drive is no longer functioning
Stopped	Disk Drive is disabled
Stopping	Disk Drive is being disabled
Starting	Disk Drive is becoming enabled
Predictive Failure	Disk Drive is functionality nominally but is predicting a failure in the near future

If the reason for StorageVolume's or LogicalDisks' state/status change from normal operating conditions is in the form of a lack of performance or redundancy qualities caused by a disk drive failure, then the Volume OperationalStatus shall be expressed as in Table 1095, “Volume-Level OperationalStatus”.

Table 1095: Volume-Level OperationalStatus

SV/LD* OperationalStatus	SV/LD ExtentStatus	DD OperationalStatus
Degraded		One or more DiskDrives may have the "Error" Operational-Status
Degraded	Rebuild	One or more DiskDrives may have the "Error" Operational-Status
Degraded	Spare In Use	One or more DiskDrives has the "Error" OperationalStatus
Error		One or more DiskDrives may have the "Error" Operational-Status
Error	Broken	One or more DiskDrives may have the "Error" Operational-Status
Error	Data Loss	One or more DiskDrives may have the "Error" Operational-Status

*SV - StorageVolume, LD - LogicalDisk

8.2.8.14.3 Cascading Considerations

Not defined in this standard.

8.2.8.14.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.8.14.5 Methods of this Profile

Not defined in this standard.

8.2.8.14.6 Registered Name and Version

Disk Drive Lite version 1.1.0

8.2.8.14.7 CIM Server Requirements

Table 1096: CIM Server Requirements for Disk Drive Lite

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.14.8 CIM Elements

Table 1097: CIM Elements for Disk Drive Lite

Element Name	Description
Mandatory Classes	
CIM_DiskDrive (8.2.8.14.8.1)	
CIM_ElementSoftwareIdentity (8.2.8.14.8.2)	
CIM_MediaPresent (8.2.8.14.8.3)	
CIM_PhysicalPackage (8.2.8.14.8.4)	
CIM_Realizes (8.2.8.14.8.5)	
CIM_SoftwareIdentity (8.2.8.14.8.6)	
CIM_StorageExtent (8.2.8.14.8.7)	
CIM_SystemDevice (8.2.8.14.8.8)	

8.2.8.14.8.1 CIM_DiskDrive

Class Mandatory: true

Table 1098: SMI Referenced Properties/Methods for CIM_DiskDrive

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
Name		string	
OperationalStatus		uint16[]	

8.2.8.14.8.2 CIM_ElementSoftwareIdentity

Class Mandatory: true

Table 1099: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SoftwareIdentity	
Dependent		CIM_ManagedElement	

8.2.8.14.8.3 CIM_MediaPresent

Class Mandatory: true

Table 1100: SMI Referenced Properties/Methods for CIM_MediaPresent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_MediaAccessDevice	
Dependent		CIM_StorageExtent	

8.2.8.14.8.4 CIM_PhysicalPackage

Class Mandatory: true

Table 1101: SMI Referenced Properties/Methods for CIM_PhysicalPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
Manufacturer		string	
Model		string	
Optional Properties/Methods			
SerialNumber		string	
PartNumber		string	

8.2.8.14.8.5 CIM_Realizes

Class Mandatory: true

Table 1102: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	
Dependent		CIM_LogicalDevice	

8.2.8.14.8.6 CIM_SoftwareIdentity

Class Mandatory: true

Table 1103: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	
Optional Properties/Methods			
Manufacturer		string	
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	
MinorVersion		uint16	

8.2.8.14.8.7 CIM_StorageExtent

Class Mandatory: true

Table 1104: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
BlockSize		uint64	
NumberOfBlocks		uint64	The number of blocks as reported by the hardware.
ConsumableBlocks		uint64	The number of usable blocks.
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	

8.2.8.14.8.8 CIM_SystemDevice

Class Mandatory: true

Table 1105: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	
PartComponent		CIM_LogicalDevice	

8.2.8.14.9 Related Standards

Table 1106: Related Standards for Disk Drive Lite

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

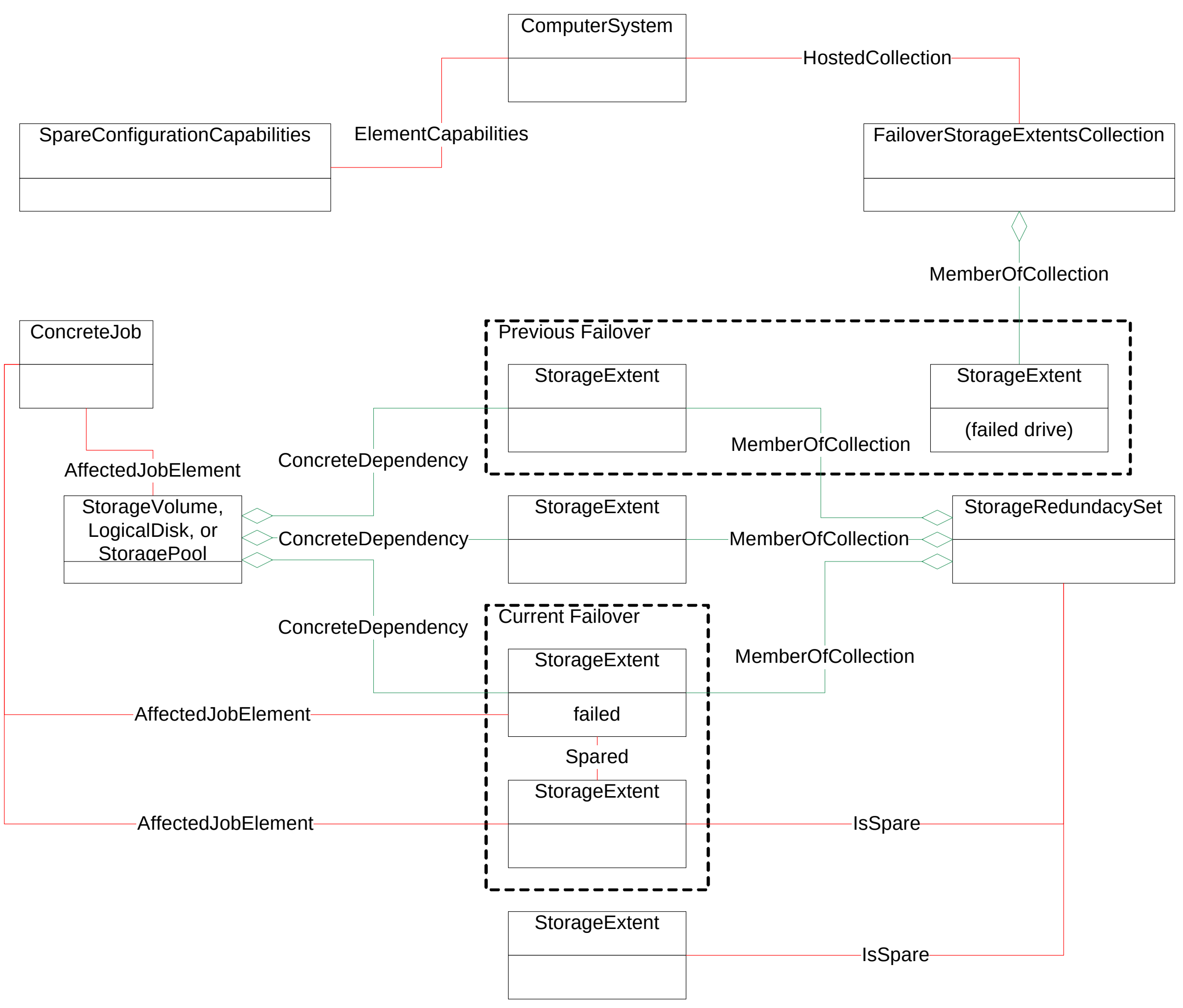
8.2.8.15 Disk Sparing Subprofile

8.2.8.15.1 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 183: "Sparing Instance Diagram".

Figure 183: Sparing Instance Diagram



The Extent Composition Subprofile (8.2.8.16) 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.

This Disk Drive Lite Subprofile (8.2.8.14) 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 the Health Package (8.2.1.6) 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.

<u>Durable Names and Correlatable IDs of the Profile</u>

The StorageVolumes are required to provide the correlatable ID, Name. See 6.2.4.2, "Guidelines for SCSI Logical Unit Names".

<u>Sparing Model</u>

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

Note: For this version of SMI-S, all control and configuration operations and the grouping of failed and unassigned extents will be EXPERIMENTAL. Each of these functionality require the use of EXPERIMENTAL SNIA_ classes.

StorageVolume and LogicalDisk is used to represent that capacity exposed to clients of the Block Server.

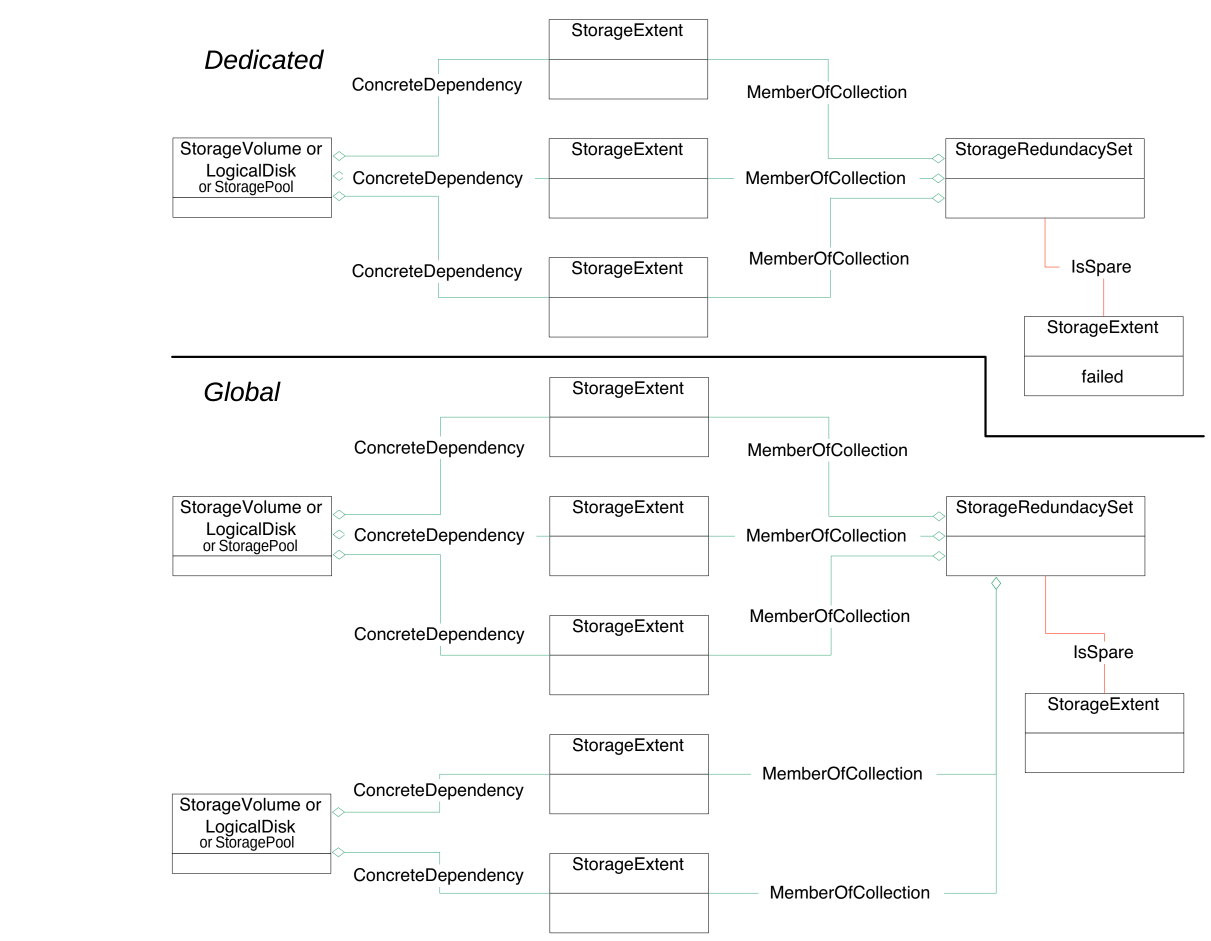
StoragePool is used to represent that grouping of capacity for later allocation to StorageVolumes or LogicalDisks.

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 StorageRedudancySet. 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 184: "Variations of RS per Storage Element".

Figure 184: 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 8.2.8.10, "Block Services Package" and this subprofile, would match this Goal with StorageRedundancySet that had at least that number of spares.

The “CIM_LogicalDisk” 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.

The “CIM_HostedCollection” is used 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 1107, “Supported Methods to Method

Mapping" describes how to map the functionality names in the SupportedSynchronousActions and SupportedAsynchronousActions arrays.

Table 1107: Supported Methods to Method Mapping

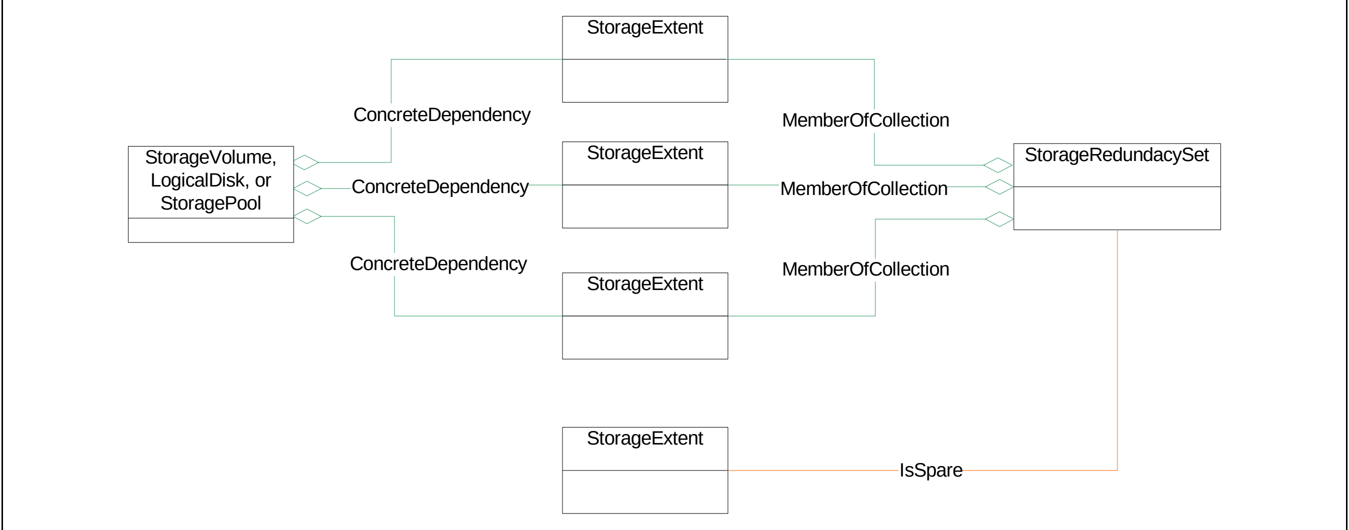
Action	Method
Assign Spares	SpareConfigurationService.AssignSpares
Unassign Spares	SpareConfigurationService.UnassignSpares
Rebuild Storage Extent	SpareConfigurationService.RebuildStorageExtent
Check Parity Consistency	SpareConfigurationService.CheckParityConsistency
Repair Parity	SpareConfigurationService.RepairParity
Fail Over	StorageRedundancySet.Failover

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.

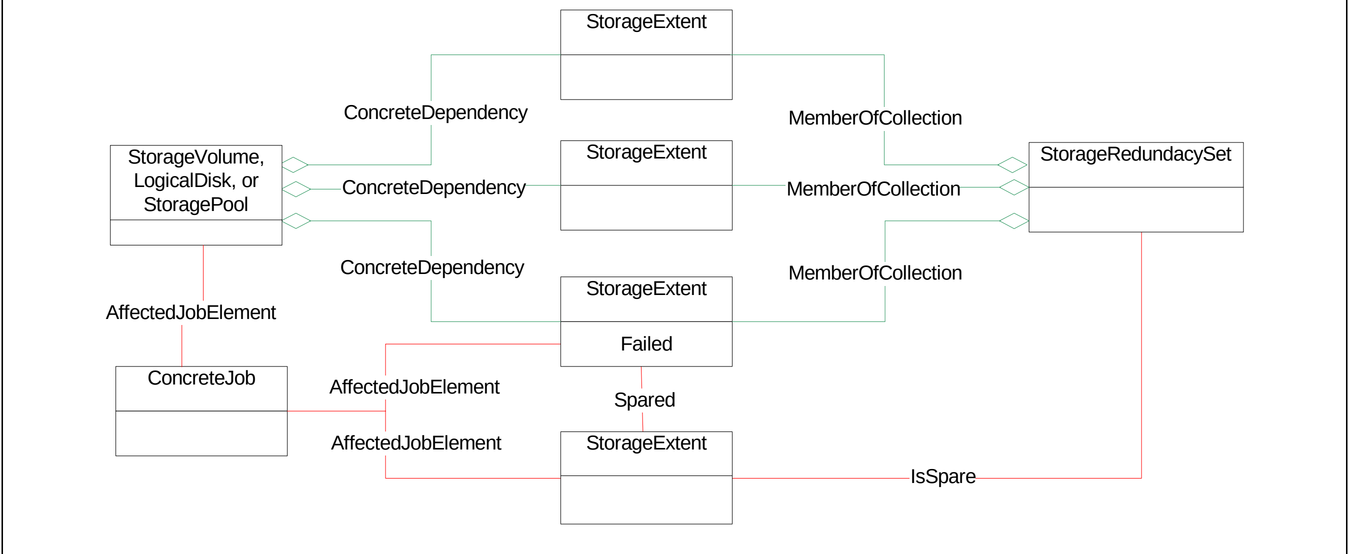
Figure 185: "Before Failure" shows a dedicated RedundancySet with a single spare.

Figure 185: Before Failure



Once the failure has occurred, a ConcreteJob is created to represent the fail over process, as shown in Figure 186: "During Failure".

Figure 186: During Failure

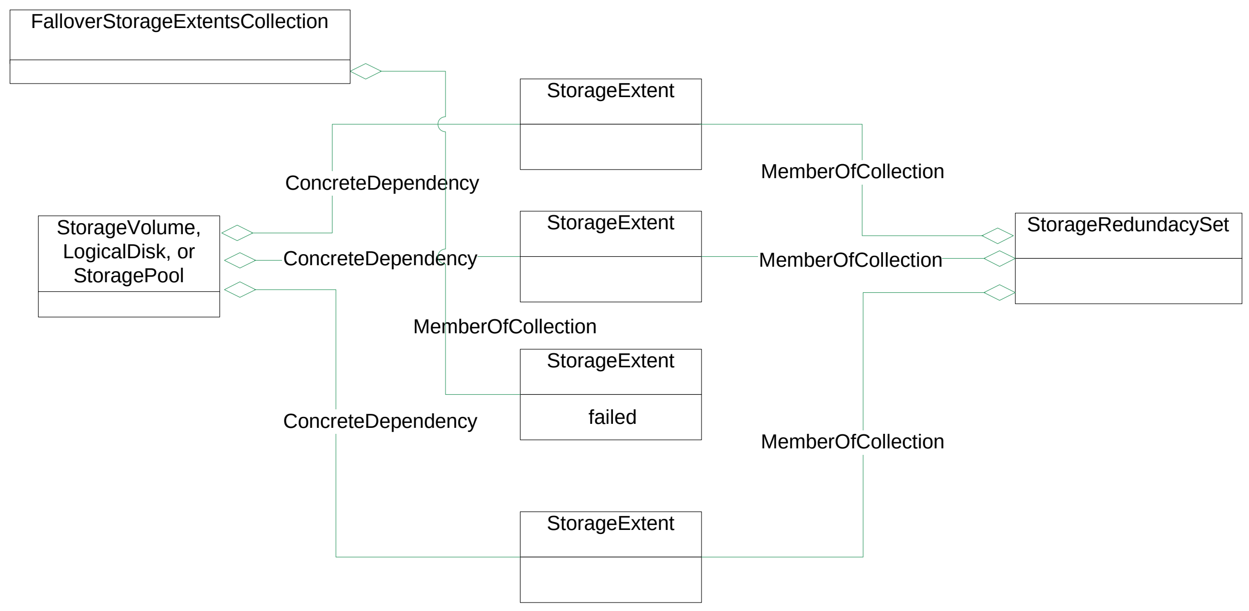


The AffectJobElement association shall associate the LogicalDisk or StorageVolume that is being failed over, the StorageExtent that has failed and is causing the fail over, and the spare StorageExtent. The associations shall remain for some period of time as per the rules in the 8.2.1.7, "Job Control 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 8.2.1.7, "Job Control Subprofile", shall contain the StorageRedundancySet.Failover() method name and parameters.

Once the fail over is complete, the failed Extent shall no longer have a ConcreteDependency 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 aConcreteDependency relationship with the spare Extent, as illustrated in Figure 187: "After Failure".

Figure 187: After Failure



EXPERIMENTAL

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 8.2.1.7, "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.

8.2.8.15.2 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.

8.2.8.15.3 Cascading Conjurations

Not defined in this standard.

8.2.8.15.4 Supported Subprofiles and Packages

Not defined in this standard.

8.2.8.15.5 Methods of the Profile

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 spared 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 primorial 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 a normal return codes., see 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration" 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 than 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 passes are already in use as a spare or as part of a StorageVolume or LogicalDisk.

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 may end up as a member of the FailoverStorageExtentsCollection. The rules for the parameters and the same descriptions of AssignSpares are true for the parameters and return codes shared between the two method definitions. This method shall not return vendor specific return codes.

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 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration" 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.

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 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration" 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.

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 8.2.8.10.5.3, "Extrinsic Methods on StorageConfiguration" 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.

EXPERIMENTAL

8.2.8.15.6 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 Table 1118:, "SMI Referenced Properties/Methods for CIM_StorageRedundancySet"for details.

8.2.8.15.6.1 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 occured
         before this point in the code.>
```

8.2.8.15.7 Registered Name and Version

Disk Sparing version 1.1.0

8.2.8.15.8 CIM Server Requirements

Table 1108: CIM Server Requirements for Disk Sparing

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.15.9 CIM Elements

Table 1109: CIM Elements for Disk Sparing

Element Name	Description
Mandatory Classes	
CIM_ConcreteDependency (8.2.8.15.9.1)	
CIM_IsSpare (8.2.8.15.9.3)	Represents the spare that may be used as a spare for any StorageExtents that is not a spare.
CIM_LogicalDisk (8.2.8.15.9.4)	
CIM_MemberOfCollection (8.2.8.15.9.5)	
CIM_Spared (8.2.8.15.9.6)	Represents the relationship between the spare and the StorageExtent that has failed and is being spared
CIM_StorageExtent (8.2.8.15.9.7)	
CIM_StoragePool (8.2.8.15.9.8)	Common elements to Primordial and Concrete Pools.
CIM_StorageRedundancySet (8.2.8.15.9.9)	
CIM_StorageVolume (8.2.8.15.9.10)	Commonly known as a LUN but without the semantics of mapping to a host (which is covered by Masking and Mapping).
Optional Classes	
CIM_HostedCollection (8.2.8.15.9.2)	Associates FailoverStorageExtentsCollection with the Block Server's ComputerSystem instance.
SNIA_FailoverStorageExtentsCollection (8.2.8.15.9.11)	The collection of StorageExtents that have failed or are not yet assigned to a Primordial StoragePool.
SNIA_SpareConfigurationCapabilities (8.2.8.15.9.12)	Instances of this class define the behavior supported by this sparing implementation.
SNIA_SpareConfigurationService (8.2.8.15.9.13)	Control the spares configuration and control of Storage-Extent data and parity consistency.

8.2.8.15.9.1 CIM_ConcreteDependency

Class Mandatory: true

Table 1110: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	The spare StorageExtent.
Dependent		CIM_ManagedElement	The storage element being spared.

8.2.8.15.9.2 CIM_HostedCollection

Associates FailoverStorageExtentsCollection with the Block Server's ComputerSystem instance.
Class Mandatory: false

Table 1111: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	

Table 1111: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Dependent		CIM_SystemSpecificCollection	

8.2.8.15.9.3 CIM_IsSpare

Represents the spare that may be used as a spare for any StorageExtents that is not a spare.
Class Mandatory: true

Table 1112: SMI Referenced Properties/Methods for CIM_IsSpare

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	
Dependent		CIM_RedundancySet	
SpareStatus		uint16	
FailoverSupported		uint16	

8.2.8.15.9.4 CIM_LogicalDisk

Class Mandatory: true

Table 1113: SMI Referenced Properties/Methods for CIM_LogicalDisk

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
Name		string	OS Device Name
NameFormat		uint16	Format for name
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
BlockSize		uint64	
NumberOfBlocks		uint64	The number of blocks that make of this LogicalDisk.
IsBasedOnUnderlyingRedundancy		boolean	
NoSinglePointOfFailure		boolean	
DataRedundancy		uint16	
PackageRedundancy		uint16	
DeltaReservation		uint8	
Optional Properties/Methods			
ElementName		string	User-friendly name

8.2.8.15.9.5 CIM_MemberOfCollection

Class Mandatory: true

Table 1114: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The StorageRedundancySet
Member		CIM_ManagedElement	The storage element being spared.

8.2.8.15.9.6 CIM_Spared

Represents the relationship between the spare and the StorageExtent that has failed and is being spared
Class Mandatory: true

Table 1115: SMI Referenced Properties/Methods for CIM_Spared

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	
Dependent		CIM_ManagedElement	

8.2.8.15.9.7 CIM_StorageExtent

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1116: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
HealthState		uint16	Reports the state of the StorageExtents underlying component.
OperationalStatus		uint16[]	Reports the operational status of the StorageExtent.

8.2.8.15.9.8 CIM_StoragePool

Common elements to Primordial and Concrete Pools.
Class Mandatory: true

Table 1117: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
ElementName		string	

Table 1117: SMI Referenced Properties/Methods for CIM_StoragePool

Property	Flags	Type	Description & Notes
PoolID		string	A unique name in the context of this system that identifies this Pool.

8.2.8.15.9.9 CIM_StorageRedundancySet

Class Mandatory: true

Table 1118: SMI Referenced Properties/Methods for CIM_StorageRedundancySet

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
RedundancyStatus		uint16	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		uint16[]	'Limited Sparing', 5, is the type of sparing supported in the subprofile
MinNumberNeeded		uint32	
MaxNumberSupported		uint32	
Failover()			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 a fail back to occur, if that also does not occur automatically.

8.2.8.15.9.10 CIM_StorageVolume

Commonly known as a LUN but without the semantics of mapping to a host (which is covered by Masking and Mapping).
Class Mandatory: true

Table 1119: SMI Referenced Properties/Methods for CIM_StorageVolume

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier

Table 1119: SMI Referenced Properties/Methods for CIM_StorageVolume

Property	Flags	Type	Description & Notes
Name		string	VPD 83 identifier for this volume (ideally a LUN WWN)
NameFormat		uint16	Format for name
ExtentStatus		uint16[]	
OperationalStatus		uint16[]	
Optional Properties/Methods			
ElementName		string	User-friendly name

8.2.8.15.9.11 SNIA_FailoverStorageExtentsCollection

The collection of StorageExtents that have failed or are not yet assigned to a Primordial StoragePool.
Class Mandatory: false
No specified properties or methods.

8.2.8.15.9.12 SNIA_SpareConfigurationCapabilities

Instances of this class define the behavior supported by this sparing implementation.
Class Mandatory: false

Table 1120: SMI Referenced Properties/Methods for SNIA_SpareConfigurationCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SupportedAsynchronousActions	N	uint16[]	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	uint16[]	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		boolean	Set to true if this storage system automatically configures spares. If set to false, the client shall use the extrinsic methods AssignSpares and UnassignSpares.
AutomaticFailOver		boolean	Set to true if this storage system automatically fails over. If set to false, the client shall use the FailOver extrinsic method, although that method may not be supported.

Table 1120: SMI Referenced Properties/Methods for SNIA_SpareConfigurationCapabilities

Property	Flags	Type	Description & Notes
MaximumSpareStorageExtents		uint16	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.

8.2.8.15.9.13 SNIA_SpareConfigurationService

Control the spares configuration and control of StorageExtent data and parity consistency.
Class Mandatory: false

Table 1121: SMI Referenced Properties/Methods for SNIA_SpareConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
AssignSpares()			
UnassignSpares()			
RebuildStorageExtent()			
CheckParityConsistency()			
RepairParity()			

8.2.8.15.10 Related Standards

Table 1122: Related Standards for Disk Sparing

Specification	Revision	Organization
CIM Infrastructure	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.2.8.16 Extent Composition Subprofile

8.2.8.16.1 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.

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.

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.

8.2.8.16.1.1 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 “Component Extents” in 8.2.8.16.1.2).

StoragePool and AllocatedStoragePool are shown in instance diagrams for context but are part of the Block Service package Read Only sub-package.

Refer to the section “Required CIM Elements” for detailed class descriptions.

8.2.8.16.1.2 Relation to other Packages and Subprofiles

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.

Component Extents

Component Extents of a pool are the most antecedent 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 “Block Services Extent Conservation” in 8.2.8.16.1.2).

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.

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.

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.

8.2.8.16.1.3 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.

Volume Composition

Figure 188: "Volume Composition from General QOS Pool" 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, can not be directly exposed. This Based on association does not represent composition or decomposition, but the main recipe (see 8.2.8.16.6.1) for this subprofile makes use of the decomposition function (i.e., complete consumption) to make this initial traversal.

Figure 188: Volume Composition from General QOS Pool

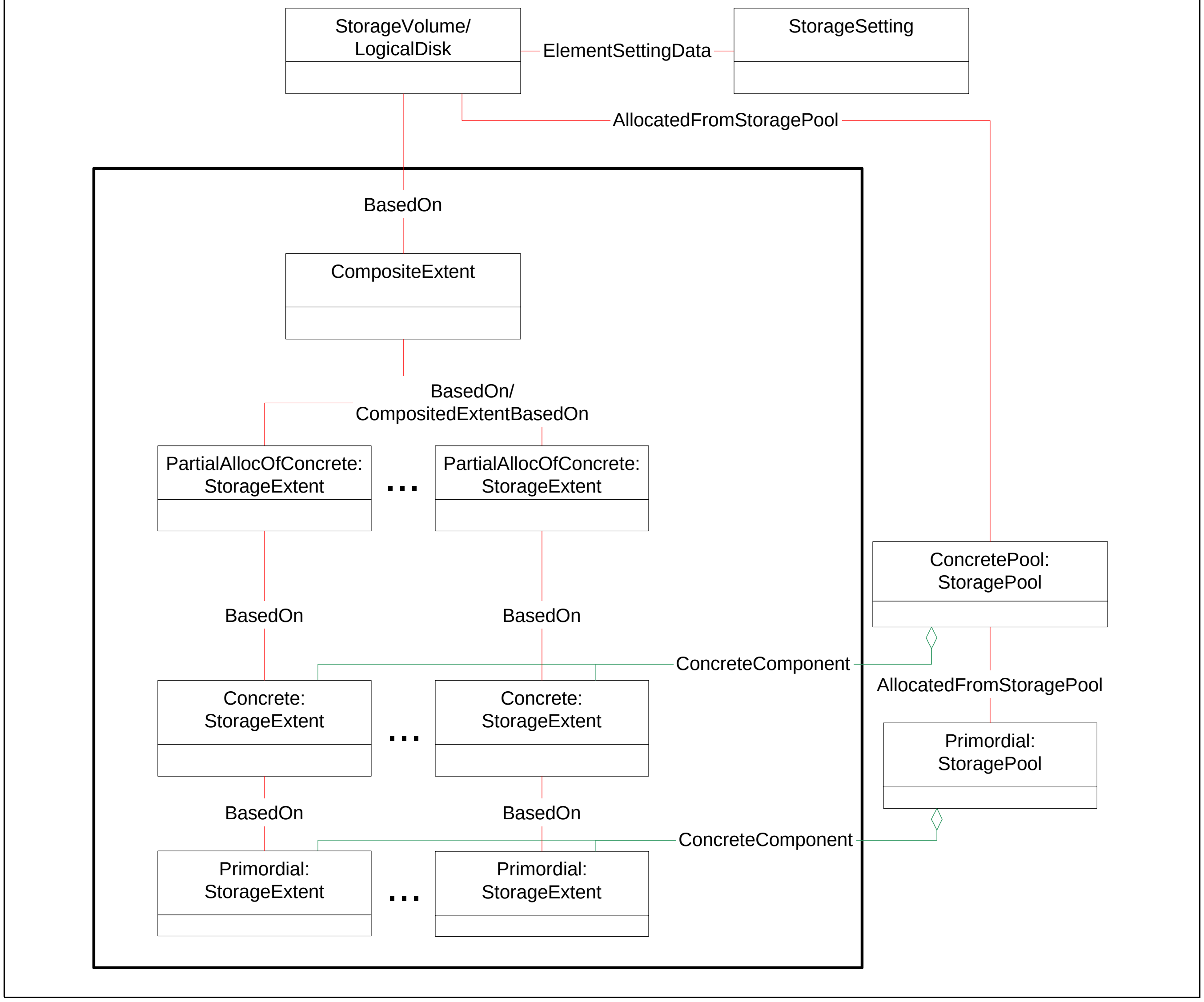


Figure 189: "Single QOS Pool Composition (RAID Groups)" shows a single composition (such as a RAID 5 or RAID 1). Not shown is the scenario where there may be two or more such back to back productions (such as a RAID 10). Also not shown is the scenario where the two productions may be in different concrete 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 the “Pool Composition” scenario 8.2.8.16.1.3)

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.

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 189: Single QOS Pool Composition (RAID Groups)

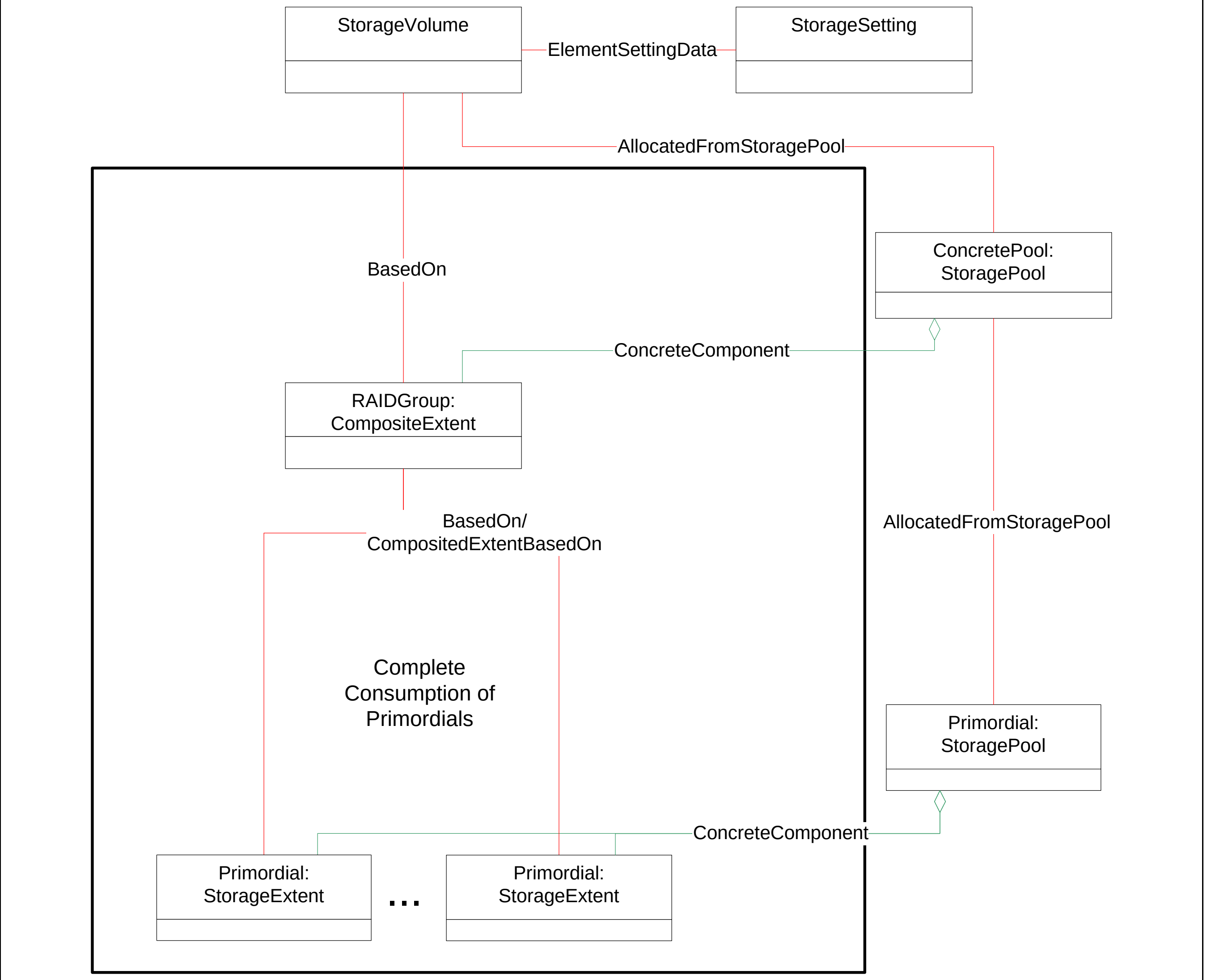
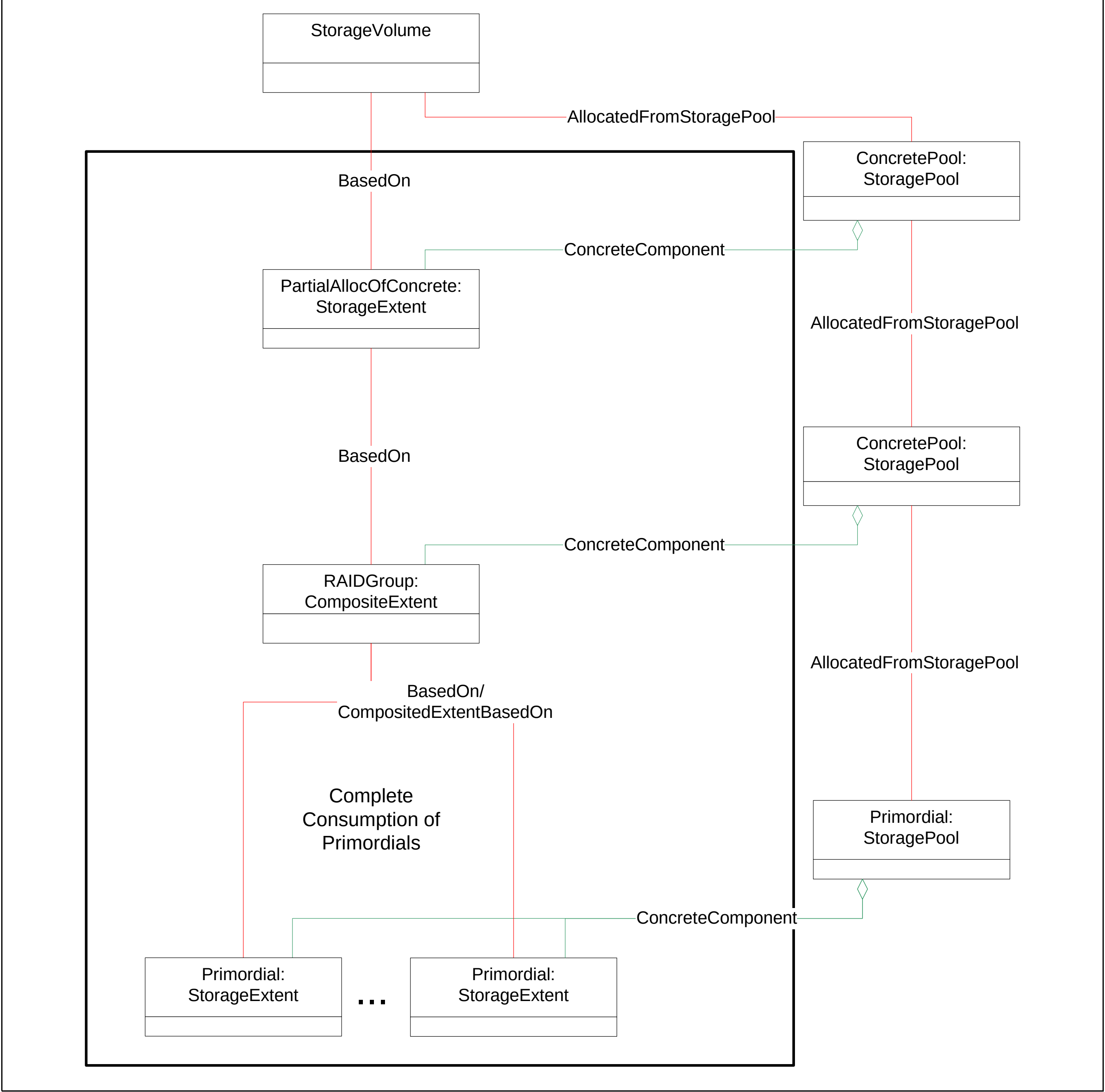

StorageVolume
ElementSettingData
StorageSetting
AllocatedFromStoragePool
BasedOn
ConcretePool: StoragePool
ConcreteComponent
RAIDGroup: CompositeExtent
BasedOn/ CompositedExtentBasedOn
AllocatedFromStoragePool
Complete Consumption of Primordials
Primordial: StoragePool
ConcreteComponent
Primordial: StorageExtent
...
Primordial: StorageExtent

Figure 190: "SIngle QOS Pool Composition - Two Concretes" 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.

Figure 190: SIngle QOS Pool Composition - Two Concretes


StorageVolume
AllocatedFromStoragePool
ConcretePool: StoragePool
BasedOn
ConcreteComponent
PartialAllocOfConcrete: StorageExtent
AllocatedFromStoragePool
BasedOn
ConcretePool: StoragePool
ConcreteComponent
RAIDGroup: CompositeExtent
AllocatedFromStoragePool
BasedOn/ CompositedExtentBasedOn
Complete Consumption of Primordials
Primordial: StoragePool
ConcreteComponent
Primordial: StorageExtent
...
Primordial: StorageExtent

Example RAID Compositions from Block Services

Table 1123, “Supported Common RAID Levels” is an abridged version of Table 1003, “RAID Mapping Table” in Block Services. Table 1003 describes the RAID Levels commonly used at the time this version of SMI-S was released. Table 1123 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 RADI 10, there is no intermediate decomposition modeled; each extent Antecedent to the top level CompositeExtent is itself a CompositeExtent.

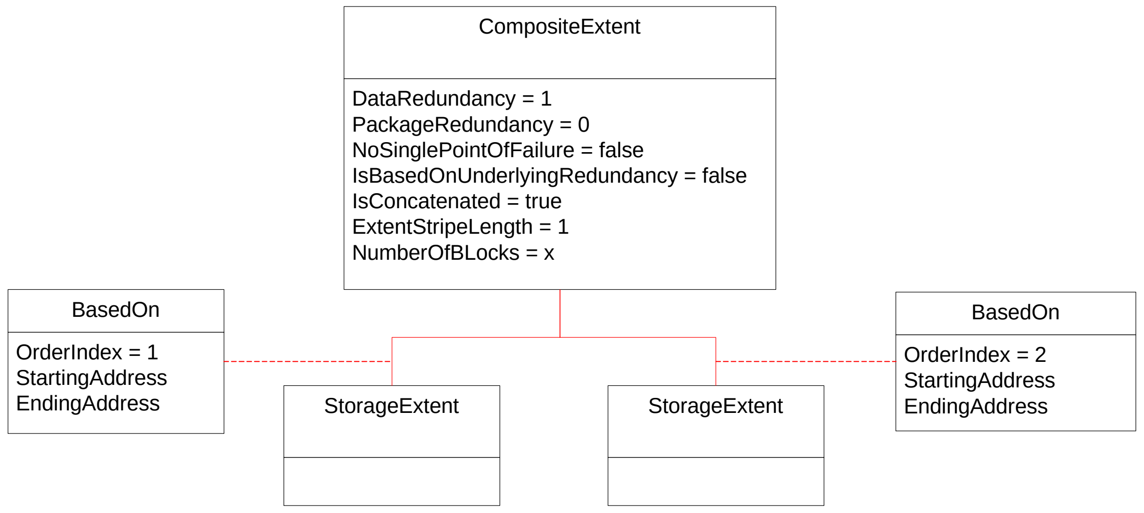
Table 1123: Supported Common RAID Levels

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

8.2.8.16.1.4 JBOD (Concatenation)

Figure 191: "Concatenation Composition" shows an partial instance diagram for a JBOD Volume or Pool, in which the Antecedent Extents are concatenated.

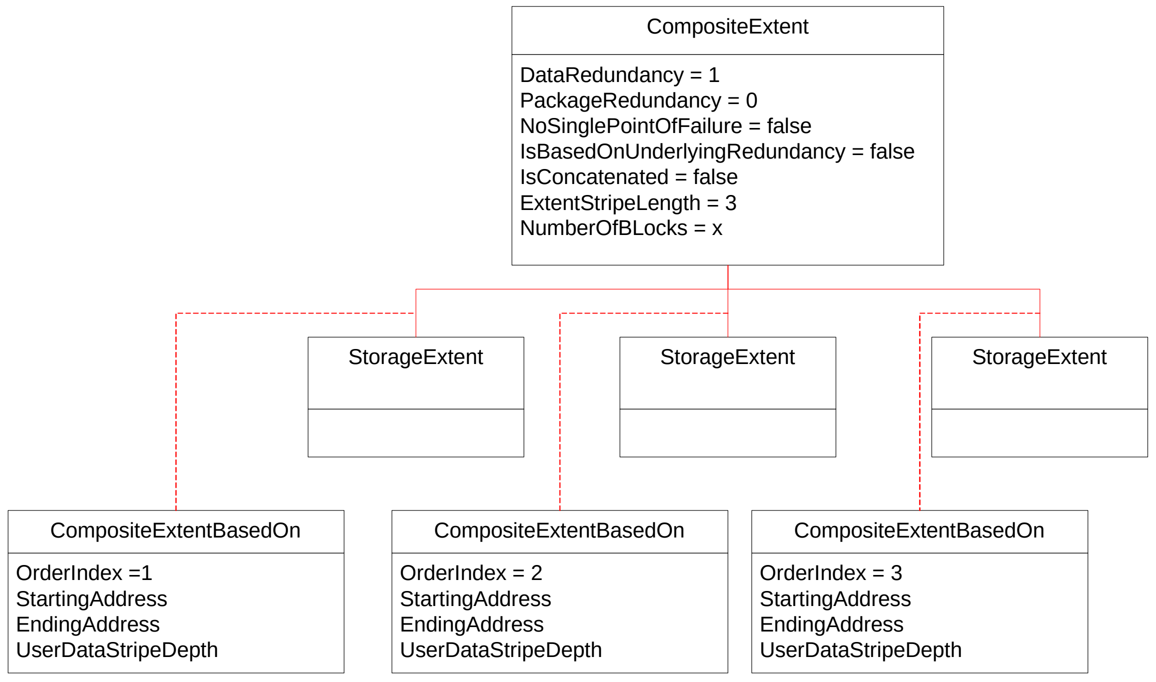
Figure 191: Concatenation Composition



8.2.8.16.1.5 RAID 0 (Striping)

Figure 192: "RAID 0 Composition" shows an partial instance diagram for a RAID 0 Volume or Pool.

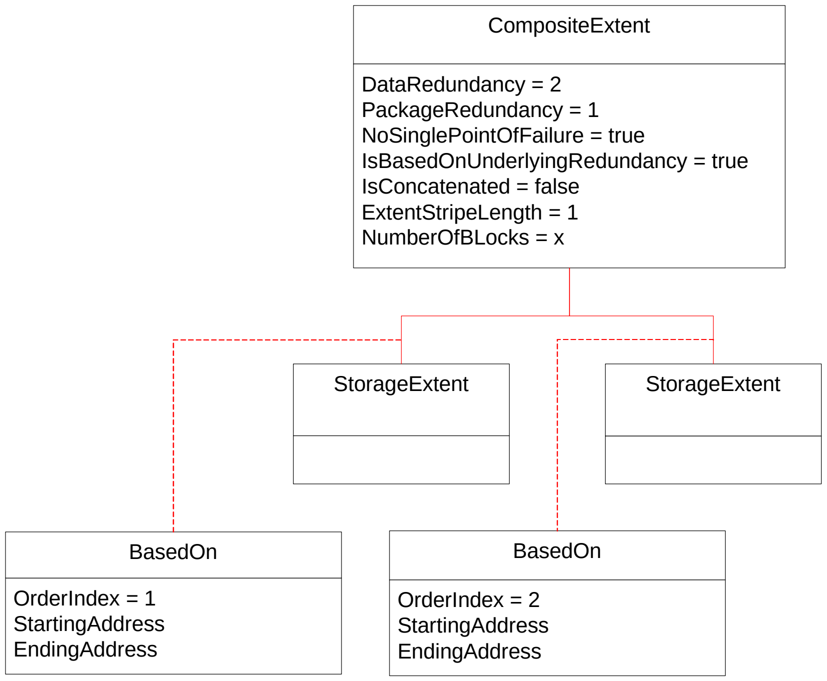
Figure 192: RAID 0 Composition



8.2.8.16.1.6 RAID 1

Figure 193: "RAID 1 Composition" shows an partial instance diagram for a RAID 1 Volume or Pool.

Figure 193: RAID 1 Composition

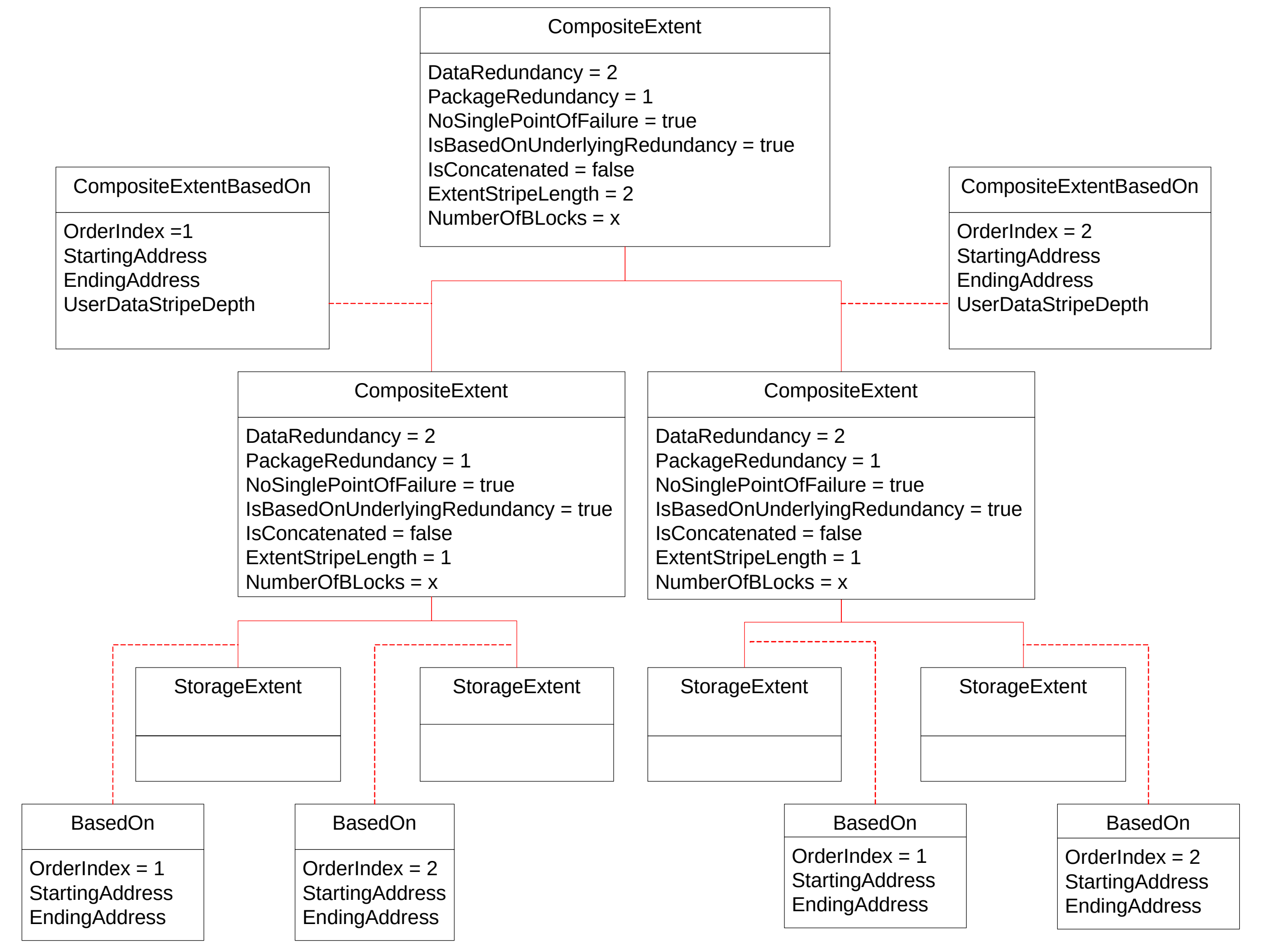


8.2.8.16.1.7 RAID 10

Figure 194: "RAID 10 Composition" 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.

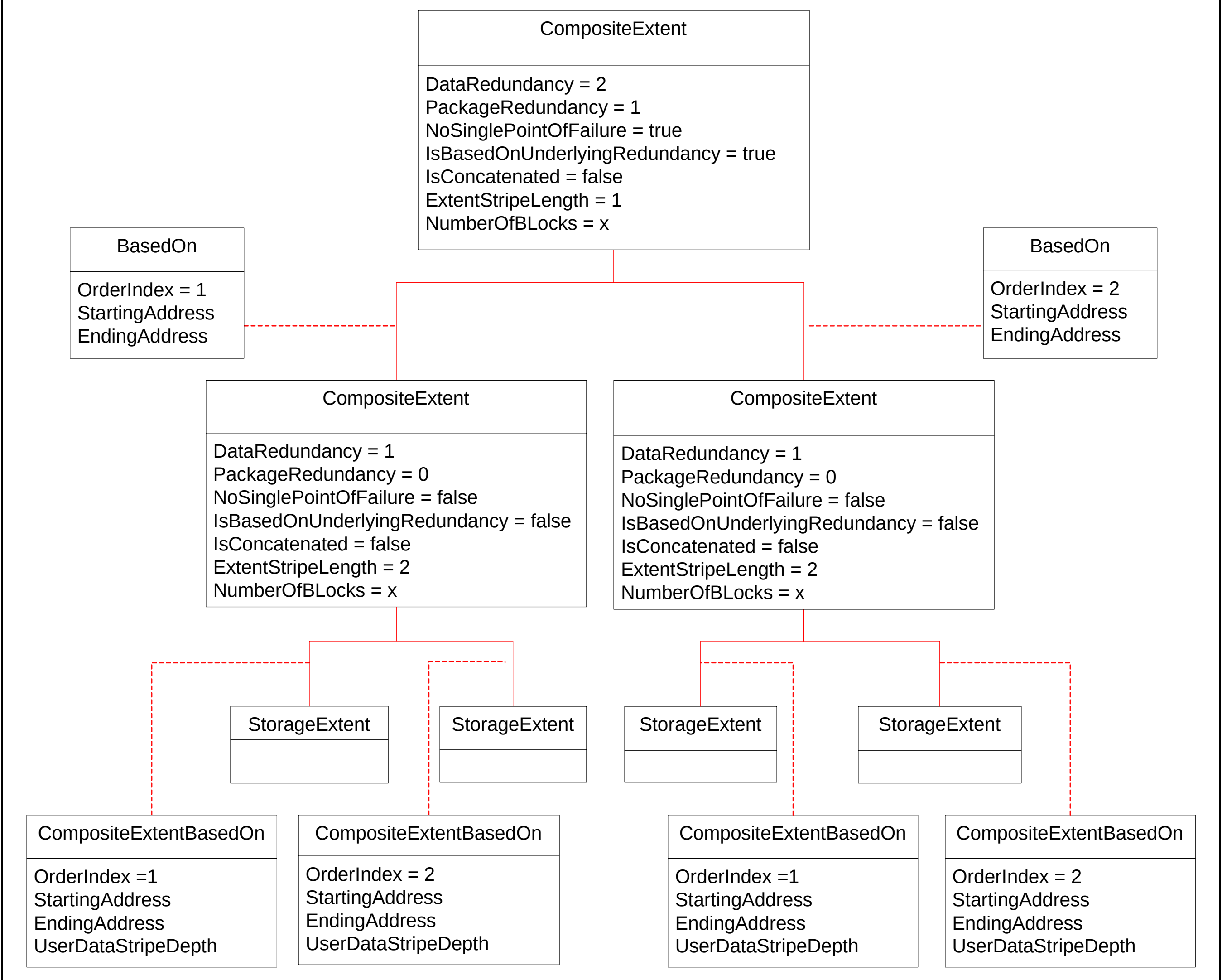
Figure 194: RAID 10 Composition



8.2.8.16.1.8 RAID 0+1

Figure 195: "RAID 0+1 Composition" shows an partial instance diagram for a RAID 0+1 Volume or Pool

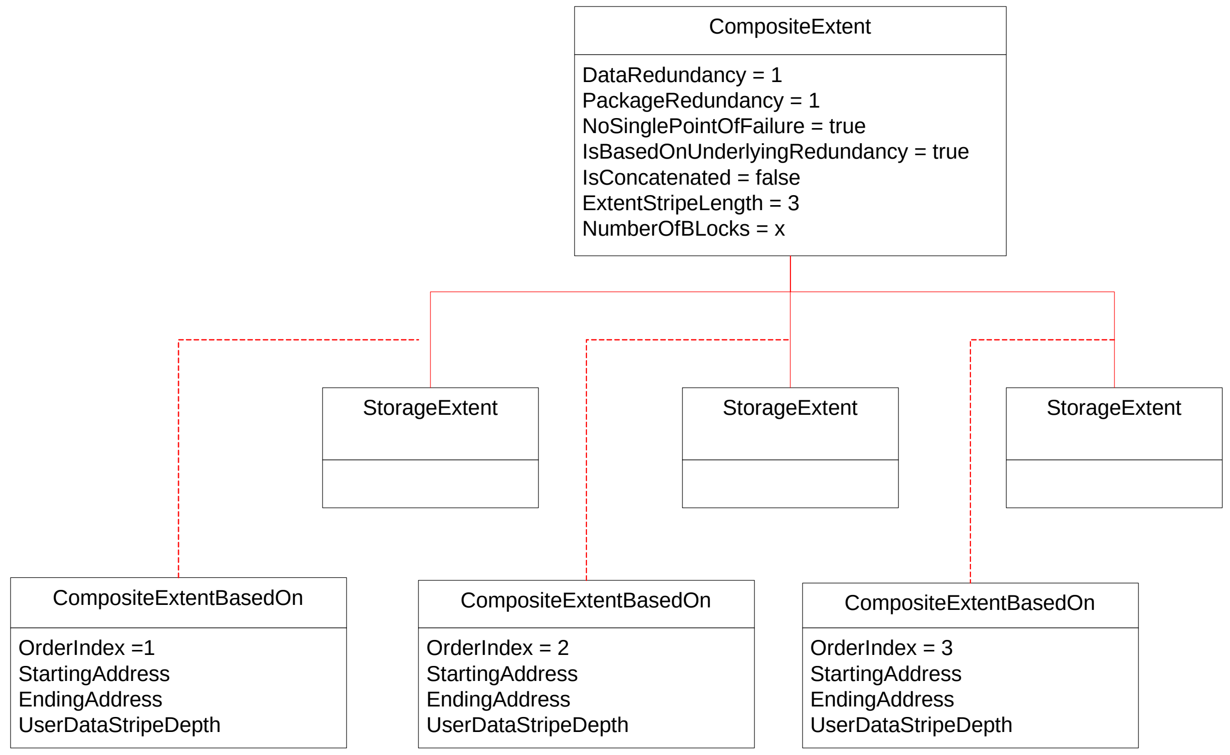
Figure 195: RAID 0+1 Composition



8.2.8.16.1.9 RAID 4 or 5

Figure 196: "RAID 4, 5 Composition" shows a partial instance diagram for a RAID 4 or 5 Volume or Pool.

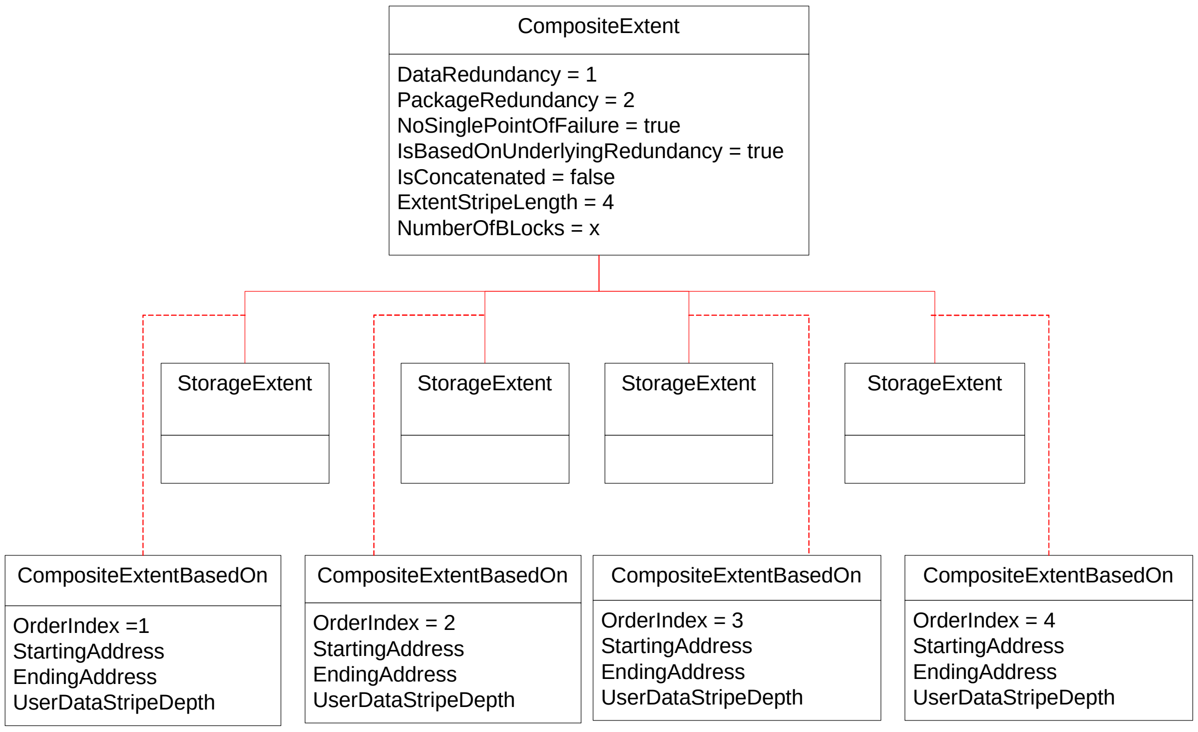
Figure 196: RAID 4, 5 Composition



8.2.8.16.1.10 RAID 6, 5DP, and 4DP

Figure 197: "RAID 6, 5DP, 4DP" shows a partial instance diagram for a RAID 6, 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.

Figure 197: RAID 6, 5DP, 4DP

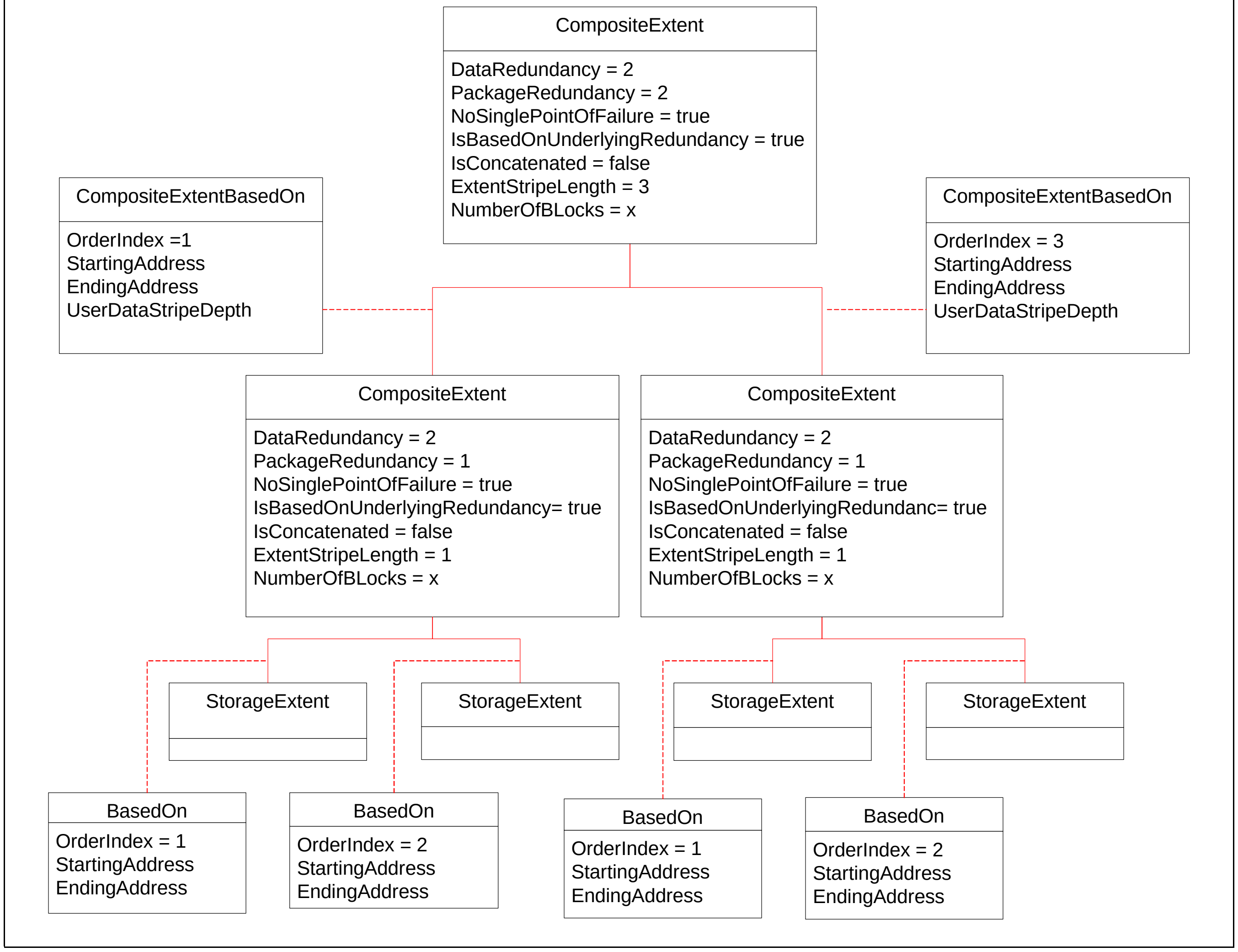


8.2.8.16.1.11 RAID 15

Figure 198: "RAID 15 Composition" 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.

Figure 198: RAID 15 Composition

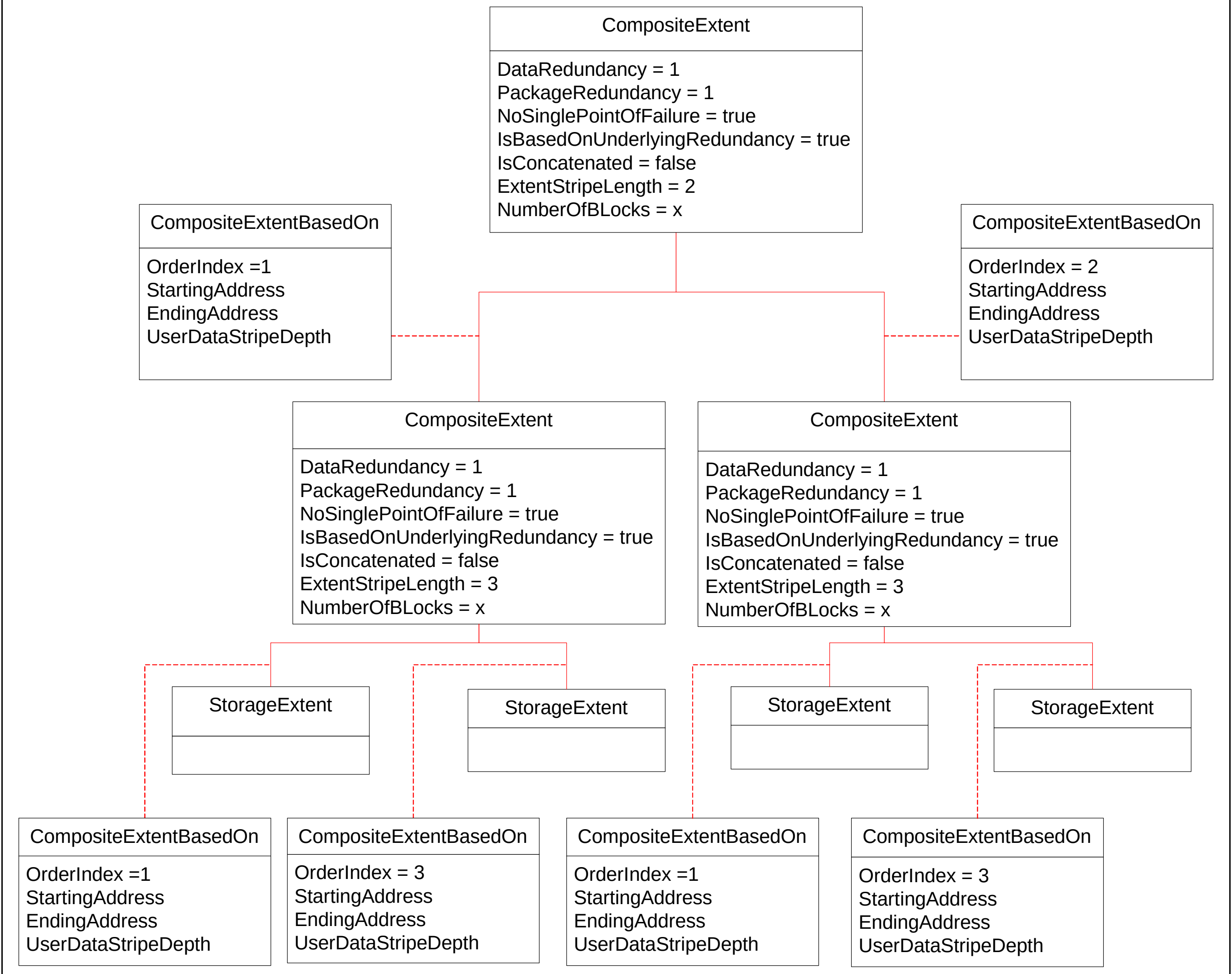


8.2.8.16.1.12 RAID 50

Figure 199: "RAID 50 Composition" 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.

Figure 199: RAID 50 Composition

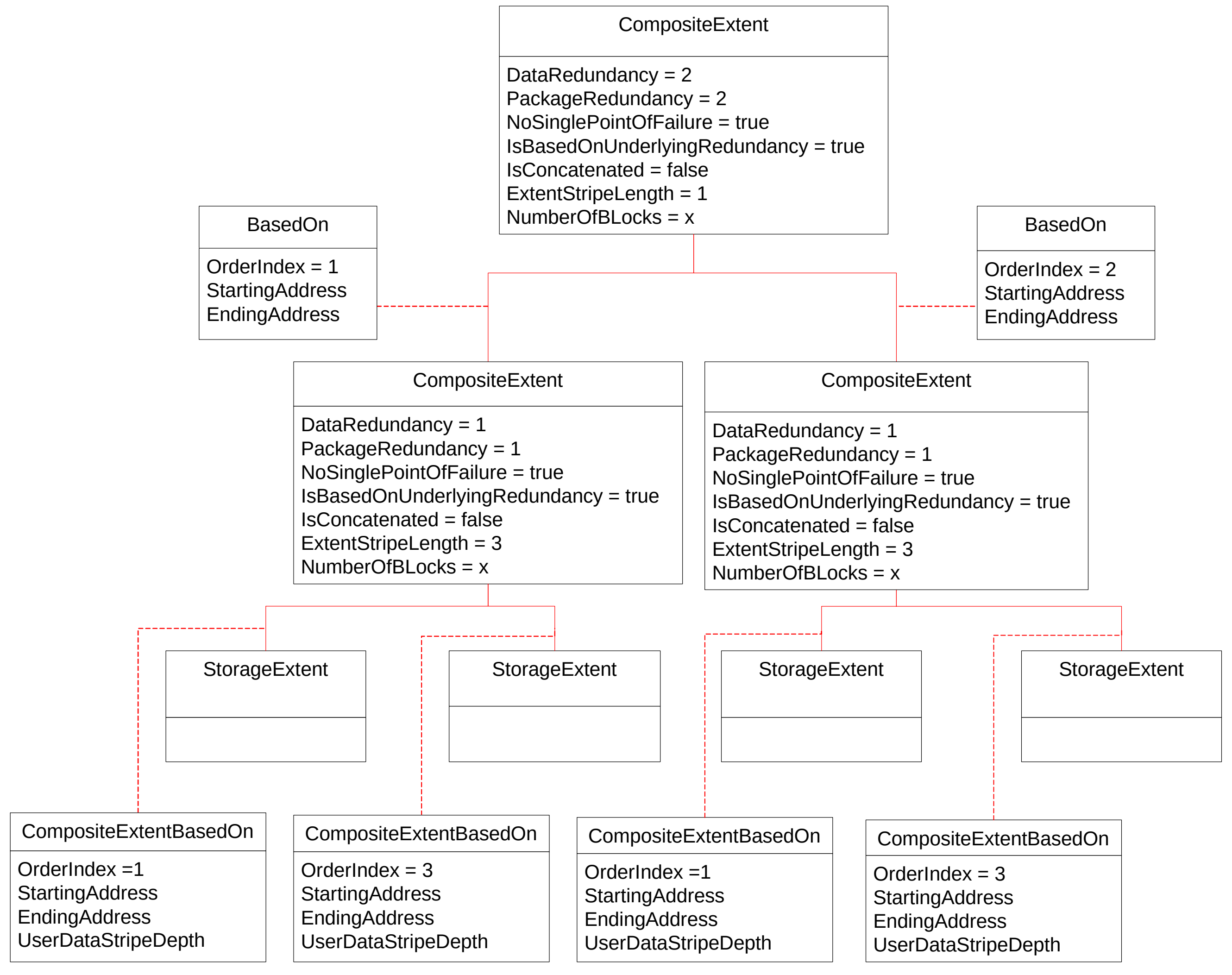


8.2.8.16.1.13 RAID 51

Figure 200: "RAID 51 Composition" 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 200: RAID 51 Composition



8.2.8.16.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.16.3 Cascading Considerations

None.

8.2.8.16.4 Supported Subprofiles and Packages

None.

8.2.8.16.5 Methods of the Profile

None.

8.2.8.16.6 Client Considerations and Recipes

8.2.8.16.6.1 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 Primoridal 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.)
    #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 ($CompositeExtent.IsConcatenated == false)
     && ($CompositeExtent.ExtentStripeLength > 1)) {

    // The TargetExtents are striped together.
    //
    // If the provider is surfacing a CIM_CompositeExtentBasedOn
    // get the Stripe Depth from the first association.
    // The assumption here is that this property is
     // the same for each association instance.
     #StripeDepth = 0;
    if (($Associations[0] ISA CIM_CompositeExtentBasedOn) {
       #StripeDepth = $Associations[0].UserDataStripeDepth
    }

     // Inspect the RAID level.
     #RAID = 0
     if ($CompositeExtent.IsBasedOnUnderlyingRedundancy) {
         #RAID = 5
```

```
                }
            } else {
                // This is either a Mirror or a Concatenation
                if (($CompositeExtent.IsBasedOnUnderlyingRedundancy == true)
                    && ($CompositeExtent.IsConcatenated == false)
                    && ($CompositeExtent.ExtentStripeLength == 1)) {
                     // The TargetExtents are mirrored together,
                     // This level is a RAID 1
                } else if (($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 Primoridal 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 composite 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->)
```

8.2.8.16.6.2 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>
   }
```

8.2.8.16.7 Registered Name and Version

Extent Composition version 1.1

8.2.8.16.8 CIM Server Requirements

Table 1124: CIM Server Requirements for Extent Composition

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.16.9 CIM Elements

Table 1125: CIM Elements for Extent Composition

Element Name	Description
Mandatory Classes	
CIM_BasedOn (8.2.8.16.9.1)	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.
CIM_CompositeExtent (8.2.8.16.9.2)	Used to represent the composition of several antecedent extents into a range of blocks with desired size or Quality-Of-Service.
CIM_CompositeExtentBasedOn (8.2.8.16.9.3)	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 and Extent ordering information.
CIM_ConcreteComponent (8.2.8.16.9.4)	Used to associate extents that are playing the Pool Component role to their parent StoragePool.
CIM_StorageExtent (8.2.8.16.9.5)	Used to represent the decomposition (partial allocation) of an Antecedent extent.

8.2.8.16.9.1 CIM_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.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1126: SMI Referenced Properties/Methods for CIM_BasedOn

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageExtent	
Dependent		CIM_StorageExtent	
OrderIndex		uint16	When the association is used in a concatenation composition, indicates the order in which the extents (and thus their block ranges) are concatenated.
Optional Properties/Methods			
StartingAddress		uint64	
EndingAddress		uint64	

8.2.8.16.9.2 CIM_CompositeExtent

Used to represent the composition of several antecedent extents into a range of blocks with desired size or Quality-Of-Service.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1127: SMI Referenced Properties/Methods for CIM_CompositeExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name	CD	string	
Primordial		boolean	Always False.
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ExtentStatus		uint16[]	
DataRedundancy		uint16	
PackageRedundancy		uint16	
NoSinglePointOfFailure		boolean	
IsBasedOnUnderlyingRedundancy		boolean	
IsConcatenated		boolean	
ExtentStripeLength		uint64	
NumberOfBlocks		uint64	If the extent maps to a hardware extent, the number of blocks as reported by the hardware.
ConsumableBlocks		uint64	The number of usable blocks.
BlockSize		uint64	

8.2.8.16.9.3 CIM_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 and Extent ordering information.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1128: SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageExtent	
Dependent		CIM_CompositeExtent	
OrderIndex		uint16	Indicates the order in which the extents have blocks striped onto them.

Table 1128: SMI Referenced Properties/Methods for CIM_CompositeExtentBasedOn

Property	Flags	Type	Description & Notes
UserDataStripeDepth		uint64	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.
Optional Properties/Methods			
StartingAddress		uint64	
EndingAddress		uint64	

8.2.8.16.9.4 CIM_ConcreteComponent

Not part of subprofile when used by Volume Manager profile

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1129: SMI Referenced Properties/Methods for CIM_ConcreteComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_ManagedElement	
PartComponent		CIM_ManagedElement	

8.2.8.16.9.5 CIM_StorageExtent

Used to represent the decomposition (partial allocation) of an Antecedent extent.
Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1130: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	
ExtentStatus		uint16[]	
NumberOfBlocks		uint64	If this extent maps to hardware, the number of blocks as reported by the hardware.
ConsumableBlocks		uint64	The number of usable blocks.
BlockSize		uint64	

Table 1130: SMI Referenced Properties/Methods for CIM_StorageExtent

Property	Flags	Type	Description & Notes
Primordial		boolean	This property is set to true if the StorageExtent represents the base of the StorageExtent hierarchy. In other words, the StorageExtents that are most Antecedent. For Arrays, these StorageExtents are generally related via MediaPresent with DiskDrives, see Disk Drive Lite Subprofile. For Storage Virtualizer, these Extents are generally associated to LUNs, see Storage Virtualizer Subprofile. Otherwise, this property should be set to false.

8.2.8.16.10 Related Standards

Table 1131: Related Standards for Extent Composition

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
Representation of CIM using XML	2.2.0	DMTF
CIM Schema	2.11.0	DMTF

DEPRECATED

8.2.8.17 Extent Mapping Subprofile (DEPRECATED)

The functionality of the Extent Mapping Subprofile (in IS24775-2006, *Storage Managemen*t) has been subsumed by 8.2.8.16, "Extent Composition Subprofile".

DEPRECATED

DEPRECATED

8.2.8.18 LUN Creation Subprofile (DEPRECATED)

The functionality of the LUN Creation and Pool Manipulation Capabilities, and Settings Subprofiles have been subsumed by the 8.2.8.10, "Block Services Package".

The LUN Creation Subprofile is defined in IS24775-2006, *Storage Managemen*t.

DEPRECATED

DEPRECATED

8.2.8.19 LUN Mapping and Masking Subprofile

The LUN Mapping and Masking Subprofile (in IIS24775-2006, *Storage Managemen*t) has been replaced by 8.2.8.20, "Masking and Mapping Subprofile".

8.2.8.19.1 Compatibility with IS24775-2006, *Storage Managemen*t *(*SMI-S 1.0) clients.

Problems with the functionality and complexity of the LUN Mapping and Masking subprofile in IIS24775-2006, *Storage Management* required some changes that may not be backwards compatible in this version of SMI-S. The mapping and Masking Subprofile now reduces the complexity by replacing the extrinsic methods specified in IS24775-2006, *Storage Managemen*t 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 this version of SMI-S.

Class and association changes to the model for this version of SMI-S:

- SAPAvailableForElement replaces the ProtocolControllerForPort association
- SCSIProtocolEndpoint replaces LogicalPort
- LogicalPort is associated to SCSIProtocolEndpoint via PortImplementsEndpoint)
- AuthorizedPrivilege associations to SystemSpecificCollection via AuthorizedSubject associations are no longer allowed

Instrumentation may be able to provide 1.0 compliant implementations (as specified in IIS24775-2006, *Storage Management)* 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 and current (1.1) implementations in separate CIM namespaces.

DEPRECATED

8.2.8.20 Masking and Mapping Subprofile

8.2.8.20.1 Description

See “8.2.8.19.1, "Compatibility with IS24775-2006, Storage Management (SMI-S 1.0) clients."” for notes on compatibility with the LUN Mapping and Masking Subprofile in IIS24775-2006, *Storage Managemen*t.

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.

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 IS24775-2006, *Storage Managemen*t *(*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.

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). SCSIProtocolEnpoint 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 201: "Generic System with no Configuration Service" and Figure 202: "Generic System with ControllerConfigurationService"“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 201: "Generic System with no Configuration Service" depicts a configuration with no LUN Masking capabilities.

Figure 201: Generic System with no Configuration Service

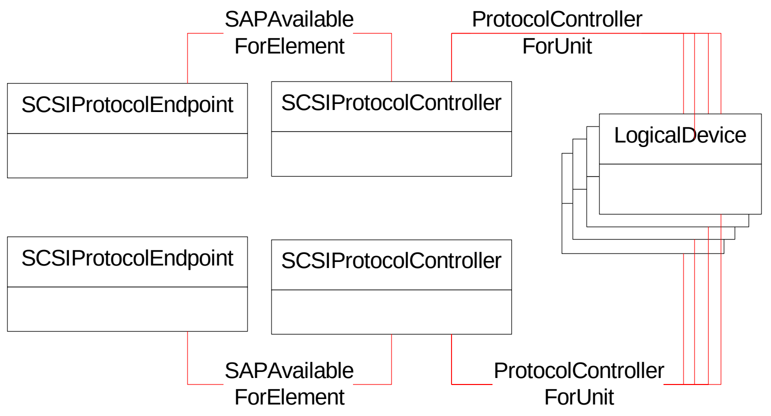
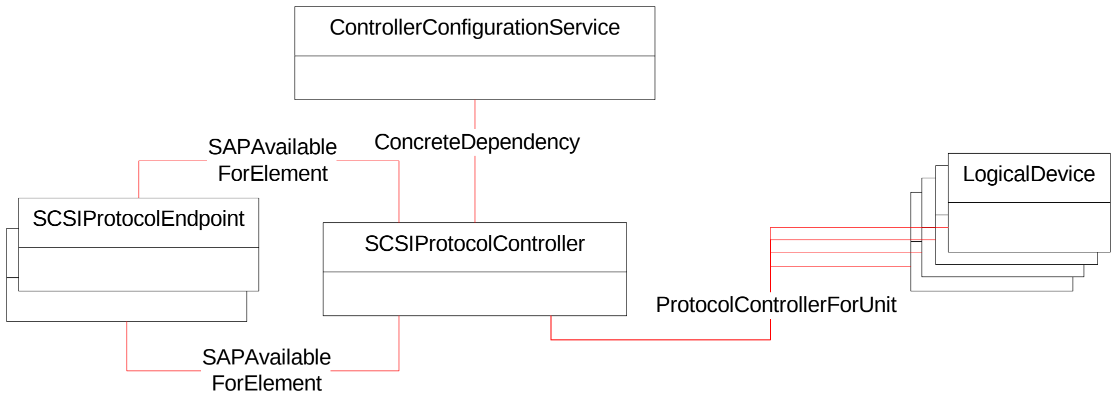


Figure 202: "Generic System with ControllerConfigurationService" depicts the same configuration in an implementation with an ControllerConfigurationService defined. In this case, access to the ProtocolController is denied to each host port unless it is specifically granted access.

Figure 202: Generic System with ControllerConfigurationService



8.2.8.20.1.1 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).

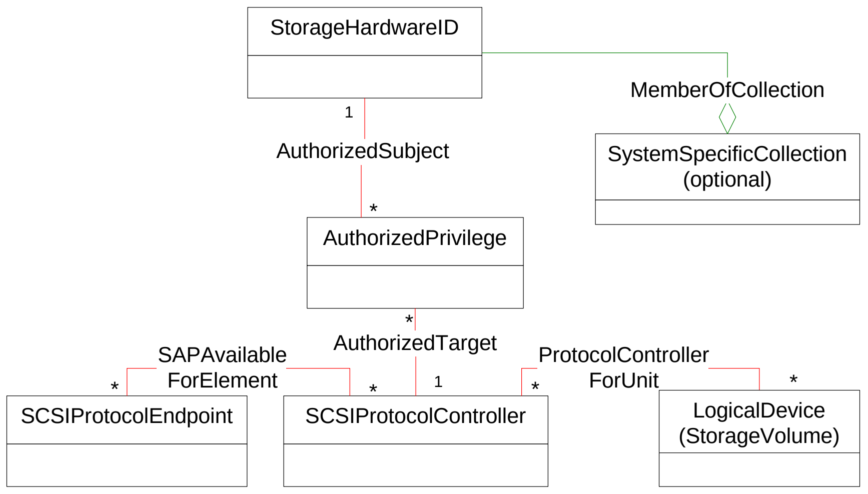
An administrator can use the ControllerConfigurationService interfaces to create "views" (SCSIProtocolControllers) of a storage system – each view exposes a subset of components that are intended to behave as a cohesive subset. In particular, a view:

- 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 203: Relationship of Initiator IDs, Endpoints, and Logical Units



Initiator ID Collections

An implementation may optionally model collections of Initiator IDs. This is modeled as depicted in Figure 203. If the implementation supports collection of initiator IDs, the instrumentation shall set ProtocolControllerMaskingCapabilities.ProtocolControllerSupportsCollections to True

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 or 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.

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.

Read-only verses Read-Write access

ExposePaths includes a DeviceAccess parameter that is used to set the DeviceAccess property of ProtocolControllerForUnit association.

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).

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.

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.

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 IS24775-2006, *Storage Managemen*t, a client could have created a configuration that does not comply with the semantics in this version of SMI-S (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.

SCSIProtocolController Properties

Table 1132: 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 DeviceAcceses 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	In 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 InitatorPortIDs, then see 2
1. This only applies to the "Create a new view" use case for ExposePaths 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.			

There are two clarifications to the property descriptions in Figure 1132: "SCSIProtocolController Property Description". 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.

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). 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 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. Figure 204: "StorageClientSettingData Model" provides an example.

Figure 204: StorageClientSettingData Model

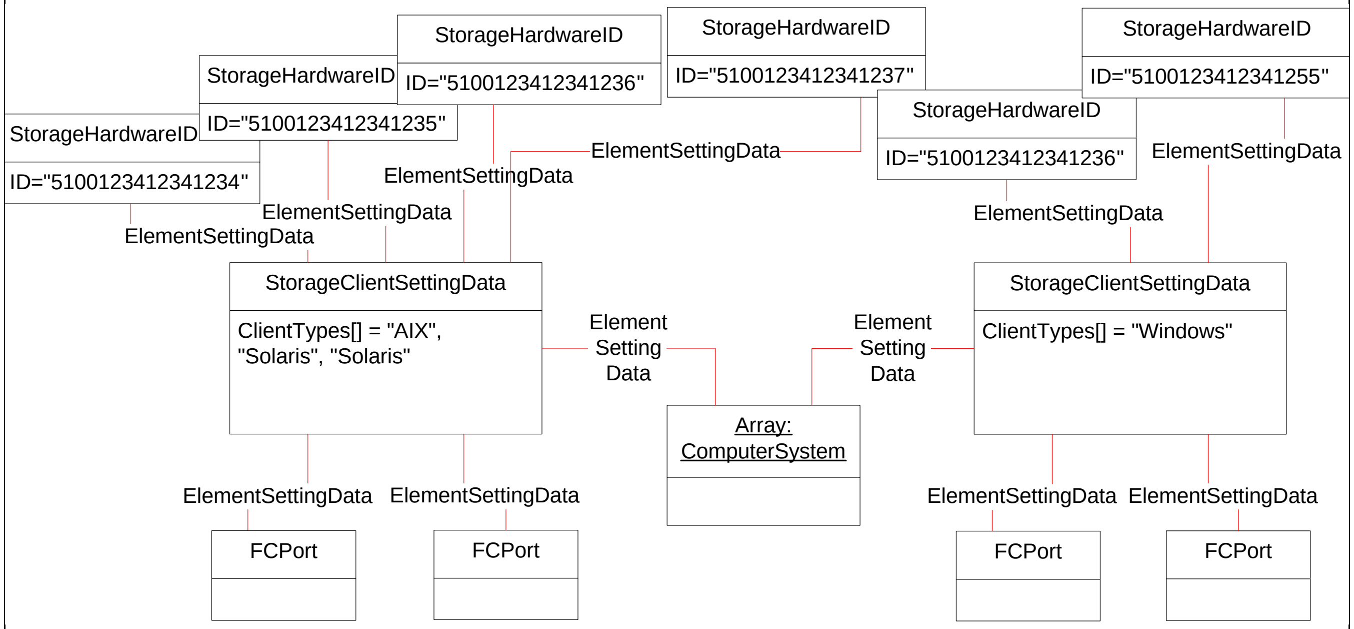
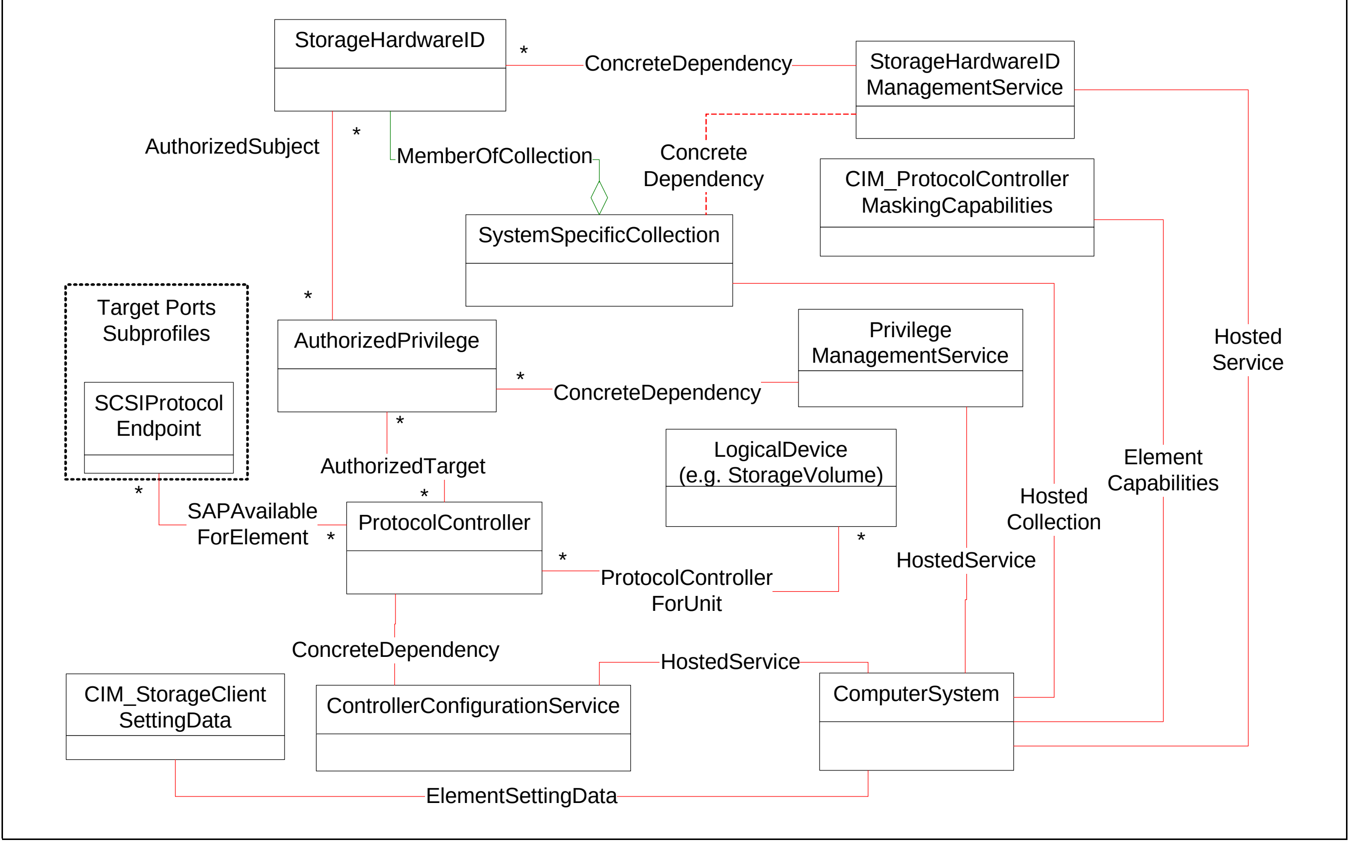


Figure 205: "Entire Model" depicts the entire model.

Figure 205: Entire Model



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.

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.

8.2.8.20.2 Health and Fault Management Considerations

None.

8.2.8.20.3 Cascading Considerations

None.

8.2.8.20.4 Supported Subprofiles, and Packages

None.

8.2.8.20.5 Methods of the Profile

8.2.8.20.5.1 ExposePaths

ExposePaths is used in place of the AssignAccess and AttachDevice methods used in IS24775-2006, *Storage Management* (SMI-S 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 is 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).

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 1133.

Table 1133: 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	containsa single SPC ref
Vendor-specific	As long as all the previous usecases are implemented, the instrumentation may support other vendor-specific combinations of parameters.					
	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.

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). If a job was started, references to the SPCs affected will be found by following the AffectedJobElement association from the job.

8.2.8.20.5.2 HidePaths

HidePaths is used in place of the HideAccess and DetachDevice methods used in IS24775-2006, *Storage Management*. 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 for the IS24775-2006, *Storage Management* (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. The use cases for HidePaths() are summarized in Table 1134, “HidePaths Use Cases”.

Table 1134: 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
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 usecases 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 created or modified. 2. Only valid if PortsPerView is "Multiple Ports per View"				

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.

8.2.8.20.5.3 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.

ExposeDefaultLUs and HideDefaultLUs are optional methods of this subprofile. However, they are linked. If an instrumentation implements one of these methods, it shall also implement the other.

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 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 a 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 1135, “Use Cases for ExposeDefaultLUs”.

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)

Table 1135: Use Cases for ExposeDefaultLUs

Parameters/ use cases	LUNames	TargetPortIDs	DeviceNumbers	DeviceAccesses	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 usecases are implemented, the instrumentation may support other vendor-specific combinations of parameters.				
1. Only valid if PortsPerView is not "All Ports share the same View" 2. Dependent on values of SPCAllows* capability properties described above 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 PortsPerView is "Multiple Ports per View"					

- 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.

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.

8.2.8.20.5.4 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).

ExposeDefaultLUs and HideDefaultLUs are optional methods of this subprofile. However, they are linked. If an instrumentation implements one of these methods it shall also implement the other

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 1136, “Use Cases for HideDefaultLUs”

Table 1136: 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

Table 1136: Use Cases for HideDefaultLUs

parameters/ use cases	LUNames	TargetPortIDs	ProtocolController (on input)
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.

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.

8.2.8.20.5.5 CreateStorageHardwareID

CreateStorageHardwareID creates a StorageHardwareID and the ConcreteDependency association between this service and the new StorageHardwareID.

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.

8.2.8.20.5.6 DeleteStorageHardwareID

DeleteStorageHardwareID deletes a StorageHardwareID and the ConcreteDependency association between the ID and the service.

Uint32 DeleteStorageHardwareID

IN CIM_StorageHardwareID REF HardwareID

REF to the StorageHardwareID to delete

8.2.8.20.5.7 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.

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.

8.2.8.20.5.8 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.

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.

8.2.8.20.6 Client Considerations and Recipes

8.2.8.20.6.1 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! >
         }
     } else {
         <ERROR! Missing Job reference>
     }
    }
}

// Step 1. Subscribe for indications on the Job
// Job success -- Status is ‘17’ (“Completed”) and ‘2’ (“OK”)
#Filter1 = “SELECT FROM CIM_InstModification
            WHERE SourceInstance ISA CIM_ConcreteJob
            AND ANY SourceInstance.OperationalStatus[*] = 17
            AND ANY SourceInstance.OperationalStatus[*] = 2 “
// Determine if the Indication already exists
// If it doesn’t, create it

// Job failure -- Status is ‘17’ (“Completed”) and ‘6’ (“Error”)
#Filter2 = “SELECT * FROM CIM_InstModification
            WHERE SourceInstance ISA CIM_ConcreteJob
            AND ANY SourceInstance.OperationalStatus[*] = 17
            AND ANY SourceInstance.OperationalStatus[*] = 6 “
// 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[“DeviceAccesses”]   = {2} // Read-Write
%InputArguments[“InitiatorPortIDs”] = {#InitiatorWWN1}

if (#PortsPerView != 4) {// 4 = All ports share the same view
    %InputArguments[“TargetPortIDs”] = {#TargetPortWWN}
}

if (#ClientSelectableDeviceNumbers == TRUE) {
  %InputArguments[“DeviceNumbers”] = {#DeviceNumber}
}
else {
  %InputArguments[“DeviceNumbers”] = 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”,
                        “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
        &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>
          }
      }
```

8.2.8.20.7 Registered Name and Version

Masking and Mapping version 1.1.0

8.2.8.20.8 CIM Server Requirements

Table 1137: CIM Server Requirements for Masking and Mapping

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.20.9 CIM Elements

Table 1138: CIM Elements for Masking and Mapping

Element Name	Description
Mandatory Classes	
CIM_AuthorizedPrivilege (8.2.8.20.9.1)	
CIM_AuthorizedSubject (8.2.8.20.9.2)	
CIM_AuthorizedTarget (8.2.8.20.9.3)	
CIM_ConcreteDependency (8.2.8.20.9.4)	
CIM_ControllerConfigurationService (8.2.8.20.9.5)	
CIM_ElementCapabilities (8.2.8.20.9.6)	
CIM_ElementSettingData (8.2.8.20.9.7)	
CIM_HostedService (8.2.8.20.9.9)	
CIM_LogicalDevice (8.2.8.20.9.10)	
CIM_PrivilegeManagementService (8.2.8.20.9.12)	
CIM_ProtocolController (8.2.8.20.9.13)	
CIM_ProtocolControllerForUnit (8.2.8.20.9.14)	
CIM_ProtocolControllerMaskingCapabilities (8.2.8.20.9.15)	
CIM_SAPAvailableForElement (8.2.8.20.9.16)	
CIM_SCSIProtocolEndpoint (8.2.8.20.9.17)	
CIM_StorageClientSettingData (8.2.8.20.9.18)	
CIM_StorageHardwareID (8.2.8.20.9.19)	
CIM_StorageHardwareIDManagementService (8.2.8.20.9.20)	
Optional Classes	
CIM_HostedCollection (8.2.8.20.9.8)	
CIM_MemberOfCollection (8.2.8.20.9.11)	
CIM_SystemSpecificCollection (8.2.8.20.9.21)	

Table 1138: CIM Elements for Masking and Mapping

Element Name	Description
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ProtocolController	Creation of a ProtocolController
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ProtocolController	Deletion of a ProtocolController
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Creation of a ProtocolControllerForUnit association
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Deletion of a ProtocolControllerForUnit association
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ProtocolControllerForUnit	Modification of a ProtocolControllerForUnit association (e.g.,changing DeviceNumber)
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_AuthorizedSubject	Creation of an AuthorizedSubject association
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_AuthorizedSubject	Deletion of an AuthorizedSubject association

8.2.8.20.9.1 CIM_AuthorizedPrivilege

Created By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Modified By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Deleted By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Class Mandatory: true

Table 1139: SMI Referenced Properties/Methods for CIM_AuthorizedPrivilege

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque and unique identifer
PrivilegeGranted		boolean	Indicates if the privilege is granted or not
Activities		uint16[]	For SMI-S, must be "Read", "Write"
Optional Properties/Methods			
ElementName		string	User friendly name

8.2.8.20.9.2 CIM_AuthorizedSubject

Created By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Modified By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Deleted By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Class Mandatory: true

Table 1140: SMI Referenced Properties/Methods for CIM_AuthorizedSubject

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	The Privilege either granted or denied to an Identity or group of Identities collected by a Role
PrivilegedElement		CIM_ManagedElement	The Subject for which Privileges are granted or denied

8.2.8.20.9.3 CIM_AuthorizedTarget

Created By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Modified By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Deleted By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths
Class Mandatory: true

Table 1141: SMI Referenced Properties/Methods for CIM_AuthorizedTarget

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Privilege		CIM_AuthorizedPrivilege	The Privilege affecting the target resource
TargetElement		CIM_ManagedElement	The target set of resources to which the Privilege applies

8.2.8.20.9.4 CIM_ConcreteDependency

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 1142: SMI Referenced Properties/Methods for CIM_ConcreteDependency

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ManagedElement	Represents the independent object in this association
Dependent		CIM_ManagedElement	Represents the object dependent on the Antecedent.

8.2.8.20.9.5 CIM_ControllerConfigurationService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

8.2.8.20.9.6 CIM_ElementCapabilities

Created By : Static
Modified By : Static

Table 1143: SMI Referenced Properties/Methods for CIM_ControllerConfigurationService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClass-Name
SystemName		string	The scoping System Name
CreationClassName		string	The name of the concrete subclass
Name		string	Unique identifer for the Service
ExposePaths()			
HidePaths()			
Optional Properties/Methods			
ExposeDefaultLUs()			
HideDefaultLUs()			

Deleted By : Static
Class Mandatory: true

Table 1144: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The ComputerSystem
Capabilities		CIM_Capabilities	The ProtocolControllerMaskingCapabil-ities

8.2.8.20.9.7 CIM_ElementSettingData

Created By : Extrinsic(s): CIM_StorageHardwareIDManagementService.CreateStorageHardwareID
Modified By : ModifyInstance
Deleted By : Extrinsic(s): CIM_StorageHardwareIDManagementService.DeleteStorageHardwareID
Class Mandatory: true

Table 1145: SMI Referenced Properties/Methods for CIM_ElementSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The managed element (e.g., computer system)
SettingData		CIM_SettingData	The SettingData object associated with the element

8.2.8.20.9.8 CIM_HostedCollection

Created By : Extrinsic(s): CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

8.2.8.20.9.9 CIM_HostedService

Created By : Static
Modified By : Static

Table 1146: SMI Referenced Properties/Methods for CIM_HostedCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The ComputerSystem
Dependent		CIM_SystemSpecificCollection	The SystemSpecificCollection

Deleted By : Static
Class Mandatory: true

Table 1147: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The ComputerSystem hosting the service
Dependent		CIM_Service	The Service hosted

8.2.8.20.9.10 CIM_LogicalDevice

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1148: SMI Referenced Properties/Methods for CIM_LogicalDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClassName
CreationClassName		string	The name of the concrete subclass
SystemName		string	The scoping System Name
DeviceID		string	Unique identifer

8.2.8.20.9.11 CIM_MemberOfCollection

Created By : Extrinsic(s):
CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection,CIM_StorageHardwareIDManagementService.AddHardwareIDsToCollection
Modified By : ModifyInstance
Deleted By : DeleteInstance
Class Mandatory: false

Table 1149: SMI Referenced Properties/Methods for CIM_MemberOfCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Collection		CIM_Collection	The SystemSpecificCollection
Member		CIM_ManagedElement	The StorageHardwareID

8.2.8.20.9.12 CIM_PrivilegeManagementService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 1150: SMI Referenced Properties/Methods for CIM_PrivilegeManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClass-Name
CreationClassName		string	The name of the concrete subclass
SystemName		string	The scoping System Name
Name		string	Uniquely identifies the Service
ElementName		string	User friendly name

8.2.8.20.9.13 CIM_ProtocolController

Created By : Extrinsic(s): CIM_ControllerConfigurationService.Expose-Paths,CIM_ControllerConfigurationService.HidePaths
Modified By : Extrinsic(s): CIM_ControllerConfigurationService.Expose-Paths,CIM_ControllerConfigurationService.HidePaths
Deleted By : Extrinsic(s): CIM_ControllerConfigurationService.Expose-Paths,CIM_ControllerConfigurationService.HidePaths
Class Mandatory: true

Table 1151: SMI Referenced Properties/Methods for CIM_ProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClass-Name
CreationClassName		string	The name of the concrete subclass
SystemName		string	The scoping SystemsName'
DeviceID		string	Unique name for the ProtocolController

8.2.8.20.9.14 CIM_ProtocolControllerForUnit

Created By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Modified By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Deleted By : Extrinsic(s): CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Class Mandatory: true

8.2.8.20.9.15 CIM_ProtocolControllerMaskingCapabilities

Created By : Static
Modified By : Static

Table 1152: SMI Referenced Properties/Methods for CIM_ProtocolControllerForUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolController	The ProtocolController
Dependent		CIM_LogicalDevice	The logical unit (eg StorageVolume) behind the ProtocolController
DeviceNumber		string	Address (e.g. LUN) of the associated Device. Shall be formatted as unseparated uppercase hexadecimal digits, with no leading 0x.
DeviceAccess		uint16	The access rights granted to the referenced logical unit as exposed through referenced ProtocolController

Deleted By : Static
Class Mandatory: true

Table 1153: SMI Referenced Properties/Methods for CIM_ProtocolControllerMaskingCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque and unique identifer
ElementName		string	User-friendly name
ValidHardwareIdTypes		uint16[]	A list of the valid values for Strorage-HardwareID.IDType
PortsPerView		uint16	Indicates the way that ports per view (ProtocolController) are handled
ClientSelectableDeviceNumbers		boolean	Indicates if the client can specify the DeviceNumbers parameter when calling ControllerConfigurationService.ExposePaths().
OneHardwareIDPerView		boolean	Set to true if this storage system limits configurations to a single subject hardware ID per view.
PrivilegeDeniedSupported		boolean	Set to true if this storage system allows a client to create a Privilege instance with PrivilegeGranted set to FALSE.
UniqueUnitNumbersPerPort		boolean	Indicates if different ProtocolContollers attached to a SCSIProtocolEndpoint can expose the same unit numbers (e.g. multiple LUN 0s) or if the numbers must be unique
ExposePathsSupported		boolean	Set to true if this storage system supports the ExposePaths and HidePaths methods.
CreateProtocolControllerSupported		boolean	This property was used in the SMI-S 1.0 LUN Mapping and Masking subprofile. It is not required in SMI-S 1.1 and shall be set to false.

Table 1153: SMI Referenced Properties/Methods for CIM_ProtocolControllerMaskingCapabilities

Property	Flags	Type	Description & Notes
MaximumMapCount		uint16	The maximum number of ProtocolCOn-trollerForUnit associations that can be associated with a single LogicalDevice (for example, StorageVolume). Zero indicates there is no limit
SPCAllowsNoLUs		boolean	Set to true if a client can create an SPC with no LogicalDevices
SPCAllowsNoTargets		boolean	Set to true if a client can create an SPC with no target SCSIProtocolEndpoints
SPCAllowsNoInitiators		boolean	Set to true if a client can create an SPC with no StorageHardwareIDs
SPCSupportsDefaultViews		boolean	Set to true if it the instrumentation sup-ports default view SPCs that exposes logical units to all initiators
Optional Properties/Methods			
ProtocolControllerSupportsCollec-tions		boolean	Indicates the storage system supports SystemSpecificCollections of Storage-HardwareIDs
OtherValidHardwareIDTypes		string[]	An array of strings describing types for valid StorageHardwareID.IDType. Used when the ValidHardwareIdTypes includes Other

8.2.8.20.9.16 CIM_SAPAvailableForElement

Created By : Extrinsic(s):
CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Modified By : Extrinsic(s):
CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Deleted By : Extrinsic(s):
CIM_ControllerConfigurationService.ExposePaths,CIM_ControllerConfigurationService.HidePaths,CIM_ControllerConfigurationService.ExposeDefaultLUs,CIM_ControllerConfigurationService.HideDefaultLUs
Class Mandatory: true

Table 1154: SMI Referenced Properties/Methods for CIM_SAPAvailableForElement

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The ManagedElement (ProtocolCon-troller) for which the SAP is available
AvailableSAP		CIM_ServiceAccessPoi nt	The Service Access Point (SCSIProto-colEndpoint) that is available

8.2.8.20.9.17 CIM_SCSIProtocolEndpoint

Created By : External
Modified By : External
Deleted By : External
Class Mandatory: true

Table 1155: SMI Referenced Properties/Methods for CIM_SCSIProtocolEndpoint

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClass-Name
SystemName		string	The scoping System Name
CreationClassName		string	The name of the concrete subclass

8.2.8.20.9.18 CIM_StorageClientSettingData

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

Table 1156: SMI Referenced Properties/Methods for CIM_StorageClientSettingData

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque and unique identifier
ElementName		string	A user-friendly name
ClientTypes		uint16[]	Array of OS names

8.2.8.20.9.19 CIM_StorageHardwareID

Created By : Extrinsic(s): CIM_StorageHardwareIDManagementService.CreateStorageHardwareID
Modified By : Static
Deleted By : Extrinsic(s): CIM_StorageHardwareIDManagementService.DeleteStorageHardwareID
Standard Names: For FibreChannel attached initiators, the StorageID Property shall follow the requirements in 6.2.4.5.2 and IDType shall be PortWWN (2). For iSCSI attached initiators, the StorageID Property shall follow the requirements in 6.2.4.5.4 and IDType shall be iSCSI Name (5).
Class Mandatory: true

Table 1157: SMI Referenced Properties/Methods for CIM_StorageHardwareID

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque and unique identifier
StorageID	N	string	The worldwide unique ID
IDType		uint16	StorageID type. Values are Other, PortWWN, NodeWWN, Hostname, and iSCSI Name

8.2.8.20.9.20 CIM_StorageHardwareIDManagementService

Created By : Static
Modified By : Static
Deleted By : Static
Class Mandatory: true

8.2.8.20.9.21 CIM_SystemSpecificCollection

Created By : Extrinsic(s): CIM_StorageHardwareIDManagementService.CreateHardwareIDCollection
Modified By : ModifyInstance

Table 1158: SMI Referenced Properties/Methods for CIM_StorageHardwareIDManagementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	The scoping System CreationClass-Name
SystemName		string	The scoping System Name
CreationClassName		string	The name of the concrete subclass
Name		string	Uniquely identifies the Service
CreateStorageHardwareID()			
DeleteStorageHardwareID()			
Optional Properties/Methods			
CreateHardwareIDCollection()			
AddHardwareIDsToCollection()			

Deleted By : DeleteInstance
Class Mandatory: false

Table 1159: SMI Referenced Properties/Methods for CIM_SystemSpecificCollection

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Opaque and unique identifier
ElementName		string	A user-friendly name

8.2.8.20.10 Related Standards

Table 1160: Related Standards for Masking and Mapping

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

DEPRECATED

8.2.8.21 Pool Manipulation Capabilities, and Settings Subprofile (DEPRECATED)

The functionality of the LUN Creation and Pool Manipulation Capabilities, and Settings Subprofiles has been subsumed by the 8.2.8.10, "Block Services Package".

The Pool Manipulation Capabilities, and Settings Subprofile is defined in IS24775-2006, *Storage Management*.

DEPRECATED

8.2.8.22 Storage Library Profile

8.2.8.22.1 Description

The schema for a storage library provides the classes and associations necessary to represent various forms of removable media libraries. This profile is based upon the CIM 2.11.0 model and defines the subset of classes that supply the necessary information for robotic storage libraries.

This profile further describes how the classes are to be used to satisfy various use cases and offers suggestions to agent implementers and client application developers. Detailed descriptions of classes are from the CIM 2.11.0 schema.

The relevant objects for a storage library should be instantiated in the name space of the provider (or agent) for a storage library resource. Whenever an instance of a class for a resource may exist in multiple name spaces a durable name is defined to aid clients in correlating the objects across name spaces. For storage libraries, durable names are defined for the following resources:

- ChangerDevice
- ComputerSystem
- MediaAccessDevice

The durable names are defined in 8.2.8.22.1.6, "Durable Names and Correlatable IDs of the Profile". All other objects do not require durable names and have instances within a single name space.

8.2.8.22.1.1 Instance Diagrams

The following instance diagrams represent five related views of the storage library profile:

a) System Level

b) MediaAccessDevice and its physical and logical relationships

c) ChangerDevice and its connections to SoftwareIdentity, ProtocolController, and StorageMediaLocation

d) StorageMediaLocation and its relationship to PhysicalMedia and other physical classes

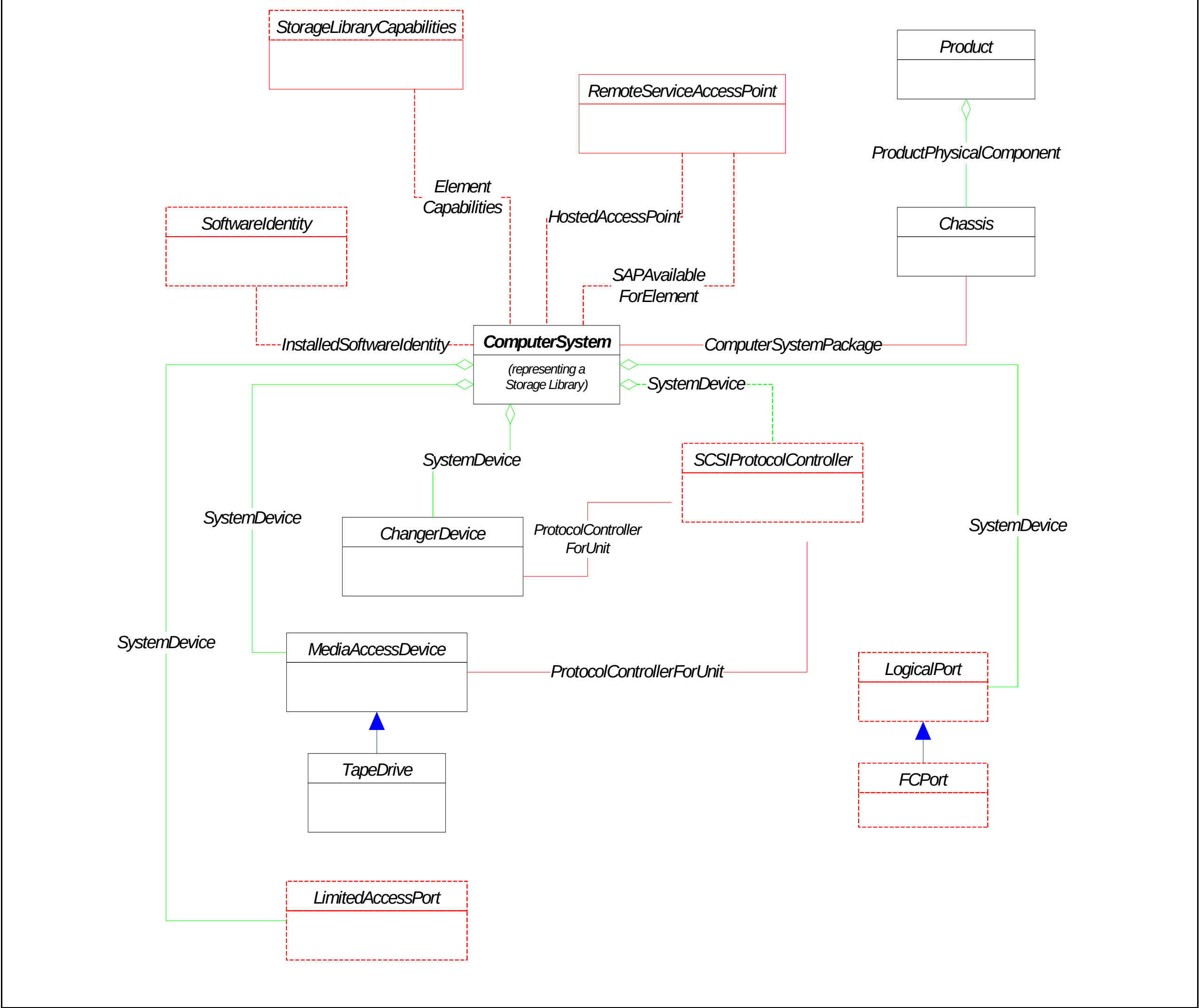
e) StorageMediaLocation and its required Realizes relationships.

8.2.8.22.1.2 System Level View

Figure 206: "Storage Library-centric Instance Diagram" shows the required components for a ComputerSystem. Note that LogicalDevice subclasses shall be associated with ComputerSystem via SystemDevice.

Note: Classes using a red outline and associations using a dotted outline represent optional components that have been included in the diagram as an aid to understanding.

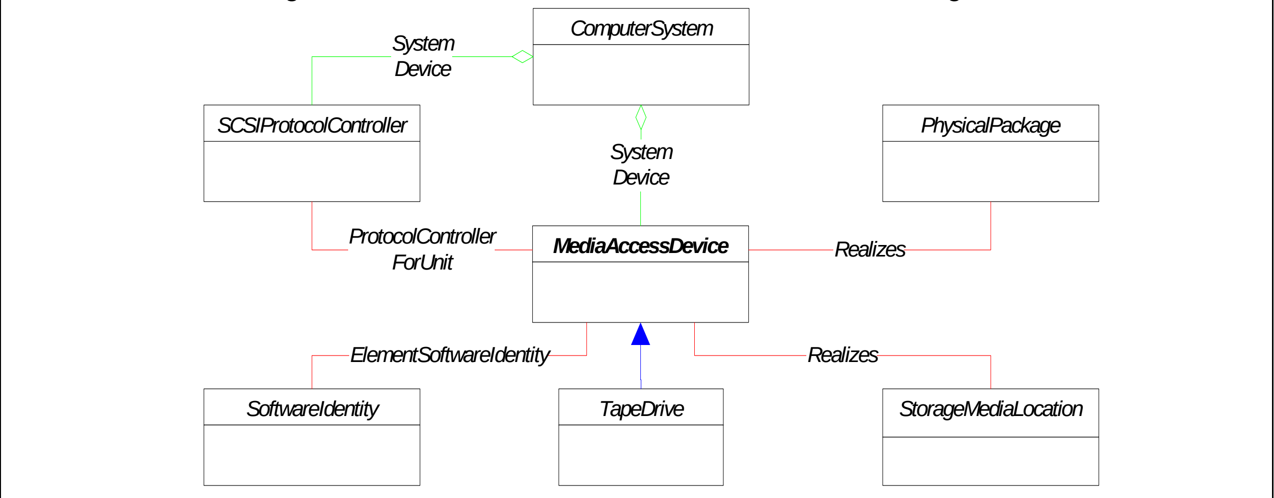
Figure 206: Storage Library-centric Instance Diagram



8.2.8.22.1.3 MediaAccessDevice-centric View

Figure 207: "MediaAccessDevice-centric Instance Diagram" shows the required classes related to MediaAccessDevice. Though not shown in this figure, both MediaAccessDevice and ProtocolController are connected to a ComputerSystem instance through the SystemDevice association. In some libraries, notably small autoloaders, external hosts access a library's ChangerDevice through the ProtocolController of a MediaAccessDevice. For such libraries, an additional ProtocolControllerForUnit association should be instantiated between the MediaAccessDevice's ProtocolController and the affected ChangerDevice. ProtocolControllerForUnit is a many-to-many association, so a single ProtocolController can be connected to multiple LogicalDevices if this accurately represents a library's configuration.

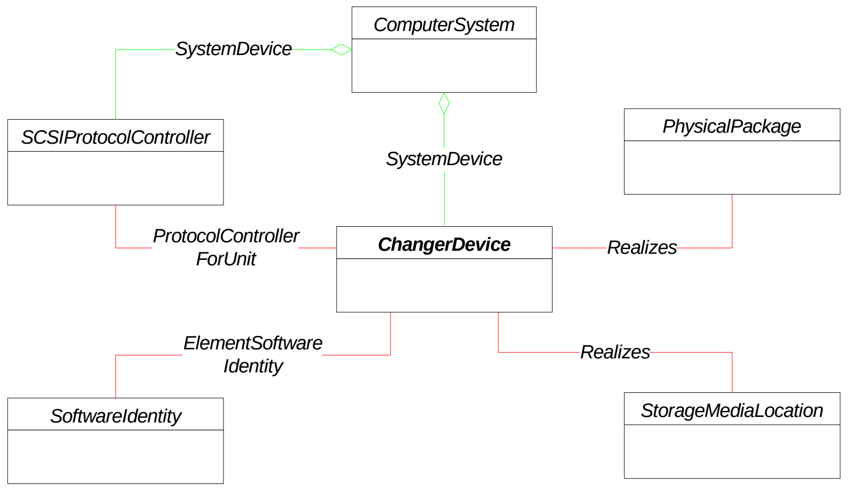
Figure 207: MediaAccessDevice-centric Instance Diagram



8.2.8.22.1.3.1 ChangerDevice-centric View

Figure 208: "ChangerDevice-centric Instance Diagram" shows the required classes related to ChangerDevice.

Figure 208: ChangerDevice-centric Instance Diagram



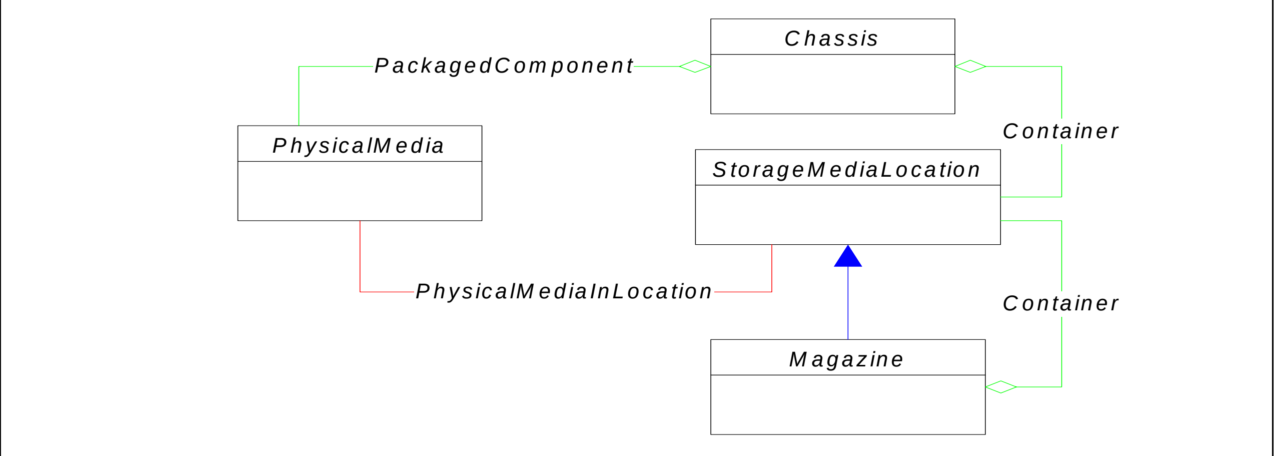
8.2.8.22.1.4 Physical View

Figure 209: "Physical View Instance Diagram" shows important physical components of a storage library and how they relate. With regard to StorageMediaLocation and Magazine, one of two implementation alternatives shall be selected:

a) Instantiate multiple Magazines associated to Chassis via Container, then instantiate StorageMediaLocations that are contained (again via Container) within each Magazine;

b) Instantiate multiple StorageMediaLocations directly associated to Chassis via Container, without the use of Magazines. Other optional classes, such as Panel, can also be used to group StorageMediaLocations, but this is not mandatory.

Figure 209: Physical View Instance Diagram

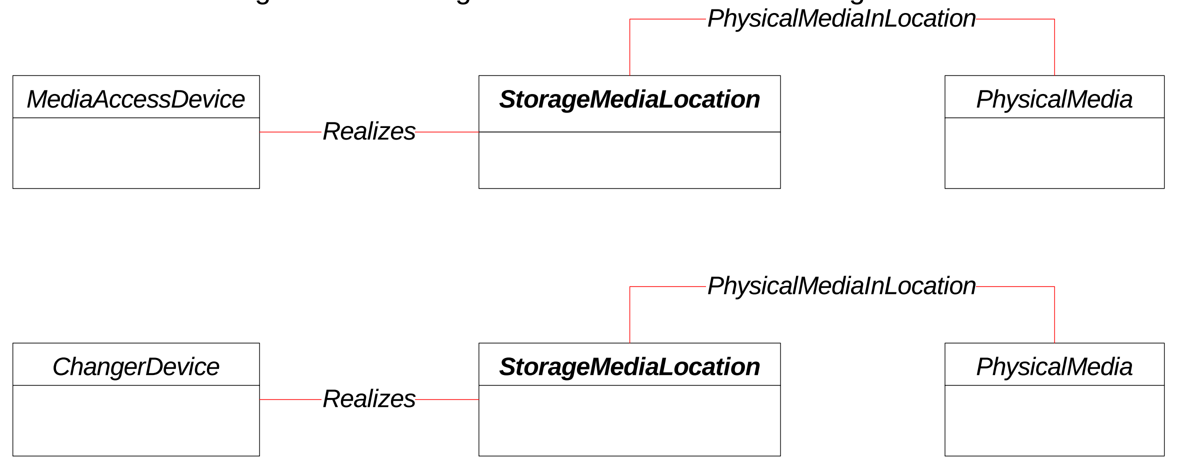


8.2.8.22.1.5 StorageMediaLocation Instance Diagram

Figure 210: "StorageMediaLocation Instance Diagram" shows relationships between various LogicalDevices (i.e., MediaAccessDevices, LimitedAccessPort, and ChangerDevice) and StorageMediaLocation. For each LogicalDevice that can hold media, at least one StorageMediaLocation shall be associated via Realizes.

The figure also shows how PhysicalMedia is conceptually placed “inside” a LogicalDevice by associating PhysicalMedia with a StorageMediaLocation that Realizes a LogicalDevice (see Figure 210: "StorageMediaLocation Instance Diagram"). All tapes, irrespective of the location, are associated with the chassis using PackagedComponent.

Figure 210: StorageMediaLocation Instance Diagram



8.2.8.22.1.6 Durable Names and Correlatable IDs of the Profile

Different implementations use different approaches to uniquely identify the SCSI units pertinent to Storage Media Libraries (i.e., Changer Devices and Media Access Devices). The agent should utilize the same Durable Name techniques described for volumes in the Disk Array section. The chosen name is stored in the Name attribute of the logical device with the corresponding setting for the NameFormat attribute. Allowable name formats and device pairings for the storage library profile are:

- FCPort: FCPort.PermanentAddress = Fibre Channel Port World Wide Name. NameFormat should be set to "WWN"
- ChangerDevice.DeviceID = Vendor+Product+Serial Number+(optional instance number). Vendor, Model and Serial number should be taken from the ChangerDevice's associated ComputerSystem, Product, and/or Chassis. An option instance number may be added to uniquely denote more than one ChangerDevice "inside" a ComputerSystem
- MediaAccessDevice (or TapeDrive).DeviceID = Vendor+Product+Serial number for the MediaAccessDevice
- ComputerSystem.Name = Vendor+Product+Serial number for the storage library and/or its associated Product and Chassis. NameFormat should be set to "Vendor+Product+Serial"

Refer to 6.2.4.5, "Standard Formats for Correlatable Names" for additional information.

8.2.8.22.2 Health and Fault Management Considerations

None

8.2.8.22.3 Cascading Considerations

None

Supported Subprofiles and Packages

Table 1161: Supported Subprofiles for Storage Library

Registered Subprofile Names	Mandatory	Version
Access Points	No	1.1.0
Location	No	1.1.0
FC Target Ports	No	1.1.0
Software	No	1.1.0
Storage Library Limited Access Port Elements	No	1.1.0
Storage Library Media Movement	No	1.1.0
Storage Library Capacity	No	1.1.0
Storage Library Element Counting	No	1.1.0
Storage Library InterLibraryPort Connection	No	1.1.0
Storage Library Partitioned Library	No	1.1.0

Table 1162: Supported Packages for Storage Library

Registered Package Names	Version
Physical Package	1.1.0

8.2.8.22.4 Methods of this Profile

None

8.2.8.22.5 Client Considerations and Recipes

8.2.8.22.5.1 Recipe Overview

While no pseudo-code-based recipes have been written for this profile, this section provides some helpful information for writing management applications and suggests techniques for addressing common use cases.

8.2.8.22.5.2 Discover a Storage Media Library

Discovery of Storage Media Libraries is achieved by looking up instances of ComputerSystem which are subclassed from System and have a corresponding Name and NameFormat property as described above under “Durable Names and Correlatable IDs of the Profile”. Specifically, NameFormat shall be set to “VendorModelSerial” and the Name shall be of the form Vendor+Product+Serial

8.2.8.22.5.3 Determine Library Physical Media Capacity

The physical media capacity of a library is the number of physical media objects that may be stored in the currently installed configuration of a Storage Media Library. This capacity may be determined by enumerating the StorageMediaLocation instances that are associated with each of the library’s Chassis objects.

In implementations that choose to include the Capacity subprofile, minimum and maximum slot capacities for a Storage Library are modeled in the ConfigurationCapacity, which is described earlier in the section on Capacity Constraints. Since this use case relies on an optional part of the profile, it may not be supported by each agent implementation.

8.2.8.22.5.4 Determine Physical Media Inventory

To determine the physical media inventory of a storage library, clients should discover the Chassis instance associated with a particular ComputerSystem (via the ComputerSystemPackage association), and enumerate the PhyscialMedia instances associated with the Chassis through the PackagedComponent association.

8.2.8.22.5.5 Discover Storage Library Control Type

The control mechanism to a library is either:

- SCSI Media Changer Commands directed to the library’s changer device,
- Library control commands directed to a Library Control service.

If a library does not have a ProtocolController instance associated via ProtocolControllerForUnit to the ChangerDevice then the client should conclude that an alternate mechanism for controlling the library is required. This mechanism may vary, but should be represented by an instance of Service as described in the section on Software/Service View for a library’s hosted services

8.2.8.22.5.6 Determine Library Drive Capacity

The current drive capacity of a library may be determined by enumerating the MediaAccessDevice instances through the SystemDevice association of the library.

When the optional Capacity subprofile is implemented, the number of drives discovered should be within the range indicated by the minimum and maximum capacity attribute found on the library Chassis’ ElementCapacity association with ConfigurationCapacity for tape drives. This bounds check is not available if the Capacity subprofile is not implemented.

8.2.8.22.5.7 Determine Drive Data Path Technology

Clients can discover the data path protocol of each drive within a storage library by enumerating MediaAccessDevice instances, then following the ProtocolControllerForUnit association linking a MediaAccessDevivce with a ProtocolController. Properties within Contoller can then be queried for more information. If the MediaAccessDevice has a fibre channel interface, an FCPort instance is linked to its ProtocolController by a ProtocolControllerForPort association. See 8.2.2.2, "FC Target Port Subprofile" for more information on fibre channel connectivity.

8.2.8.22.5.8 Find asset Information

Information about the entire storage library is modeled in the Chassis instances associated with the ComputerSystem. Chassis properties include Manufacturer, Model, Version, and Tag. Tag is an arbitrary identifying string.

To identify asset information for the logical devices, a client should access the corresponding logical device through the ComputerSystem object’s SystemDevice association. For each logical device instance the client may then check for asset information from the PhysicalElement associated through a Realizes association. Product information may also be available through the corresponding ProductPhysicalElement/ProductPhysicalComponent aggregation.

8.2.8.22.5.9 Discovery of Mailslots, Import/Export Elements or LimitedAccessPorts in a Storage Library

Clients may determine the number of LimitedAccessPorts in a library by enumerating the LimitedAccessPorts connected to a ComputerSystem instance via the SystemDevice association.

Note that some smaller libraries do not have the type of import/export element modeled by LimitedAccessPort. As a result, LimitedAccessPort elements are included in an (optional) subprofile (see 8.2.8.29, "Limited Access Port Elements Subprofile").

8.2.8.22.5.10 Counting assets in large storage libraries

Very large libraries may contain dozens of MediaAccessDevices and many thousands of StorageMediaLocations and PhysicalMedia. The intrinsic enumerateInstances() method is commonly used to count or gather CIM object instances of this type. Clients may find that using enumerateInstances() to count assets in very large libraries requires an excessive amount of time and processing resources. Providers supporting large libraries may also find that excessive time and resources are consumed attempting to return the bulk of data requested in enumerateInstances() calls. The following suggestions may be of help in situations where large libraries are of interest:

- Omit Qualifiers from enumerateInstances() or getInstance() requests;
- Request only the lowest-level child class of interest for examination or counting;
- Request only the properties of interest in enumerateInstances() or getInstance() requests. When only a count of existing objects is desired, omit all properties from the request;
- Use the intrinsic enumerateInstanceNames() or associatorNames() method instead of enumerateInstances() when only a count of existing objects is desired. The enumerateInstanceNames() and associatorNames() calls are much “lighter weight” overall than enumerateInstances();
- If the provider supports it, use the Physical Elements Count subprofile to quickly count PhysicalMedia and StorageMediaLocation instances. Note that this subprofile is optional and experimental and may not be supported by some providers.

8.2.8.22.6 Registered Name and Version

Storage Library version 1.1.0

8.2.8.22.7 CIM Server Requirements

Table 1163: CIM Server Requirements for Storage Library

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.22.8 CIM Elements

Table 1164: CIM Elements for Storage Library

Element Name	Description
Mandatory Classes	
CIM_ChangerDevice (8.2.8.22.8.1)	
CIM_Chassis (8.2.8.22.8.2)	
CIM_ComputerSystem (8.2.8.22.8.3)	'Top level' system that represents the whole Storage Library.
CIM_ComputerSystemPackage (8.2.8.22.8.4)	
CIM_ElementSoftwareIdentity (8.2.8.22.8.6)	
CIM_ElementSoftwareIdentity (8.2.8.22.8.7)	
CIM_MediaAccessDevice (8.2.8.22.8.8)	
CIM_PackagedComponent (8.2.8.22.8.9)	
CIM_PhysicalMedia (8.2.8.22.8.10)	
CIM_PhysicalMediaInLocation (8.2.8.22.8.11)	
CIM_ProtocolControllerForUnit (8.2.8.22.8.12)	
CIM_Realizes (8.2.8.22.8.13)	
CIM_SCSIProtocolController (8.2.8.22.8.14)	
CIM_SoftwareIdentity (8.2.8.22.8.15)	
CIM_StorageMediaLocation (8.2.8.22.8.17)	
CIM_SystemDevice (8.2.8.22.8.18)	This association links all **LogicalDevices** to the scoping system.
Optional Classes	
CIM_ElementCapabilities (8.2.8.22.8.5)	Class to implement the association between the top-level ComputerSystem representing a Storage Library and its StorageLibraryCapabilities
CIM_StorageLibraryCapabilities (8.2.8.22.8.16)	Describes the capabilities of the Storage Library represented by the top level ComputerSystem this is associated with

Table 1164: CIM Elements for Storage Library

Element Name	Description
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ComputerSystem	Creation of a storage library instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ComputerSystem	Deletion of a storage library instance
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_PhysicalMedia	Creation of a physical media instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_PhysicalMedia	Deletion of a physical media instance
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_MediaAccessDevice	Creation of a media access device instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_MediaAccessDevice	Deletion of a media access device instance
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ChangerDevice	Creation of a Changer Device instance
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ChangerDevice	Deletion of a Changer Device instance
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND PreviousInstance.OperationalStatus <> SourceInstance.OperationalStatus	Deprecated WQL - Change in OperationalStatus of a storage library
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_MediaAccessDevice AND PreviousInstance.OperationalStatus <> SourceInstance.OperationalStatus	Deprecated WQL - Change in OperationalStatus for a media access device
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ChangerDevice AND PreviousInstance.OperationalStatus <> SourceInstance.OperationalStatus	Deprecated WQL - Change in OperationalStatus for a Changer Device
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem AND PreviousInstance.CIM_ComputerSystem::OperationalStatus <> SourceInstance.CIM_ComputerSystem::OperationalStatus	CQL - Change in OperationalStatus of a storage library
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_MediaAccessDevice AND PreviousInstance.CIM_MediaAccessDevice::OperationalStatus <> SourceInstance.CIM_MediaAccessDevice::OperationalStatus	CQL - Change in OperationalStatus for a media access device
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ChangerDevice AND PreviousInstance.CIM_ChangerDevice::OperationalStatus <> SourceInstance.CIM_ChangerDevice::OperationalStatus	CQL - Change in OperationalStatus for a Changer Device

8.2.8.22.8.1 CIM_ChangerDevice

Class Mandatory: true

Table 1165: SMI Referenced Properties/Methods for CIM_ChangerDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
DeviceID		string	
MediaFlipSupported		boolean	
ElementName		string	
OperationalStatus		uint16[]	Status of the changer device.
Optional Properties/Methods			
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.

8.2.8.22.8.2 CIM_Chassis

Class Mandatory: true

Table 1166: SMI Referenced Properties/Methods for CIM_Chassis

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
LockPresent		boolean	
SecurityBreach		uint16	
IsLocked		boolean	
ElementName		string	
Manufacturer		string	
Model		string	
SerialNumber		string	

8.2.8.22.8.3 CIM_ComputerSystem

'Top level' system that represents the whole Storage Library.
Created By : External
Class Mandatory: true

Table 1167: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	

Table 1167: SMI Referenced Properties/Methods for CIM_ComputerSystem

Property	Flags	Type	Description & Notes
Name		string	Unique identifier for the storage library. This should take the form of a string consisting of Vendor+Product+Serial-Number, derived from SCSI Inquiry Pages.
Dedicated		uint16[]	Indicates that this computer system is dedicated to operation as a storage library
NameFormat		string	Format for Name property. **HID** is a required format. Others are optional.
OperationalStatus		uint16[]	Overall status of the library
ElementName		string	User-friendly name
Optional Properties/Methods			
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.
PrimaryOwnerContact	M	string	Contact details for storage library owner
PrimaryOwnerName	M	string	Owner of the storage library

8.2.8.22.8.4 CIM_ComputerSystemPackage

Created By : External
Class Mandatory: true

Table 1168: SMI Referenced Properties/Methods for CIM_ComputerSystemPackage

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalPackage	The PhysicalPackage
Dependent		CIM_ComputerSystem	The top-level ComputerSystem representing the Storage Library.

8.2.8.22.8.5 CIM_ElementCapabilities

Class to implement the association between the top-level ComputerSystem representing a Storage Library and its StorageLibraryCapabilities
Class Mandatory: false

Table 1169: SMI Referenced Properties/Methods for CIM_ElementCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
ManagedElement		CIM_ManagedElement	The top-level ComputerSystem representing the Storage Library
Capabilities		CIM_Capabilities	The capabilities of the Storage Library

8.2.8.22.8.6 CIM_ElementSoftwareIdentity

Class Mandatory: true

Table 1170: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_SoftwareIdentity	The software asset.
Dependent		CIM_ManagedElement	The device that uses the software.

8.2.8.22.8.7 CIM_ElementSoftwareIdentity

Class Mandatory: true

Table 1171: SMI Referenced Properties/Methods for CIM_ElementSoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Dependent		CIM_ManagedElement	
Antecedent		CIM_SoftwareIdentity	

8.2.8.22.8.8 CIM_MediaAccessDevice

Class Mandatory: true

Table 1172: SMI Referenced Properties/Methods for CIM_MediaAccessDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
DeviceID		string	
OperationalStatus		uint16[]	
NeedsCleaning		boolean	If unknown, set to False.
MountCount		uint64	
Optional Properties/Methods			
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.

8.2.8.22.8.9 CIM_PackagedComponent

Class Mandatory: true

Table 1173: SMI Referenced Properties/Methods for CIM_PackagedComponent

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PhysicalPackage	
PartComponent		CIM_PhysicalComponent	

8.2.8.22.8.10 CIM_PhysicalMedia

Class Mandatory: true

Table 1174: SMI Referenced Properties/Methods for CIM_PhysicalMedia

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
Capacity		uint64	0 = unknown. If CleanerMedia=True, then ignore Capacity value.
MediaType		uint16	
CleanerMedia		boolean	If unknown, set to False
DualSided		boolean	
LabelStates		uint16[]	
LabelFormats		uint16[]	
PhysicalLabels		string[]	
RemovalConditions		uint16	
Optional Properties/Methods			
MediaDescription		string	

8.2.8.22.8.11 CIM_PhysicalMediaInLocation

Class Mandatory: true

Table 1175: SMI Referenced Properties/Methods for CIM_PhysicalMediaInLocation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_StorageMediaLocation	
Dependent		CIM_PhysicalMedia	

8.2.8.22.8.12 CIM_ProtocolControllerForUnit

Class Mandatory: true

Table 1176: SMI Referenced Properties/Methods for CIM_ProtocolControllerForUnit

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_ProtocolController	The ProtocolController.
Dependent		CIM_LogicalDevice	The MediaAccessDevice or ChangerDevice.
Optional Properties/Methods			
DeviceNumber		string	The target device visible through the controller.

8.2.8.22.8.13 CIM_Realizes

Class Mandatory: true

Table 1177: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	
Dependent		CIM_LogicalDevice	

8.2.8.22.8.14 CIM_SCSIProtocolController

This is only required if FC Ports claim backwards compatibility with SMI-S 1.0

Class Mandatory: true

Table 1178: SMI Referenced Properties/Methods for CIM_SCSIProtocolController

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
SystemName		string	
CreationClassName		string	
DeviceID		string	Opaque identifier
OperationalStatus		uint16[]	
Optional Properties/Methods			
ElementName		string	
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.
MaxUnitsControlled		uint32	

8.2.8.22.8.15 CIM_SoftwareIdentity

Class Mandatory: true

Table 1179: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	
VersionString		string	The software of firmware version of the device (ChangerDevice, MediaAccess-Device, or a SCSIProtocolController)
Manufacturer		string	
Optional Properties/Methods			
Classifications		uint16[]	4 = Application Software, 10 = Firm-ware
BuildNumber		uint16	
MajorVersion		uint16	
RevisionNumber		uint16	

Table 1179: SMI Referenced Properties/Methods for CIM_SoftwareIdentity

Property	Flags	Type	Description & Notes
MinorVersion		uint16	

8.2.8.22.8.16 CIM_StorageLibraryCapabilities

Describes the capabilities of the Storage Library represented by the top level ComputerSystem this is associated with
Class Mandatory: false

Table 1180: SMI Referenced Properties/Methods for CIM_StorageLibraryCapabilities

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
InstanceID		string	Unique Identifier for this Capabilities class. See MOF for specific format
ElementName		string	A user-friendly name
Optional Properties/Methods			
Capabilities		uint16[]	Array of general capabilities for the Storage Library (see MOF)
MaxAuditTime		uint64	Number of seconds it takes for the library to complete an audit or "inventory" operations.

8.2.8.22.8.17 CIM_StorageMediaLocation

Class Mandatory: true

Table 1181: SMI Referenced Properties/Methods for CIM_StorageMediaLocation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
LocationType		uint16	
LocationCoordinates		string	
MediaTypesSupported		uint16[]	
MediaCapacity		uint32	

8.2.8.22.8.18 CIM_SystemDevice

This association links all **LogicalDevices** to the scoping system.
Class Mandatory: true

Table 1182: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The top-level ComputerSystem representing the Storage Library.
PartComponent		CIM_LogicalDevice	The logical devices on the Storage Library.

8.2.8.22.9 Related Standards

Table 1183: Related Standards for Storage Library

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.8.23 Element Counting Subprofile

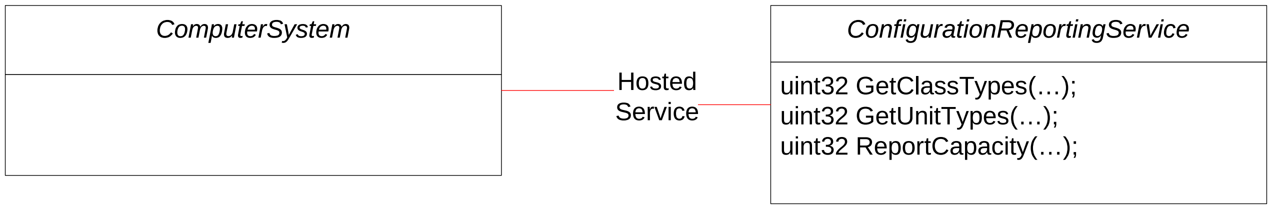
8.2.8.23.1 Description

The Element counting subprofile defines methods to count the number of physical tapes, storage media locations, and other classes within a storage library (or other system type). Such methods allow clients to avoid retrieving all *instances* of physical element classes simply to count them. Therefore, network traffic will be saved between client applications and storage library providers. These methods are modeled by the ConfigurationReportingService hosted by the storage library's (or other system type's) top-level ComputerSystem.

Instance Diagram

Figure 211: "Instance Diagram" provides a sample instance diagram.

Figure 211: Instance Diagram



Discovery

The Element counting subprofile, as currently defined, is not an advertised profile. Support for the Element Counting Subprofile can be obtained through the Storage Library Profile (or other top-level system profile as appropriate).

8.2.8.23.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.23.3 Cascading Considerations

Not defined in this standard.

8.2.8.23.4 Supported Subprofiles and Packages

The Element counting subprofile requires the Storage Library profile. Other top-level device profiles may also be able to make use of this subprofile, but such compatibility is not guaranteed.

8.2.8.23.5 Methods of the Profile

8.2.8.23.5.1 GetClassTypes

GetClassTypes returns the list of class types that a given ManagedElement – typically, a storage library's top-level ComputerSystem or Chassis – supports or has installed. Calling GetClassTypes in the first step in a three step process to obtain a count of desired elements. (See 8.2.8.23.6, "Client Considerations and Recipes" for an overview and example).

The GetClassTypes method uses the following parameters:

[IN] uint16 InquiryType = “Installed” or “Supports”

When “Installed” is specified, the method will return the list of countable classes that the associated ComputerSystem currently has installed or contained within its scope. When “Supports” is specified, the method will return the list of countable classes that the associated ComputerSystem potentially supports, though no such class instances may currently be installed or contained within its scope.

[IN] boolean Recursive = true or false

For the purposes of the current subprofile, the value of the Recursive parameter is not relevant. Until defined otherwise, clients should specify “false”, and expect that the value will not affect operation of the GetClassTypes method in any way.

[IN] CIM_ManagemedElement REF Target = a CIM object pointer to the to the top-level ComputerSystem to which ConfigurationReportingService is associated. In some cases, a pointer the ComputerSystem’s Chassis may be appropriate. This parameter reinforces that the ConfigurationReportingService is returning information on the storage library’s (or other top-level profile’s) ComputerSystem or Chassis. Classes to be returned or counted are considered to be uniquely within the scope of this top-level ComputerSystem or Chassis.

[IN (false), OUT] string ClassTypes[] = an array of class types that can be counted by the service. One value of this parameter will be selected by the client and used when calling GetUnitTypes() and ReportCapacity(), described below. The method/service provider may return a string representation of any valid CIM class which it can report a count on. For example, a storage library provider might return “CIM_PhysicalMedia” to indicate that this service allows clients to obtain a count of PhysicalMedia instances currently associated with the Target ComputerSystem or Chassis instance. Other example values would be “CIM_StorageMediaLocation” and “CIM_MediaAccessDevice”

The GetClassTypes method also returns one of the following status values:

“Success”, “Not Supported”, “Unknown”, “Timeout”, “Failed”, “DMTF Reserved”, “Vendor Specific”. In general, it is expected that “Success” will be returned on successful execution and “Failed” or “Timeout” will be returned when errors occur in executing this method on the provider/server side. If “Not Supported” is returned, the client may still attempt to call the GetUnitTypes and ReportCapacity methods, but a known value for the ClassType parameter will not be available to the client up front. “Unknown” indicates that the result cannot be determined for the given parameter combination at this time.

8.2.8.23.5.2 GetUnitTypes

GetUnitTypes returns the type of “unit” relationships that can be specified by the client when counting class instances associated with a top-level ComputerSystem or Chassis. Calling GetUnitTypes in the second step in a three step process to obtain a count of desired elements. (See 8.2.8.23.6, "Client Considerations and Recipes" for an overview and example).

The GetUnitTypes method uses many of the same parameters as GetClassTypes, including:

[IN] uint16 InquiryType: see details in 8.2.8.23.5.1, "GetClassTypes". “Supported” or “Installed” are valid enumerated values.

[IN] boolean Recursive: see details under in 8.2.8.23.5.1, "GetClassTypes". Generally, a value of “false” is expected.

[IN] CIM_ManagedElement REF Target: see details in 8.2.8.23.5.1, "GetClassTypes". A pointer to the top-level ComputerSystem associated with this ConfigurationReportingService. In some cases, a pointer to the top-level Chassis may be appropriate.

[IN] string ClassType: see details see details in 8.2.8.23.5.1, "GetClassTypes". The class type to be counted.

[IN (false) OUT] uint16 UnitTypes[] = an array of "relationship types" to help specify how the class instances to be counted are associated with the top-level ComputerSystem or Chassis specified by Target. Many values are available for UnitTypes, but clients should expect that only "Contained" or "Connected" will be returned by storage library providers. Other values, such as "None", "Front Side", and "Memory" should not be returned until future definition of their meaning is documented. Clients will use one of the values returned in this parameter when calling ReportCapacity.

The GetUnitTypes method also returns one of the following status values:

"Success", "Not Supported", "Unknown", "Timeout", "Failed", "DMTF Reserved", "Vendor Specific". In general, it is expected that "Success" will be returned on successful execution and "Failed" or "Timeout" will be returned when errors occur in executing this method on the provider/server side. If "Not Supported" is returned, the client may still attempt to call the ReportCapacity method, but a known value for the UnitType parameter will not be available to the client up front. In general, clients should attempt to specify "Contained" or "Connected" when calling ReportCapacity. "Unknown" indicates that the result cannot be determined for the given parameter combination at this time.

8.2.8.23.5.3 ReportCapacity

ReportCapacity returns the number or count of a given class types that the given ManagementElement – typically, a storage library's top-level ComputerSystem or Chassis – supports or has installed. Calling ReportCapacity in the third step in a three step process to obtain a count of desired elements. (See 8.2.8.23.6, "Client Considerations and Recipes" for an overview and example).

The ReportCapacity method uses many of the same parameters as GetClassTypes and GetUnitTypes, including:

[IN] uint16 InquiryType: see details in 8.2.8.23.5.1, "GetClassTypes". "Supported" or "Installed" are valid enumerated values.

[IN] boolean Recursive: see details in 8.2.8.23.5.1, "GetClassTypes". Generally, a value of "false" is expected.

[IN] CIM_ManagedElement REF Target: see details in 8.2.8.23.5.1, "GetClassTypes". A pointer to the top-level ComputerSystem associated with this ConfigurationReportingService. In some cases, a pointer to the top-level Chassis may be appropriate.

[IN] string ClassType: see details in 8.2.8.23.5.1, "GetClassTypes". The class type to be counted.

[IN] uint16 UnitType: see details in 8.2.8.23.5.1, "GetClassTypes". Generally, the "Contained" or "Connected" enumerated value will be used.

[IN (false), OUT] uint64 NumberOfUnits = the number of "supported" or "installed" ClassType instances "contained" or "connected" in a given Target ComputerSystem's (or Chassis's) scope. Obtaining this count is the purpose of the ConfigurationReportingService.

The ReportCapacity method also returns one of the following status values:

"Success", "Not Supported", "Unknown", "Timeout", "Failed", "DMTF Reserved", "Vendor Specific". In general, it is expected that "Success" will be returned on successful execution and "Failed" or "Timeout" will be returned when errors occur in executing this method on the provider/server side. If "Not Supported" is returned, it may indicate that the Target, ClassType, or UnitType parameters are in error. Supported values for ClassType and UnitType should be obtained by calling GetClassTypes and

GetUnitTypes prior to calling ReportCapacity. “Unknown” indicates that the result cannot be determined for the given parameter combination at this time.

8.2.8.23.6 Client Considerations and Recipes

ConfigurationReportingService may be used by clients interested in quickly obtaining a count or “number of” desired instances. For example, a client may want to know the number of PhysicalMedia instances associated with a particular storage library, but the time and overhead associated with enumerating the instances of these objects – through the extrinsic enumerateInstances() or enumerateInstanceNames() methods – can be excessive.

To use ConfigurationReportingService, clients call three methods in succession: GetClassTypes, GetUnitTypes, and ReportCapacity. GetClassTypes returns the list of class types that can be counted. This information is then used to call GetUnitTypes, which returns a list of “unit” relationships (e.g., “Connected” or “Contained”). This value and other information is then passed to ReportCapacity, which returns the count of desired class instances.

An example: A client wants to count the number of PhysicalMedia instances associated with a storage library (itself represented by a top-level ComputerSystem and Chassis instance). Having discovered a ConfigurationReportingService associated with the ComputerSystem of interest, the client will call:

```
uint32 GetClassTypes (
     InquiryType = “Installed”,
     Recursive = “false”,
     Target = CIM object path to the ComputerSystem of interest,
     &ClassTypes[] = pointer to the countable classes, as returned by the
provider/service)
```

Assuming that GetClassTypes returns a value of “Success”, the client may examine the ClassTypes[] array and find that it contains “CIM_MediaAccessDevice”, “CIM_PhysicalMedia”, “CIM_StorageMediaLocation”, and “CIM_MediaTranferDevice”. Since this client is interested in PhysicalMedia, it would use the “CIM_PhysicalMedia” value use to call GetUnitTypes:

```
uint32 GetUnitTypes (
     InquiryType = “Installed”,
     Recursive = “false”,
     Target = CIM object path to the ComputerSystem of interest,
     ClassType = “CIM_PhysicalMedia”
     &UnitTypes[] = pointer to the supported “unit” relationship types, as
returned by the provider/service)
```

Assuming that GetUnitTypes returns a value of “Success”, the client may examine the UnitTypes[] array and find that it contains only “Contained”. The client would then use this value to call ReportCapacity:

```
uint32 ReportCapacity (
     InquiryType = “Installed”,
     Recursive = “false”,
     Target = CIM object path to the ComputerSystem of interest,
     ClassType = “CIM_PhysicalMedia”,
     UnitType = “Contained”
     &NumberOfUnits)
```

Assuming that ReportCapacity returns a value of “Success”, the client should examine the NumberOfUnits value to determine the number of CIM_PhysicalMedia “contained” or currently “installed” in the Target ComputerSystem.

In general, it is expected that “Success” will be returned on successful execution of these three methods, and “Failed” or “Timeout” will be returned when errors occur in executing these methods on the provider/server side. If "Not Supported" is returned, it may indicate that the Target, ClassType, or UnitType parameters are in error.

8.2.8.23.7 Registered Name and Version

Storage Library Element Counting version 1.1.0

8.2.8.23.8 CIM Server Requirements

Table 1184: CIM Server Requirements for Storage Library Element Counting

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.23.9 CIM Elements

Table 1185: CIM Elements for Storage Library Element Counting

Element Name	Description
Mandatory Classes	
CIM_ConfigurationReportingService (8.2.8.23.9.1)	
CIM_HostedService (8.2.8.23.9.2)	

8.2.8.23.9.1 CIM_ConfigurationReportingService

Class Mandatory: true

Table 1186: SMI Referenced Properties/Methods for CIM_ConfigurationReportingService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
GetClassTypes()			
GetUnitTypes()			
ReportCapacity()			

8.2.8.23.9.2 CIM_HostedService

Class Mandatory: true

Table 1187: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The hosting Storage Library.
Dependent		CIM_Service	The configuration reporting service.

8.2.8.23.10 Related Standards

Table 1188: Related Standards for Storage Library Element Counting

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.24 InterLibraryPort Connection Subprofile

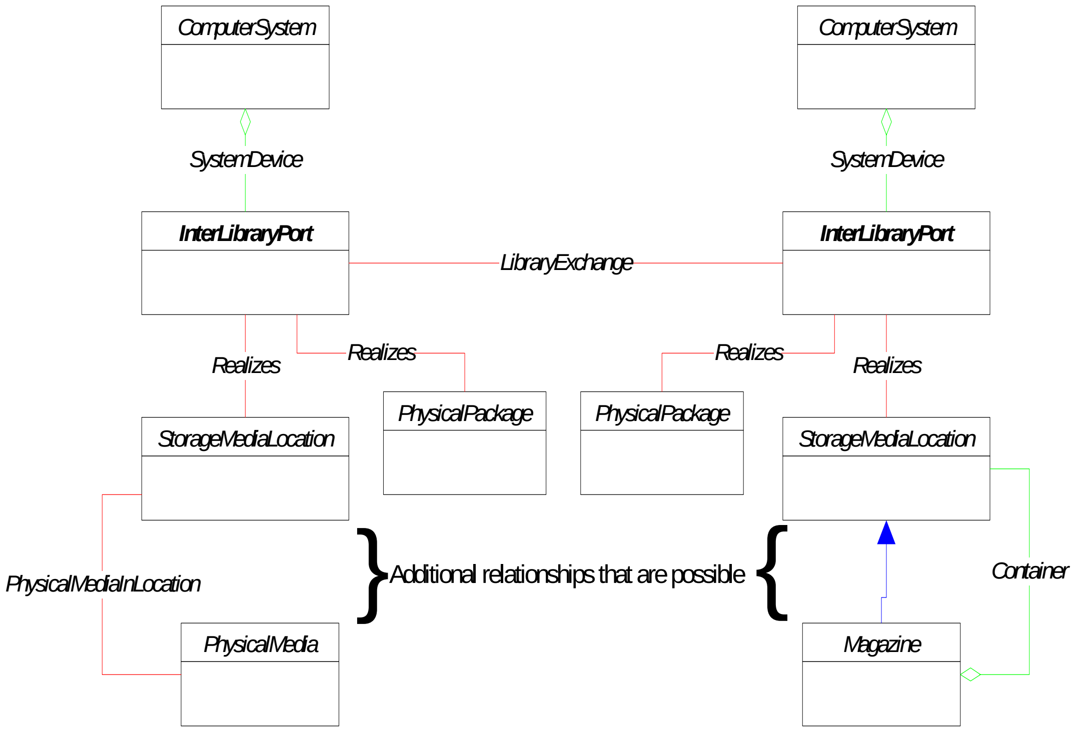
8.2.8.24.1 Description

Support of InterLibraryPort devices, a.k.a. pass-thru ports or cartridge exchange mechanisms, is designated as optional in this profile. However, when such a device exists the agent representing the library should instantiate this class for each port. When one or more libraries are connected via an Inter-Library Port and the corresponding agents are working with separate name spaces a mechanism is required for correlating the LibraryExchange association that represents the port connection.

Instance Diagrams

Figure 212: "InterLibraryPort Connection Instance Diagram" provides a sample instance diagram.

Figure 212: InterLibraryPort Connection Instance Diagram



Durable Names and Correlatable IDs

A Durable Name is not defined by this profile for InterLibraryPort instances and remains unspecified. This is not an issue when associated InterLibraryPort instances are within the same name space.

8.2.8.24.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.24.3 Cascading Considerations

Not defined in this standard.

8.2.8.24.4 Supported Subprofiles and Packages

None.

8.2.8.24.5 Methods of the Profile

None.

8.2.8.24.6 Client Considerations and Recipes

None.

8.2.8.24.7 Registered Name and Version

Storage Library InterLibraryPort Connection version 1.1.0

8.2.8.24.8 CIM Server Requirements

Table 1189: CIM Server Requirements for Storage Library InterLibraryPort Connection

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.24.9 CIM Elements

Table 1190: CIM Elements for Storage Library InterLibraryPort Connection

Element Name	Description
Mandatory Classes	
CIM_InterLibraryPort (8.2.8.24.9.1)	InterLibraryPorts represent hardware that transports Physical Media between connected Storage Libraries. The LibraryExchange association identifies the connected Libraries, by identifying the connected InterLibraryPorts.
CIM_LibraryExchange (8.2.8.24.9.2)	This relationship identifies that two storage libraries are connected through their InterLibraryPorts.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_InterLibraryPort	Creation of an instance of InterLibraryPort
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_InterLibraryPort	Deletion of an instance of InterLibraryPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_InterLibraryPort AND SourceInstance.OperationalStatus <> PreviousInstance.OperationalStatus	Deprecated WQL - Change in OperationalStatus of a InterLibraryPort
SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_InterLibraryPort AND SourceInstance.CIM_InterLibraryPort::OperationalStatus <> PreviousInstance.CIM_InterLibraryPort::OperationalStatus	CQL - Change in OperationalStatus of a InterLibraryPort

8.2.8.24.9.1 CIM_InterLibraryPort

InterLibraryPorts represent hardware that transports Physical Media between connected Storage Libraries.
The LibraryExchange association identifies the connected Libraries, by identifying the connected InterLibraryPorts.
Class Mandatory: true

Table 1191: SMI Referenced Properties/Methods for CIM_InterLibraryPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
DeviceID		string	
LastAccessed		datetime	Last access time of the port by the library
ImportCount		uint64	The number of times the port was used to move physical media into the storage library
ExportCount		uint64	The number of times the port was used to move physical media out of the storage library

Table 1191: SMI Referenced Properties/Methods for CIM_InterLibraryPort

Property	Flags	Type	Description & Notes
Direction		uint16	Identifies whether the port can be used to import physical media, export physical media or both
OperationalStatus		uint16[]	Status of the InterLibrary port.
Optional Properties/Methods			
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.

8.2.8.24.9.2 CIM_LibraryExchange

This relationship identifies that two storage libraries are connected through their InterLibraryPorts.
Class Mandatory: true

Table 1192: SMI Referenced Properties/Methods for CIM_LibraryExchange

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_InterLibraryPort	The InterLibraryPort of one storage library
Dependent		CIM_InterLibraryPort	The InterLibraryPort of the connected library

8.2.8.24.10 Related Standards

Table 1193: Related Standards for Storage Library InterLibraryPort Connection

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.25 Partitioned/Virtual Library Subprofile

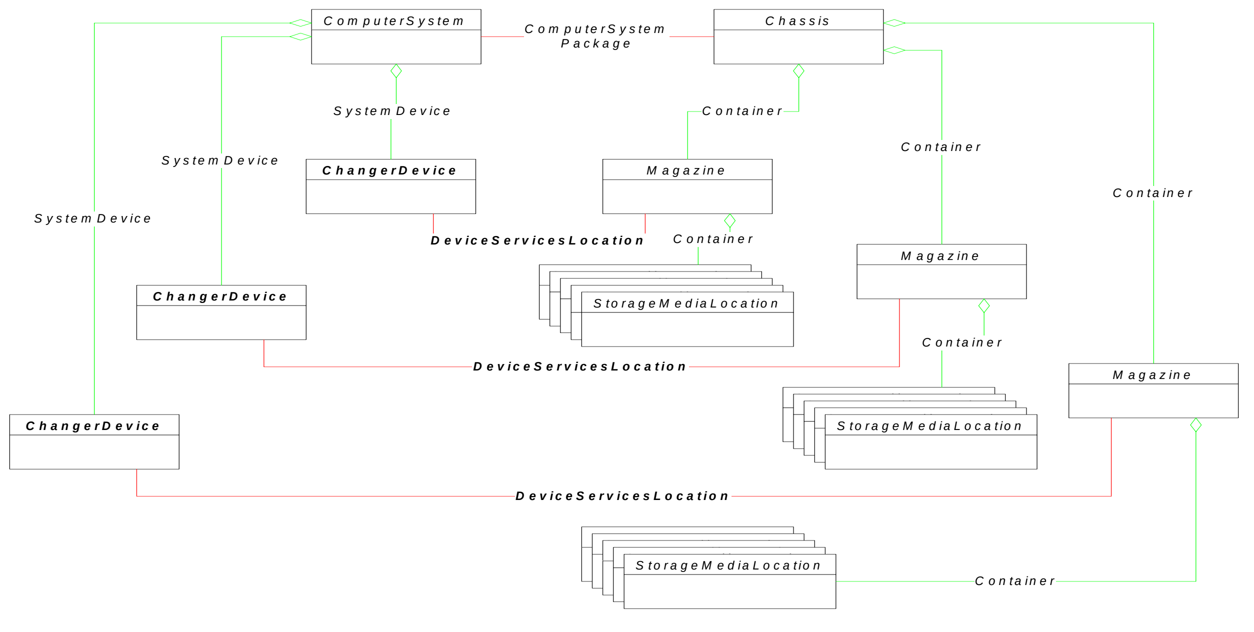
8.2.8.25.1 Description

Many libraries allow “partitioning”: the splitting up of library resources into pools used by different clients or hosts. Partitioning may also involve “virtualization”, used here to mean the representation of a single physical ChangerDevice as multiple logical ChangerDevices that can each be accessed or controlled independently. Each “virtual” ChangerDevice accesses its own group of StorageMediaLocations. No methods for configuration of partitioning, virtualization, or access control are provided in this profile. Instead, a simple model is given to allow multiple (virtual) ChangerDevices to exist within a single storage library, where each ChangerDevice can access a specific subset of pre-existing StorageMediaLocations within that storage library

Instance Diagrams

In this example, three “virtual” ChangerDevices within a single StorageLibrary have orthogonal access to three sets of Magazines or StorageMediaLocations, all contained within the Chassis (see Figure 213: "Virtual ChangerDevices").

Figure 213: Virtual ChangerDevices



8.2.8.25.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.25.3 Cascading Considerations

Not defined in this standard.

8.2.8.25.4 Supported Subprofiles and Packages

None.

8.2.8.25.5 Methods of the Profile

None.

See parent sections.

8.2.8.25.6 Client Considerations and Recipes

None.

8.2.8.25.7 Registered Name and Version

Storage Library Partitioned Library version 1.1.0

8.2.8.25.8 CIM Server Requirements

Table 1194: CIM Server Requirements for Storage Library Partitioned Library

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.25.9 CIM Elements

Table 1195: CIM Elements for Storage Library Partitioned Library

Element Name	Description
Mandatory Classes	
CIM_Container (8.2.8.25.9.1)	The containment relationship of Magazines within a Chassis or StorageMediaLocations within a Magazine.
CIM_DeviceServicesLocation (8.2.8.25.9.2)	
CIM_Magazine (8.2.8.25.9.3)	

8.2.8.25.9.1 CIM_Container

The containment relationship of Magazines within a Chassis or StorageMediaLocations within a Magazine.
Class Mandatory: true

Table 1196: SMI Referenced Properties/Methods for CIM_Container

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PhysicalPackage	The container.
PartComponent		CIM_PhysicalElement	The elements in the container.

8.2.8.25.9.2 CIM_DeviceServicesLocation

Class Mandatory: true

Table 1197: SMI Referenced Properties/Methods for CIM_DeviceServicesLocation

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_MediaTransferDevice	MediaTransferDevice that handles media from the StorageMediaLocation.
Dependent		CIM_StorageMediaLocation	The StorageMediaLocation that is serviced.

8.2.8.25.9.3 CIM_Magazine

Class Mandatory: true

Table 1198: SMI Referenced Properties/Methods for CIM_Magazine

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
LocationType		uint16	"Magazine"
LocationCoordinates		string	
MediaTypesSupported		uint16[]	
Optional Properties/Methods			
MediaCapacity		uint32	The maximum number of PhysicalMedia that this Partitioned library can hold.

8.2.8.25.10 Related Standards

Table 1199: Related Standards for Storage Library Partitioned Library

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.9	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.26 Library Capacity Subprofile

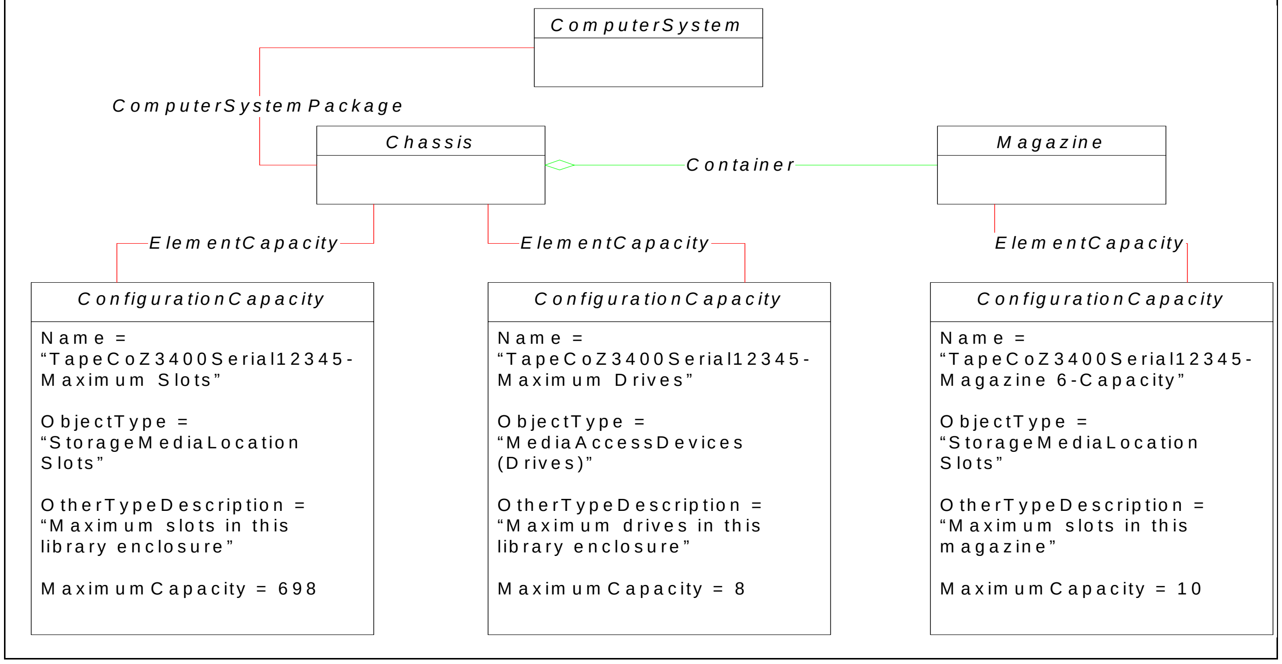
8.2.8.26.1 Description

By adding two classes (ConfigurationCapacity and ElementCapacity) servers can publish the minimum and maximum number of slots, drives, magazines, media changers, and other elements associated with a given storage library.

Instance Diagrams

Figure 214: "Library Capacity Instance Diagram" illustrates the use of ConfigurationCapacity and ElementCapacity in conjunction with the basic storage library profile.

Figure 214: Library Capacity Instance Diagram


ComputerSystem
ComputerSystemPackage
Chassis
Container
Magazine
ElementCapacity
ElementCapacity
ElementCapacity
ConfigurationCapacity
Name = "TapeCoZ3400Serial12345-Maximum Slots"
ObjectType = "StorageMediaLocation Slots"
OtherTypeDescription = "Maximum slots in this library enclosure"
MaximumCapacity = 698
ConfigurationCapacity
Name = "TapeCoZ3400Serial12345-Maximum Drives"
ObjectType = "MediaAccessDevices (Drives)"
OtherTypeDescription = "Maximum drives in this library enclosure"
MaximumCapacity = 8
ConfigurationCapacity
Name = "TapeCoZ3400Serial12345-Magazine 6-Capacity"
ObjectType = "StorageMediaLocation Slots"
OtherTypeDescription = "Maximum slots in this magazine"
MaximumCapacity = 10

8.2.8.26.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.26.3 Cascading Considerations

Not defined in this standard.

8.2.8.26.4 Supported Subprofiles and Packages

None.

8.2.8.26.5 Client Considerations and Recipes

None.

8.2.8.26.6 Registered Name and Version

Storage Library Capacity version 1.1.0

8.2.8.26.7 CIM Server Requirements

Table 1200: CIM Server Requirements for Storage Library Capacity

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	No
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.26.8 CIM Elements

Table 1201: CIM Elements for Storage Library Capacity

Element Name	Description
Mandatory Classes	
CIM_ConfigurationCapacity (8.2.8.26.8.1)	ConfigurationCapacity provides information on the minimum and maximum number of slots, drives, magazines, media changers, and other elements associated with a given storage library.
CIM_ElementCapacity (8.2.8.26.8.2)	

8.2.8.26.8.1 CIM_ConfigurationCapacity

ConfigurationCapacity provides information on the minimum and maximum number of slots, drives, magazines, media changers, and other elements associated with a given storage library.
Class Mandatory: true

Table 1202: SMI Referenced Properties/Methods for CIM_ConfigurationCapacity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Name		string	
ObjectType		uint16	Other, Processors, Power Supplies, see MOF
MinimumCapacity		uint64	
MaximumCapacity		uint64	
Optional Properties/Methods			
OtherTypeDescription		string	

8.2.8.26.8.2 CIM_ElementCapacity

Class Mandatory: true

Table 1203: SMI Referenced Properties/Methods for CIM_ElementCapacity

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Capacity		CIM_PhysicalCapacity	
Element		CIM_PhysicalElement	

8.2.8.26.9 Related Standards

Table 1204: Related Standards for Storage Library Capacity

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.27 LibraryAlert Events/Indications for Library Devices

8.2.8.27.1 Description

Historically, media libraries have been managed using both SCSI and SNMP interfaces. A number of library management standards have been defined based on these interfaces, including the "TapeAlert" error events flags. These events alert subscribing clients to current or pending error conditions related to a library, drives, or media. The SCSI implementation of TapeAlert is described in the SCSI Stream Commands (SSC-2) and SCSI Media Changer Commands (SMC-2) specifications.

In order to carry these useful asynchronous events into the WBEM/CIM domain, the TapeAlert events have been mapped into instances of the AlertIndication class. This CIM class provides a general means for communicating asynchronous events to subscribing clients and TapeAlert events/indications -- hereafter referred to more generally as "LibraryAlert" indications -- shall be specified by filling in standard values for the properties of an AlertIndication.

8.2.8.27.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.27.3 Cascading Considerations

Not defined in this standard.

8.2.8.27.4 Supported Subprofiles and Packages

None.

8.2.8.27.5 Methods of the Profile

None.

8.2.8.27.6 Client Considerations and Recipes

For all LibraryAlert indications, the following properties of AlertIndication shall be static and set to the values shown in Table 1205, "LibraryAlert Property Settings".

Table 1205: LibraryAlert Property Settings

Property Name	Property type	Property Value
Description	string	"LibraryAlert Indication"
AlertType	Uint16 (enumeration)	5 = "Device Alert"
ProabableCause	Uint16 (enumeration)	1 = "other"
Trending	Uint16 (enumeration)	1 = "Not Applicable"
SystemCreationClassName	string	"CIM_ComputerSystem"

Clients may identify a received AlertIndication as a LibraryAlert indication primarily by the value of "LibraryAlert Indication" in the Description property. The following Query attribute on an IndicationFilter instance should be provided by the agent for these alerts:

```
SELECT * FROM CIM_Alert
WHERE Description=”LibraryAlert Indication”
```

The following AlertIndication properties for LibraryAlert indications shall be vendor-specific and no specification or restriction of values is made here:

Table 1206: Vendor Specific Properties of LibraryAlert

Property Name	Property type	Property Value
OtherSeverity	string	specified by vendor
EventID	string	specified by vendor
ProviderName	string	specified by vendor

A small number of AlertIndication properties for LibraryAlert indications shall have variable values that are restricted within a small range, as follows:

Table 1207: Variable Alert Properties for LibraryAlert

Property Name	Property type	Property Value
SystemName	string	Name property value for the StorageLibrary instance that is associated with this unique indication
AlertingManagedElement	string	CIMInstance in string format for element to which this indication applies: MediaAccessDevice, StorageLibrary, or PhysicalMedia

The remaining AlertIndication properties for LibraryAlert indications shall have values derived from the SCSI TapeAlert specifications: SCSI Stream Commands (SSC-2) and SCSI Media Changer Commands (SMC-2).

Note that a small number of indications apply only to Tape libraries, while all other indications apply generically to any library type. Those indications that are tape-specific may be identified by the following strings in the OtherAlertType property:

Table 1208: SCSI TapeAlert-based Properties

Property Name	Property type	Property Value
OtherAlertType	string	"Tape snapped/cut in the drive where media can be de-mounted."
OtherAlertType	string	"Tape snapped/cut in the drive where media cannot be de-mounted."
OtherAlertType	string	"The drive is having severe trouble reading or writing, which will be resolved by a retension cycle."

The remaining AlertIndication properties and values for all LibraryAlert indications are shown in Table 1209, “LibraryAlert AlertIndication Properties”. Note that the OtherAlertType property, in particular, serves to uniquely identify each of the LibraryAlert indications.

Table 1209: LibraryAlert AlertIndication Properties

Event/Alert Summary	**AlertIndication “Mapped” Properties from SSC-2 and SMC-2 Specs**			
	OtherAlert Type	**Perceived Severity**	**ProbableCause Description**	**Recommended Action[]**
	string	**Uint16**	**string**	**string**
Read Warning	“The drive is having severe trouble reading.”	“3” = “Degraded/Warning”	“The drive is having problems reading data. No data has been lost, but there has been a reduction in the performance.”	
Write Warning	“The drive is having severe trouble writing.”	“4” = “Warning”	“Worn out Media”	“1. Discard the worn out media” “2. Use a new cleaning media”
Hard Error	“The drive had a hard read or write error.”	“5” = “Warning”	“Bad Media or Drive. The operation has stopped because an error has occurred while reading or writing data that the drive cannot correct.”	
Media	“Media can no longer be written/read, or performance is severely degraded.”	“6” = “Critical”	“Bad Media”	“1. Copy any data you require from this media.” “2. Do not use this media again.” “3. Restart the operation with a different media.”
Read Failure	“The drive can no longer read data from the storage media.”	“6” = “Critical”	“Worn out media”	“1. Replace media.” “2. Call the drive supplier help line.”
Write Failure	“The drive can no longer write data to the media.”	“6” = “Critical”	“The media is from a faulty batch or the drive is faulty: “	“1. Use known-good media to test the drive. “ “2. If the problem persists, call the media drive supplier”
Media Life	“The media has exceeded its specified life.“	“3” = “Degraded/Warning”	“The media has reached the end of its calculated useful life: “	“1. Copy any data you need to another media.” 2. Discard the old media.”

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Not Data Grade	"The cartridge is not data-grade. Any data you write to the media is at risk. Replace the cartridge with a data-grade media."	"3" = "Degraded/Warning"	"The cartridge is not data-grade. Any data you write to the media is at risk."	"Replace the cartridge with a data-grade media."
Write Protect	"Write command is attempted to a write protected media."	"6" = "Critical"	"Replace with writable media"	"You are trying to write to a write protected cartridge. Remove the write protection or use another media."
No Removal	"Manual or software unload attempted when prevent media removal is on."	"2" = "Information"	"Wait until drive is not in-use"	"You cannot eject the cartridge because the drive is in use. Wait until the operation is complete before ejecting the cartridge."
Cleaning Media	"Cleaning media loaded into drive"	"2" = "Information"	"The media in the drive is a cleaning cartridge."	"Replace this media with writeable media"
Unsupported Format	"Attempted load of unsupported media format (e.g., DDS2 in DDS1 drive)."	"2" = "Information"	"You have tried to load a cartridge of a type that is not supported by this drive."	"Insert media of a type supported by this drive"
Recoverable Snapped Tape	"Tape snapped/cut in the drive where media can be de-mounted."	"6" = "Critical"	"The operation has failed because the tape in the drive has snapped:"	"1. Discard the old tape." "2. Restart the operation with a different tape."
Unrecoverable Snapped Tape	"Tape snapped/cut in the drive where media cannot be de-mounted."	"6" = "Critical"	"The operation has failed because the tape in the drive has snapped:"	"1. Do not attempt to extract the tape cartridge." "2. Call the tape drive supplier help line."
Memory Chip In Cartridge Failure	"Memory chip failed in cartridge."	"3" = "Degraded/Warning"	"The memory in the media has failed, which reduces performance.	"Do not use the cartridge for further write operations."
Forced Eject	"Manual or forced eject while drive actively writing or reading."	"6" = "Critical"	"The operation has failed because the media was manually de-mounted while the drive was actively writing or reading."	

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Read Only Format	"Media loaded that is read-only format."	"3" = "Degraded/Warning"	"You have loaded a cartridge of a type that is read-only in this drive. The cartridge will appear as write protected."	
Directory Corrupted On Load	"Drive powered down while loaded, or permanent error prevented the directory being updated."	"3" = "Degraded/Warning"	"The directory on the cartridge has been corrupted. File search performance will be degraded. "	"The directory can be rebuilt by reading all the data on the cartridge."
Nearing Media Life	"Media may have exceeded its specified number of passes."	"2" = "Information"	"The storage media is nearing the end of its calculated life."	"1. Use another storage media for your next backup. "2. Store this storage media in a safe place in case you need to restore data from it."
Clean Now	"The drive thinks it has a head clog or needs cleaning."	"6" = "Critical"	"The drive needs cleaning:"	"1. If the operation has stopped, eject the storage media and clean the drive." "2. If the operation has not stopped, wait for it to finish and then clean the drive. Check the drive user's manual for device specific cleaning
Clean Periodic	"The drive is ready for a periodic cleaning."	"3" = "Degraded/Warning"	"The drive is due for routine cleaning:"	"1. Wait for the current operation to finish." "2. Then use a cleaning cartridge. Check the drive user's manual for device specific cleaning instructions."
Expired Cleaning Media	"The cleaning media has expired."	"6" = "Critical"	"The last cleaning cartridge used in the drive has worn out:"	"1. Discard the worn out cleaning cartridge." "2. Wait for the current operation to finish." "3. Then use a new cleaning cartridge."

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication “Mapped” Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Invalid Cleaning Media	“Invalid cleaning media type used.”	“6” = “Critical”	“The last cleaning cartridge used in the drive was an invalid type:”	“1. Do not use this cleaning cartridge in this drive.” “2. Wait for the current operation to finish.” “3. Then use a valid cleaning cartridge.”
Retension Requested	“The drive is having severe trouble reading or writing, which will be resolved by a retension cycle.”	“3” = “Information”	“The drive has requested a retension operation.”	
Dual-Port Interface Error	“Failure of one interface port in a dual-port configuration (i.e., Fibre Channel)”	“3” = “Degraded/Warning”	“A redundant interface port on the drive has failed.”	
Cooling Fan Failure	“Fan failure inside drive mechanism or drive enclosure.”	“3” = “Degraded/Warning”	“A drive cooling fan has failed.”	“Replace cooling fan or drive enclosure”
Power Supply Failure	“Redundant power supply unit failure inside the drive enclosure or rack subsystem.”	“3” = “Degraded/Warning”	“A redundant power supply has failed inside the drive enclosure.”	“Check the enclosure user’s manual for instructions on replacing the failed power supply.”
Power Consumption	“Power consumption of the drive is outside specified range.”	“3” = “Degraded/Warning”	“The drive power consumption is outside the specified range.”	
Drive Maintenance	“The drive requires preventive maintenance (not cleaning).“	“3” = “Degraded/Warning”	“Preventive maintenance of the drive is required.”	Check the drive users manual for device specific preventive maintenance tasks or call the drive supplier help line.”
Hardware A	“The drive has a hardware fault that requires reset to recover.”	“6” = “Critical”	“The drive has a hardware fault”	“1. Eject the media or magazine.” “2. Reset the drive.” “3. Restart the operation.”
Hardware B	“The drive has a hardware fault that is not read/write related or requires a power cycle to recover.”	“6” = “Critical”	“The drive has a hardware fault”	“1. Turn the drive off and then on again.” “2. Restart the operation.” “3. If the problem persists, call the drive supplier help line.”

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Interface	"The drive has identified an interface fault."	"3" = "Degraded/Warning"	"Bad cable or drive interface."	"1. Check the cables and cable connections." "2. Restart the operation."
Eject Media	"Error recovery action: Media Ejected"	"6" = "Critical"		"1. Eject the media or magazine." "2. Insert the media or magazine again." "3. Restart the operation."
Download Failure	"Firmware download failed."	"3" = "Degraded/Warning"	"The firmware download has failed because you have tried to use the incorrect firmware for this drive."	"Obtain the correct firmware and try again."
Drive Humidity	"Drive humidity limits exceeded."	"3" = "Degraded/Warning"	"Bad drive fan"	"Replace fan or drive enclosure"
Drive Temperature	"Drive temperature limits exceeded."	"3" = "Degraded/Warning"	"Bad cooling fan"	"Replace fan or drive enclosure"
Drive Voltage	"Drive voltage limits exceeded."	"3" = "Degraded/Warning"	"Bad drive power supply"	"Check the drive users manual for device specific preventive maintenance tasks or call the drive supplier help line."
Predictive Failure	"Predictive failure of drive hardware."	"6" = "Critical"		"A hardware failure of the drive is predicted. Call the drive supplier help line."
Diagnostics Required	"The drive may have a hardware fault that may be identified by extended diagnostics (i.e., SEND DIAGNOSTIC command)."	"3" = "Degrading/Warning"	"The drive may have a hardware fault."	"1. Run extended diagnostics to verify and diagnose the problem. Check the drive user's manual for device specific instructions on running extended diagnostic tests."
Loader Hardware A	"Loader mechanism is having trouble communicating with the drive."	"6" = "Critical"	"The changer mechanism is having difficulty communicating with the drive:"	"1. Turn the autoloader off then on." "2. Restart the operation." "3. If a problem persists, call the drive supplier help line."

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication “Mapped” Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Loader Stray Media	“Stray media left in loader after previous error recovery.”	“6” = “Critical”	“A media has been left in the autoloader by a previous hardware fault:”	“1. Insert an empty magazine to clear the fault.” “2. If the fault does not clear, turn the auto-loader off and then on again.” “3. If the problem per-sists, call the drive sup-plier help line.”
Loader Hard-ware B	“Loader mechanism has a hardware fault.”	“3“= “Degrading/Warn-ing”	“There is a problem with the autoloader mechanism.”	
Loader Door	“Changer door open.”	“6” = “Critical”	“The operation has failed because the autoloader door is open:”	“1. Clear any obstruc-tions from the auto-loader door.” “2. Eject the magazine and then insert it again.” “3. If the fault does not clear, turn the auto-loader off and then on again.” “4. If the problem per-sists, call the drive sup-plier help line.”
Loader Hard-ware C	“The loader mecha-nism has a hardware fault that is not mechanically related.”	“6” = “Critical”	“The autoloader has a hardware fault:”	“1. Turn the auto-loader off and then on again.” “2. Restart the opera-tion.” “3. If the problem per-sists, call the drive sup-plier help line. Check the autoloader user’s manual for device spe-cific instructions on turning the device power on and off.”
Loader Maga-zine	“Loader magazine not present.”	“6” = “Critical”	“The autoloader can-not operate without the magazine: “	“1. Insert the magazine into the autoloader.” “2. Restart the opera-tion.”

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication “Mapped” Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	**Perceived Severity**	**ProbableCause Description**	**Recommended Action[]**
	string	**Uint16**	**string**	**string**
Loader Predictive Failure	“Predictive failure of loader mechanism hardware”	“3” = “Degrading/Warning”		“A hardware failure of the changer mechanism is predicted. Call the drive supplier help line.”
Load Statistics	“Drive or library powered down with media loaded.”	“3” = “Degrading/Warning”	“Media statistics have been lost at some time in the past.”	
Media Directory Invalid at Unload	“Error preventing the media directory being updated on unload.”	“3” = “Degrading/Warning”	“The directory on the media just unloaded has been corrupted.”	“The directory can be rebuilt by reading all the data.”
Media System area Write Failure	“Write errors while writing the system area on unload.”	“6” = “Critical”	“The media just unloaded could not write its system area successfully: “	“1. Copy data to another cartridge.” “2. Discard the old cartridge.”
Media System Area Read Failure	“Read errors while reading the system area on load.”	“6” = “Critical”	“The media system area could not be read successfully at load time: “	“1. Copy data to another cartridge.”
No Start of Data	“Media damaged, bulk erased, or incorrect format.”	“6” = “Critical”	“The start of data could not be found on the media:”	“1. Check that you are using the correct format media.” “2. Discard the media or return the media to your supplier.”
Loading Failure	“The drive is unable to load the media”	“6” = “Critical”	“The operation has failed because the media cannot be loaded and threaded.”	“1. Remove the cartridge, inspect it as specified in the product manual, and retry the operation.” “2. If the problem persists, call the drive supplier help line.”
Library Hardware A	“Changer mechanism is having trouble communicating with the internal drive”	“6” = “Critical”	“The library mechanism is having difficulty communicating with the drive: “	“1. Turn the library off then on.” “2. Restart the operation.” “3. If the problem persists, call the library supplier help line.”
Library Hardware B	“Changer mechanism has a hardware fault”	“3” = “Degrading/Warning”		“There is a problem with the library mechanism. If problem persists, call the library supplier help line.”

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Library Hardware C	"The changer mechanism has a hardware fault that requires a reset to recover."	"6" = "Critical"	"The library has a hardware fault"	"1. Reset the library." "2. Restart the operation. Check the library user's manual for device specific instructions on resetting the device."
Library Hardware D	"The changer mechanism has a hardware fault that is not mechanically related or requires a power cycle to recover."	"6" = "Critical"	"The library has a hardware fault:"	"1. Turn the library off then on again." "2. Restart the operation." "3. If the problem persists, call the library supplier help line. Check the library user's manual for device specific instructions on turning the device power on and off."
Library Diagnostic Required	"The changer mechanism may have a hardware fault which would be identified by extended diagnostics."	"3" = "Degrading/Warning"	"The library mechanism may have a hardware fault."	Run extended diagnostics to verify and diagnose the problem. Check the library user's manual for device specific instructions on running extended diagnostic tests."
Library Interface	"The library has identified an interface fault"	"6" = "Critical"	"Bad cable"	"1. Check the cables and connections." "2. Restart the operation."
Failure Prediction	"Predictive failure of library hardware"	"3" = "Degrading/Warning"		"A hardware failure of the library is predicted. Call the library supplier help line."
Library Maintenance	"Library preventative maintenance required."	"3" = "Degrading/Warning"		"Preventive maintenance of the library is required. Check the library user's manual for device specific preventative maintenance tasks, or call your library supplier help line."

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication “Mapped” Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Library Humidity Limits	“Library humidity limits exceeded“	“6” = “Critical”	“Library humidity range is outside the operational conditions”	
Library Temperature Limits	“Library temperature limits exceeded”	“6” = “Critical”	“Library temperature is outside the operational conditions”	
Library Voltage Limits	“Library voltage limits exceeded”	“6” = “Critical”	“Potential problem with a power supply.”	
Library Stray Media	“Stray cartridge left in library after previous error recovery”	“6” = “Critical”	“Cartridge left in picker or drive”	“1. Insert an empty magazine to clear the fault.” “2. If the fault does not clear, turn the library off and then on again.” “3. If the problem persists, call the library supplier help line.“
Library Pick Retry	“Operation to pick a cartridge from a slot had to perform an excessive number of retries before succeeding”	“3” = “Degrading/Warning”	“There is a potential problem with the drive ejecting cartridges or with the library mechanism picking a cartridge from a slot.”	“1.Run diagnostics to determine the health of the Library.” “2. If the problem persists, call the library supplier help line.”
Library Place Retry	“Operation to place a cartridge in a slot had to perform an excessive number of retries before succeeding”	“3” = “Degrading/Warning”	“Worn cartridge or bad storage slot/magazine”	“1. No action needs to be taken at this time.” “2. If the problem persists, call the library supplier help line.”
Library Load Retry	“Operation to load a cartridge in a drive had to perform an excessive number of retries before succeeding”	“3” = “Degrading/Warning”	“Worn cartridge or picker”	“1. Run diagnostics to determine the health of the library.”
Library Door	“Library door open is preventing the library from functioning”	“6” = “Critical”	“The library has failed because the door is open:”	“1. Clear any obstructions from the library door.” “2. Close the library door.” “3. If the problem persists, call the library supplier help line.”
Library Mailslot	“Mechanical problem with import/export mailslot”	“6” = “Critical”	“There is a mechanical problem with the library media mailslot.”	“1. Check for wedged storage media in import/export mailslot”

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Library Magazine	"Library magazine not present"	"6" = "Critical"	"Administrator has removed the library's magazine"	"1. Insert the magazine into the library." "2. Restart the operation."
Library Security	"Library door opened then closed during operation"	"3" = "Degrading/Warning"	"Administrator is trying to remove or insert a storage media"	
Library Security Mode	"Library security mode changed"	"2" = "Information"	"Administrator changed security mode"	"The library security mode has been changed. The library has either been put into secure mode, or the library has exited the secure mode. This is for information purposes only. No action is required."
Library Offline	"Library manually turned offline"	"2" = "Information"	"The library has been manually turned offline and is unavailable for use."	
Library Drive Offline	"Library turned internal drive offline."	"2" = "Information"	"Drive failure"	"A drive inside the library has been taken offline. This is for information purposes only. No action is required."
Library Scan Retry	"Operation to scan the bar code on a cartridge had to perform an excessive number of retries before succeeding"	"3" = "Degrading/Warning"	"There is a potential problem with the bar code label or the scanner hardware in the library mechanism."	"1. No action needs to be taken at this time." "2. If the problem persists, call the library supplier help line."
Library Inventory	"Inconsistent media inventory"	"6" = "Critical"	"Media label has changed or bad Bar code scanner subsystem problem."	"1. Redo the library inventory to correct inconsistency." "2. Restart the operation. Check the applications user's manual or the hardware user's manual for specific instructions on redoing the library inventory."
Library Illegal Operation	"Illegal operation detected"	"3" = "Degrading/Warning"	"A library operation has been attempted that is invalid at this time."	

Table 1209: LibraryAlert AlertIndication Properties (Continued)

Event/Alert Summary	AlertIndication "Mapped" Properties from SSC-2 and SMC-2 Specs			
	OtherAlert Type	Perceived Severity	ProbableCause Description	Recommended Action[]
	string	Uint16	string	string
Dual-Port Interface Error	"Failure of one interface port in a dual-port configuration"	"3" = "Degrading/Warning"	"A redundant interface port on the library has failed."	
Cooling Fan Failure	"One or more fans inside the library have failed. Internal flag state only cleared when all flags are working again"	"3" = "Degrading/Warning"	"Bad cooling Fan"	
Power Supply	"Redundant power supply failure inside the library subsystem"	"3" = "Degrading/Warning"	"Bad Power Supply"	"A redundant power supply has failed inside the library. Check the library user's manual for instructions on replacing the failed power supply. "
Power Consumption	"Power consumption of one or more devices inside the library is outside the specified range"	"3" = "Degrading/Warning"	"The library power consumption is outside the specified range."	
Pass Through Mechanism Failure	"Error occurred in pass-through mechanism during self test or while attempting to transfer a cartridge between library modules"	"6" = "Critical"	"A failure has occurred in the cartridge pass-through mechanism between two library modules."	
Cartridge in Pass-through Mechanism	"Cartridge left in the pass-through mechanism between two library modules"	"6" = "Critical"		"A cartridge has been left in the pass-through mechanism from a previous hardware fault. Check the library users guide for instructions on clearing this fault."
Unreadable barcode Labels	"Unable to read a bar code label on a cartridge during library inventory/scan"	"2" = "Information"	"Bad Bar Code Labels or Scanner"	"The library was unable to read the bar code on a cartridge."

8.2.8.27.7 Registered Name and Version

SML_Events version 1.1.0

8.2.8.27.8 CIM Server Requirements

Table 1210: CIM Server Requirements for SML_Events

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.27.9 CIM Elements

Table 1211: CIM Elements for SML_Events

Element Name	Description
Mandatory Classes	
CIM_AlertIndication (8.2.8.27.9.1)	

8.2.8.27.9.1 CIM_AlertIndication

Class Mandatory: true

Table 1212: SMI Referenced Properties/Methods for CIM_AlertIndication

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Description		string	"LibraryAlertIndication"
AlertType		uint16	5 = "Device Alert"
ProbableCause		uint16	1 = "other"
Trending		uint16	1 = "Not Applicable"
SystemCreationClassName		string	CIM_ComputerSystem
OtherSeverity		string	Specified by vendor
EventID		string	Specified by vendor
ProviderName		string	Specified by vendor
SystemName		string	
AlertingManagedElement		string	
OtherAlertType		string	
PerceivedSeverity		uint16	
ProbableCauseDescription		string	

8.2.8.27.10 Related Standards

Table 1213: Related Standards for SML_Events

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

EXPERIMENTAL

8.2.8.28 Media Movement Subprofile

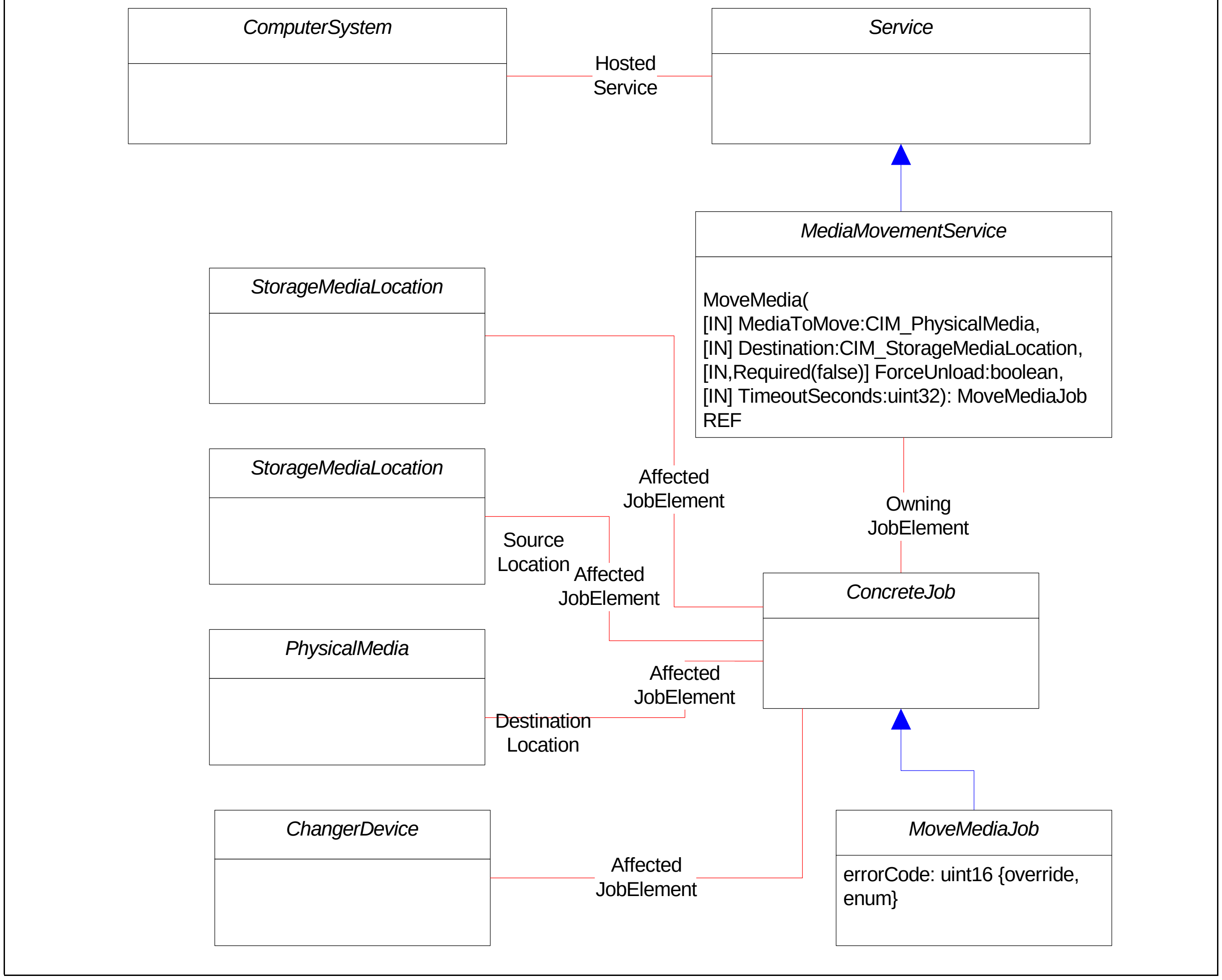
8.2.8.28.1 Description

The Media Movement Subprofile defines a method to physically move a PhysicalMedia element from its current StorageMediaLocation to another StorageMediaLocation within the library with which the media is compatible. Such a method is convenient for purposes including library maintenance, self test, and demonstration. The method is implemented by a HostedService associated with the ComputerSystem which models the storage library. The method supports asynchronous operation according to the Job Control Subprofile.

Instance Diagram: Library-centric View

Figure 215: "Storage Library Centric View" illustrates the subprofile from the library perspective.

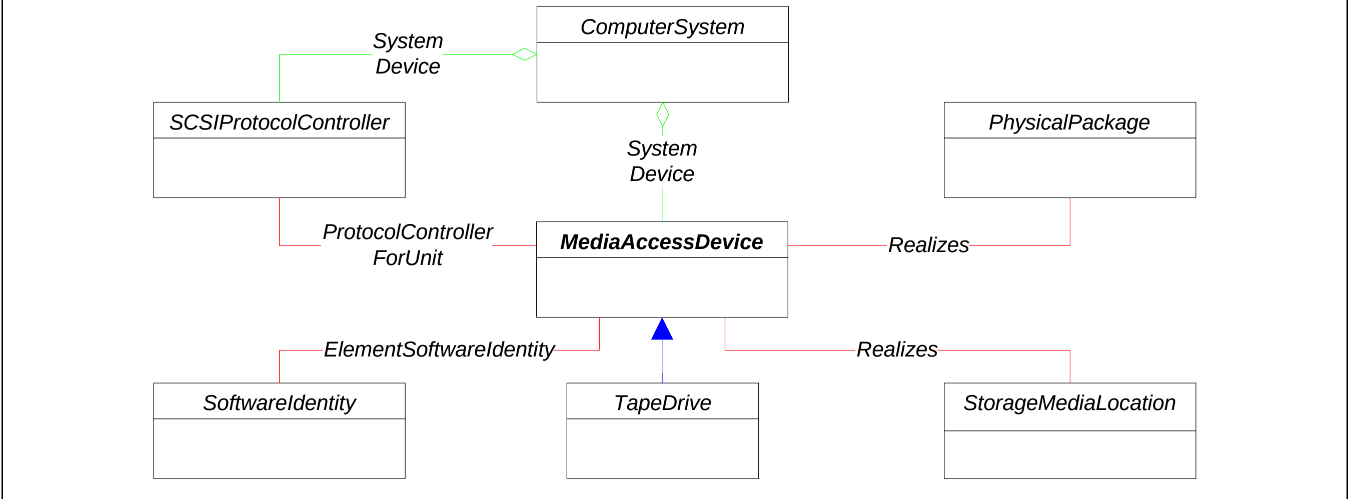
Figure 215: Storage Library Centric View



Instance Diagram: Media-centric view

When the move media operation is performed, the storage library shall physically move the medium, and then update the storage library's CIM object model. In particular, the StorageMediaInLocation association between the PhyscialMedia instance and the source StorageMediaLocation instance shall be removed and a new association made between the PhysicalMedia instance and the destination StorageMediaLocation. This is illustrated in Figure 216: "Media-centric View".

Figure 216: Media-centric View



8.2.8.28.2 Health and Fault Management Considerations

8.2.8.28.2.1 NULL Instance Handling

If a non-null instance of ConcreteJob is returned by the MoveMedia method, the implementation shall report errors which occur during the execution of the job through the ConcreteJob.GetError() method. See 8.2.1.6, "Health Package" for details.

8.2.8.28.2.2 8.1 Media Movement Subprofile Standard Messages

The standard messages specific to this profile are listed Table 1214:, "Media Movement Standard Messages".

Table 1214: Media Movement Standard Messages

Message ID	Message Name
1	Source Media not Found
2	Destination Location Full
3	Invalid Source Media
4	Invalid Destination Location
5	Media not Compatible with Destination
6	Reservation Conflict
7	Busy
8	Hardware Error
9	Internal Model Error
10	Command Sequence Error

8.2.8.28.3 Cascading Considerations

Not defined in this standard.

8.2.8.28.4 Supported Subprofiles and Packages

None.

8.2.8.28.5 Methods of the Profile

8.2.8.28.5.1 Moving a piece of PhysicalMedia

```
uint32 MoveMedia(
    [OUT, Description(“Reference to the job (may be null if job completed.)”)]
     CIM_ConcreteJob REF MoveMediaJob,
   [IN, Description( "The piece of media to be moved" ) ]
   CIM_PhysicalMedia REF MediaToMove,
   [IN, Description( "The destination location" ) ]
   CIM_StorageMediaLocation REF Destination,
     [IN, Required(false),
        Description( "Optional parameter instructing the storage library to "
         "first unload the media if it is loaded in a MediaAccessDevice." ) ]
     boolean ForceUnload,
  [IN, Required(false),
       Description( "The timeout time in seconds" ) ]
  unit32 Timeout )
```

Error returns are:

```
{ "Job Completed with No Error", "Not Supported”, "Unknown", "Timeout",
  "Failed", "Invalid Parameter", "In Use", "DMTF Reserved",
  "Method Parameters Checked - Job Started", "Busy", "Method Reserved",
  "Vendor Specific" }
```

The MoveMedia method takes as input references to the media to be moved, the destination location, and a timeout value. The method attempts to initiate a process on the Storage Library which will perform the media movement. If the process is successfully initiated, the MoveMedia returns a ConcreteJob object and an integer return code indicating the status of the job creation. If a non-null instance of ConcreteJob is returned, the instance shall be associated with an instance of MethodResult as specified by the Job Control Subprofile. See 8.2.1.7, "Job Control Subprofile" for details of job creation and execution.

Timeout parameter

The optional Timeout parameter allows the MediaMovementService process or a sub-process to handle job timeout rather than delegating the responsibility to the SMI client. If the Timeout parameter is omitted (set to “null”), the method shall use the library’s default behavior, which may be vendor or library specific.

ForceUnload parameter

When set to “true”, the optional ForceUnload parameter instructs the Storage Library to first unload the PhysicalMedia if it is loaded in a MediaAccessDevice. If the ForceUnload parameter is set to “false” and the PhysicalMedia is loaded in a MediaAccessDevice, the job shall fail and the ConcreteJob’s GetError() method shall return an instance of

Error indicating “Media Loaded in Access Device”, an error message specific to the Media Movement Subprofile. If the ForceUnload parameter is omitted (set to “null”), the method shall use the library’s default behavior, which may be vendor or library specific.

8.2.8.28.6 Client Considerations and Recipes

8.2.8.28.6.1 Concurrent library access by SMI clients and other applications.

The MoveMedia method introduces an alternate path to modify the configuration of the storage library, possibly interfering with the operation of other applications using the library concurrently. The MoveMedia method shall be used with caution in situations where applications other than the SMI client are moving media in the storage library.

8.2.8.28.6.2 Use of the ForceUnload parameter

Forcing a MediaAccessDevice to unload media while in use by other applications may cause data loss.

8.2.8.28.6.3 Job Lifecycle Indications

SMI Servers implementing the Job Control profile are required to support a set of indications which indicate transitions in the operational status of the job. In particular, an indication shall be provided when a job stops, either successfully or with an error condition. The server may also generate indications for change in job status or percent complete. See 8.2.1.7.9, "CIM Elements" of “Job Control Subprofile” for indication subscription details.

8.2.8.28.7 Registered Name and Version

Storage Library Media Movement version 1.1.0

8.2.8.28.8 CIM Server Requirements

Table 1215: CIM Server Requirements for Storage Library Media Movement

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	Yes
Indications	Yes
Instance Manipulation	Yes
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.28.9 CIM Elements

Table 1216: CIM Elements for Storage Library Media Movement

Element Name	Description
Mandatory Classes	
CIM_HostedService (8.2.8.28.9.1)	The relationship between the top-level ComputerSystem representing the Storage Library and the MediaMovementService
SNIA_MediaMovementService (8.2.8.28.9.2)	

8.2.8.28.9.1 CIM_HostedService

The relationship between the top-level ComputerSystem representing the Storage Library and the MediaMovementService
Class Mandatory: true

Table 1217: SMI Referenced Properties/Methods for CIM_HostedService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_System	The Storage Library
Dependent		CIM_Service	The MediaMovementService

8.2.8.28.9.2 SNIA_MediaMovementService

Class Mandatory: true

Table 1218: SMI Referenced Properties/Methods for SNIA_MediaMovementService

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
Name		string	
MoveMedia()			

8.2.8.28.10 Related Standards

Table 1219: Related Standards for Storage Library Media Movement

Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

EXPERIMENTAL

8.2.8.29 Limited Access Port Elements Subprofile

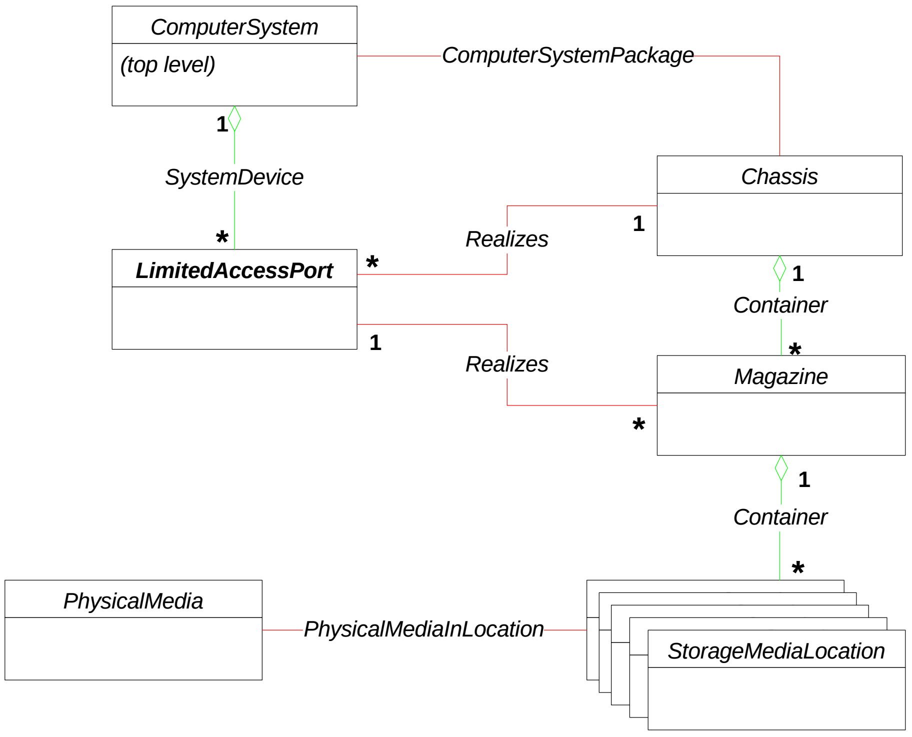
8.2.8.29.1 Description

Most libraries contain Limited Access Ports elements (a.k.a., mailslots, cartridge access ports, or import/export elements). This subprofile defines the classes necessary to publish information about these common components.

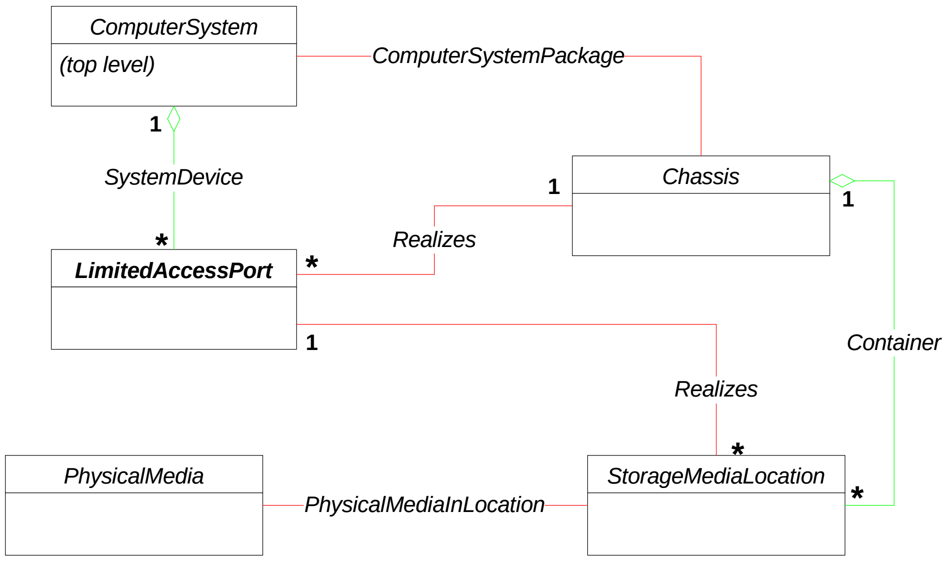
Instance Diagram

Figure 217: "LimitedAccessPort Linkages" shows the relationship between LimitedAccessPorts and other portions of the Storage Library profile.

Figure 217: LimitedAccessPort Linkages



Tape Libraries with Magazines in LimitedAccessPorts



Tape Libraries with no Magazines in LimitedAccessPorts

8.2.8.29.2 Health and Fault Management Considerations

Not defined in this standard.

8.2.8.29.3 Cascading Considerations

Not defined in this standard.

8.2.8.29.4 Supported Subprofiles and Packages

None.

8.2.8.29.5 Methods of the Profile

None.

8.2.8.29.5.1 Client Considerations and Recipes

None

8.2.8.29.6 Registered Name and Version

Storage Library Limited Access Port Elements version 1.1.0

8.2.8.29.7 CIM Server Requirements

Table 1220: CIM Server Requirements for Storage Library Limited Access Port Elements

Profile	Mandatory
Association Traversal	Yes
Basic Read	Yes
Basic Write	No
Indications	Yes
Instance Manipulation	No
Qualifier Declaration	No
Query	No
Schema Manipulation	No

8.2.8.29.8 CIM Elements

Table 1221: CIM Elements for Storage Library Limited Access Port Elements

Element Name	Description
Mandatory Classes	
CIM_Container (8.2.8.29.8.1)	The containment relationship of Magazines within a Chassis or StorageMediaLocations within a Magazine.
CIM_LimitedAccessPort (8.2.8.29.8.2)	LimitedAccessPorts represent hardware that transports physical media into or out of a Storage Library. They are identified as 'limited' since these ports do not provide access to all the PhysicalMedia or StorageMe-diaLocations in a Library, but only to a subset.
CIM_Magazine (8.2.8.29.8.3)	
CIM_Realizes (8.2.8.29.8.4)	The relationship between a LimitedAccessPort and the StorageMediaLocations, Magazines or Chassis to which it has access.
CIM_SystemDevice (8.2.8.29.8.5)	The relationship between a LimitedAccessPort and its hosting top-level ComputerSystem which represents the Storage Library.
Mandatory Indications	
SELECT * FROM CIM_InstCreation WHERE SourceIn-stance ISA CIM_LimitedAccessPort	Creation of an instance of LimitedAccessPort
SELECT * FROM CIM_InstDeletion WHERE SourceIn-stance ISA CIM_LimitedAccessPort	Deletion of an instance of LimitedAccessPort
SELECT * FROM CIM_InstModification WHERE Sour-ceInstance ISA CIM_LimitedAccessPort AND SourceInstance.OperationalStatus <> PreviousIn-stance.OperationalStatus	Deprecated WQL - Change in OperationalStatus of a LimitedAccessPort
SELECT * FROM CIM_InstModification WHERE Sour-ceInstance ISA CIM_LimitedAccessPort AND SourceInstance.CIM_LimitedAccessPort::Opera-tionalStatus <> PreviousIn-stance.CIM_LimitedAccessPort::OperationalStatus	CQL - Change in OperationalStatus of a LimitedAccess-Port

8.2.8.29.8.1 CIM_Container

The containment relationship of Magazines within a Chassis or StorageMediaLocations within a Magazine.
Class Mandatory: true

Table 1222: SMI Referenced Properties/Methods for CIM_Container

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_PhysicalPackage	The container.
PartComponent		CIM_PhysicalElement	The elements in the container.

8.2.8.29.8.2 CIM_LimitedAccessPort

LimitedAccessPorts represent hardware that transports physical media into or out of a Storage Library.
They are identified as 'limited' since these ports do not provide access to all the PhysicalMedia or StorageMediaLo-cations in a Library, but only to a subset.

Class Mandatory: true

Table 1223: SMI Referenced Properties/Methods for CIM_LimitedAccessPort

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
SystemCreationClassName		string	
CreationClassName		string	
SystemName		string	
DeviceID		string	
Extended		boolean	When true, the port's StorageMediaLocations are accessible to a human operator. When false, the StorageMediaLocations are accessible to a PickerElement.
ElementName		string	User-friendly name
OperationalStatus		uint16[]	Status of the LimitedAccessPort.
Optional Properties/Methods			
StatusDescriptions		string[]	Additional information related to the values in OperationalStatus.

8.2.8.29.8.3 CIM_Magazine

Class Mandatory: true

Table 1224: SMI Referenced Properties/Methods for CIM_Magazine

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
CreationClassName		string	
Tag		string	
LocationType		uint16	"Magazine"
LocationCoordinates		string	
MediaTypesSupported		uint16[]	
MediaCapacity		uint32	The maximum number of PhysicalMedia that this StorageMediaLocation can hold.
Optional Properties/Methods			
PhysicalLabels		string[]	
LabelStates		uint16[]	
LabelFormats		uint16[]	

8.2.8.29.8.4 CIM_Realizes

The relationship between a LimitedAccessPort and the StorageMediaLocations, Magazines or Chassis to which it has access.

Class Mandatory: true

Table 1225: SMI Referenced Properties/Methods for CIM_Realizes

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
Antecedent		CIM_PhysicalElement	The StorageMediaLocation, Magazines or Chassis to which the LimitedAccessPort has access.
Dependent		CIM_LogicalDevice	The LimitedAccessPort.

8.2.8.29.8.5 CIM_SystemDevice

The relationship between a LimitedAccessPort and its hosting top-level ComputerSystem which represents the Storage Library.
Class Mandatory: true

Table 1226: SMI Referenced Properties/Methods for CIM_SystemDevice

Property	Flags	Type	Description & Notes
Mandatory Properties/Methods			
GroupComponent		CIM_System	The Storage Library.
PartComponent		CIM_LogicalDevice	The LimitedAccessPort.

8.2.8.29.9 Related Standards

Table 1227: Related Standards for Storage Library Limited Access Port Elements

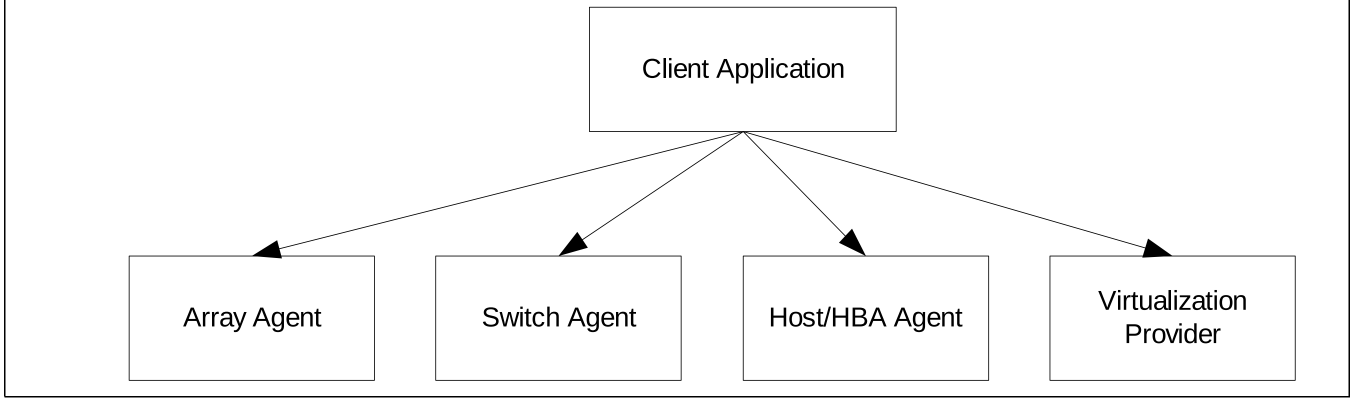
Specification	Revision	Organization
CIM Infrastructure Specification	2.3.0	DMTF
CIM Operations over HTTP	1.2.0	DMTF
CIM Schema	2.11.0	DMTF

8.3 Cross Profile Considerations

8.3.1 Overview

Many applications access data from multiple profiles to perform operations. This section describes algorithms that can be used to associate objects from different profiles to understand connections between the profiles. The algorithms use Durable Names to match objects from different profiles. Below are simplified instance diagrams that are used to illustrate the algorithms.

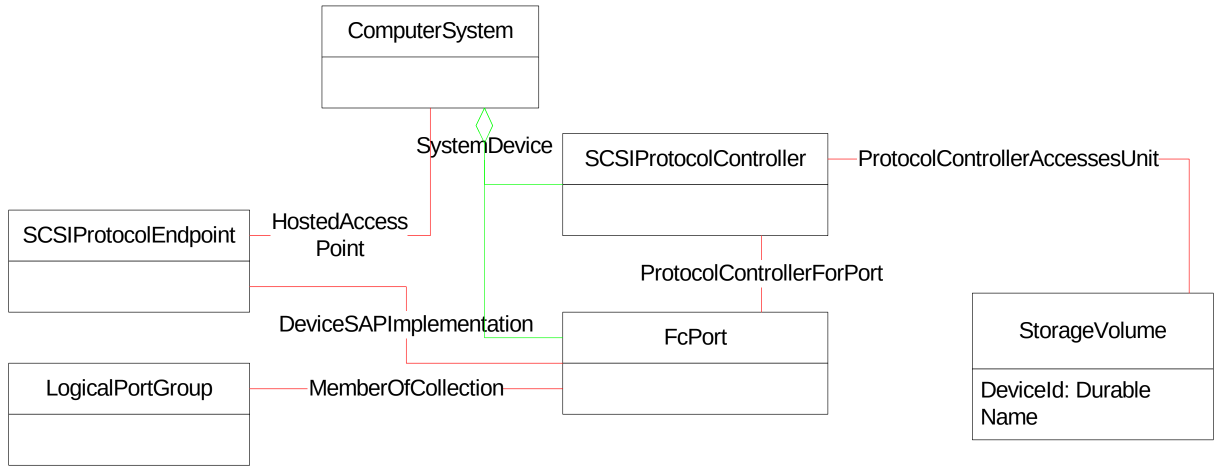
Figure 218: System Diagram



8.3.2 HBA model

This model represents a simple “Host Bus Adapter”. The model includes objects that represent a single port Fibre channel HBA. The model also includes a storage volume being accessed through the HBA.

Figure 219: Host Bus Adapter Model



8.3.2.1 Recipes

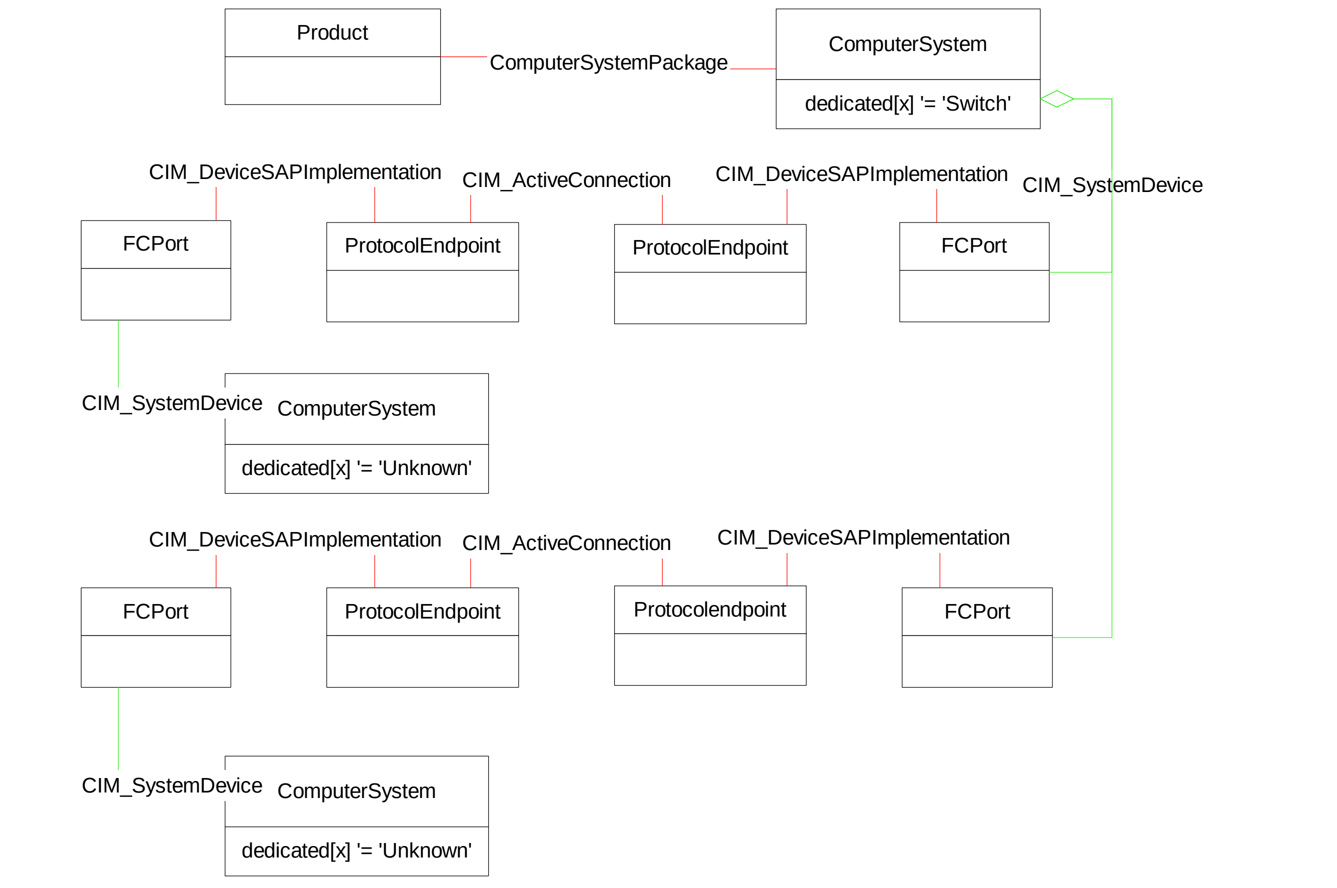
8.3.2.1.1 Disclaimer

The recipes in this section are included for illustrative purposes only. As of this version of SMI-S, these recipes are not part of CTP and may not have been validated.

8.3.3 Switch Model

This model represents a two-port Fibre channel switch. The model also includes objects representing links to remote ports the switch agent knows about, and ComputerSystems

Figure 220: Switch Model



8.3.3.1 Recipes

8.3.3.1.1 Create MAP

```
// DESCRIPTION
// Create a map of how elements in a SAN are connected together via Fibre-Channel
ports
//
// The map is built in array $attachedFcPorts->[], where the index is a
// WWN of any device port on the SAN, and the value at that index is
// the object path of the connected switch port.
//
// First find all the switches in a SAN.  Get all the FCPorts for each
// switch and get the Attached FCPorts for each Switch FCPort.  Save
// these device ports in the map described above.

// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1.  All agents/namespaces supporting Fabric Profile previously identified using
SLP

// Do this for each CIMOM supporting Fabric Profile

switches[] = enumerateInstances(“CIM_ComputerSystem”, true, false, true, true,
null)

for #i in $switches[]
{
    if (!contains(5, $switches[#i].Dedicated))
        continue // only process switches, not other computer systems

    $fcPorts->[] = AssociatorNames(
        $switches[#i].getObjectPath(),
        “CIM_SystemDevice”,
        “CIM_FCPort”,
        “GroupComponent”,
        “PartComponent”)

    for #j in $fcPorts->[]
    {
        $protocolEndpoints->[] = AssociatorNames(
             fcPorts->[#j],
             “CIM_DeviceSAPImplementation”,
             “CIM_ProtocolEndpoint”,
             “Antecedent”,
             “Dependent”);

       // NOTE - It is possible for this collection to be empty (ports that are not
        // connected).  It is NOT possible for this collection to have more than
one
        // element
        if ($protocolEndpoints->[].length == 0)
            continue

        $attachedProtocolEndpoints->[] = AssociatorNames(
            $protocolEndpoints->[0],
         “CIM_ActiveConnection”,
            “CIM_ProtocolEndpoint”,
            null, null) // NOTE: role & resultRole are null as the
                        // direction of the association is not
                        // dictated by the specification
        for #k in $attachedProtocolEndpoints->[] {
            // $attachedFcPort is either a device port or an ISLÂ’d
         // switch port from another switch. We store this result
         // (i.e. which device FCPort is connected to which switch
         // FCPort) in a suitable data structure for subsequent
         // correlation to ports discovered on devices.
            $attachedFcPorts->[] = Associators(
        $attachedProtocolEndpoints->[#k],
        “CIM_DeviceSAPImplementation”,
        “CIM_FCPort”,
                “Dependent”,
                “Antecedent”,
                false,
                false,
                [“PermanentAddress”])

           $attachedFcPort = $attachedFcPorts[0] // Exactly one member guaranteed
by model
            #wwn = $attachedFcPort.PermanentAddress
         $attachedFcPorts->[#wwn] = $fcPorts->[#j]
        }
    }
}
```

8.3.3.1.2 HBA to Switch Physical Path

```
// DESCRIPTION
// Determine physical path from HBA to switch.
//
// For each HBA port on every host, determine the connected switch
// port.  NOTE: Not every HBA port will be connected to a switch port,
// and not every switch port will be connected to a device port.  Only
// the connections between HBA ports and switch ports are discovered
// by this recipe
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.  All agents/namespaces supporting HBA Profile previously identified using
SLP
// 2.  Array $attachedFcPorts->[] is a map of how elements in a SAN are
// connected together via Fibre-Channel ports. Each index is a WWN of
// any device port on the SAN, and the value at that index is the
// connected switch port.

// Do this for each CIMOM supporting HBA Profile

$hosts[] = enumerateInstances(“CIM_ComputerSystem”)
```

```
for #i in $hosts->[]
{
    if (!contains(0, $hosts[#i].Dedicated))
        continue // only process systems that are “not dedicated”

    $fcPorts[] = Associators(
        $hosts[#i].getObjectPath(),
        “CIM_SystemDevice”,
        “CIM_FCPort”,
        “GroupComponent”,
        “PartComponent”,
        false,
        false,
        [“PermanentAddress”])

    for #j in $fcPorts[]
    {
        // Get the FCPort WWN
        #wwn = $fcPorts[#j].PermanentAddress

        // Match this device port WWN to one (or less) switch
        // ports, by using the mapping table
        $attachedSwitchPort-> = $attachedFcPorts->[#wwn]

       // Note - if there is no entry in the mapping array, this
       // port is not connected to any switch
    }
}
```

8.3.3.1.3 Array to Switch Physical Path

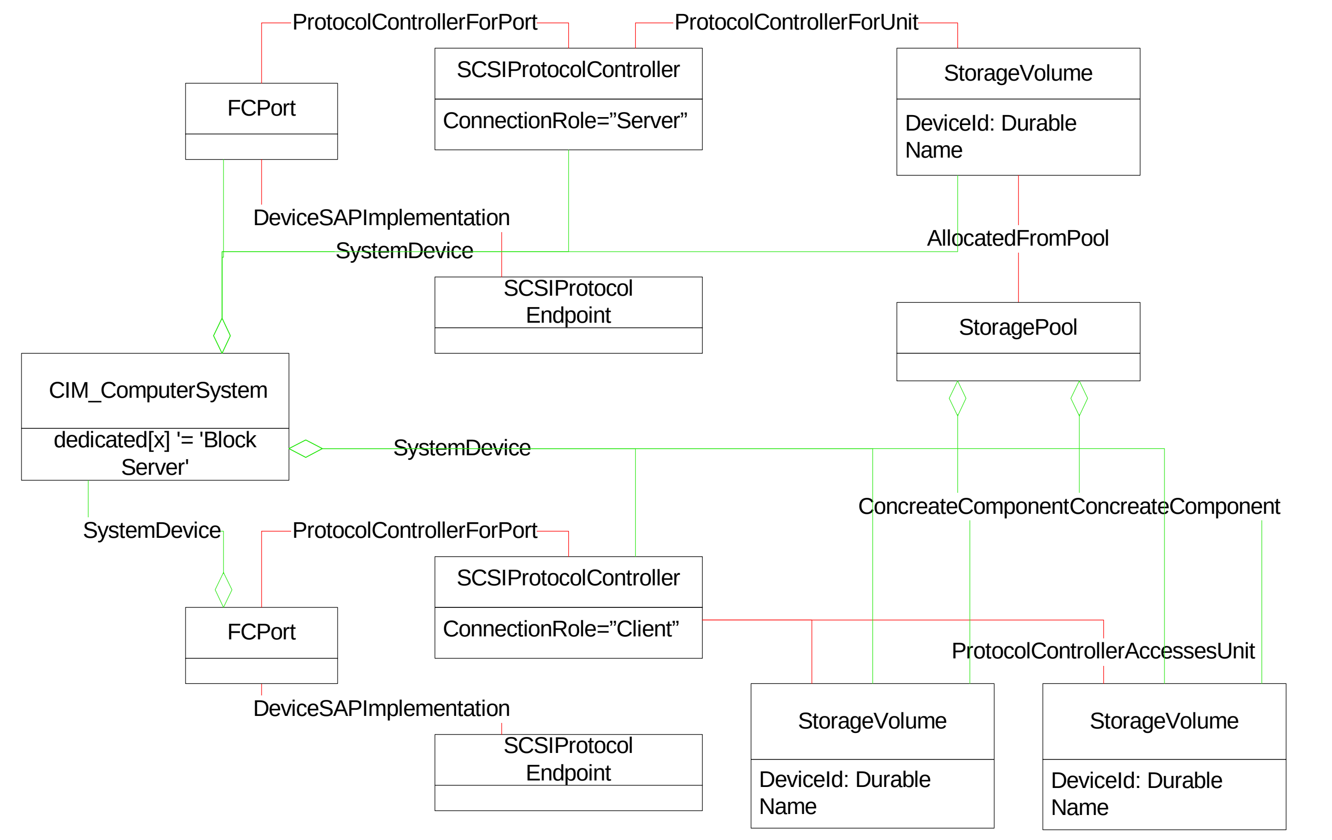
```
// DESCRIPTION
// Determine physical path from Storage Arrays to Switches
//
// For each fibre-channel port on every array, determine the connected
// switch port.  NOTE: This identifies the FrontEnd I/O Controllers
// (and Storage Arrays) whose ports are physically connected to
// some of the ports of some of the Switches. This recipe does not
// distinguish and does not filter the front-end FC Port from the
// back-end FC Ports.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1.  All agents/namespaces supporting Array Profile previously identified using
SLP
```

```
// 2.  Array $attachedFcPorts[] is a map of how elements in a SAN are
// connected together via Fibre-Channel ports. Each index is a WWN of
// any device port on the SAN, and the value at that index is the
// connected switch port.

// Do this for each CIMOM supporting the Array Profile

$storageArrays[] = enumerateInstances(“CIM_ComputerSystem”);


// NOTE: Some of the ports contained will be back-end ports, but they will
//   have no connectivity to switches, so we won’t distinguish them
//   from unconnected front-end ports

for #i in $storageArrays[]
{
    if (!contains(3, $storageArrays[#i].Dedicated))
        continue // only process systems that are dedicated “storage”

    if (!contains(15, $storageArrays[#i].Dedicated))
        continue // only process systems that are dedicated “block server”

    $fcPorts[] = Associators(
        $storageArrays[#i].getObjectPath(),
     “CIM_SystemDevice”,
     “CIM_FCPort”,
        “GroupComponent”,
        “PartComponent”,
        false,
        false,
        [“PermanentAddress”])

    for #j in $fcPorts[]
    {
     // Get the FCPort WWN
     #wwn = $fcPorts[#j].PermanentAddress

     // Match this device port WWN to one (or less) switch
     // ports, by using the mapping table

        $attachedSwitchPort-> = $attachedFcPorts->[#wwn]

       // Note - if there is no entry in the mapping array, this
       // port is not connected to any switch
    }
}
```

8.3.4 Array Model

This is a simple model of a disk array. The array has a single controller with a single Fibre channel port on the front end and a single parallel SCSI port for the disks. The model shows two disks that are members of a single redundancy group. Part of the redundancy group is made available over the Fibre channel as a single volume.

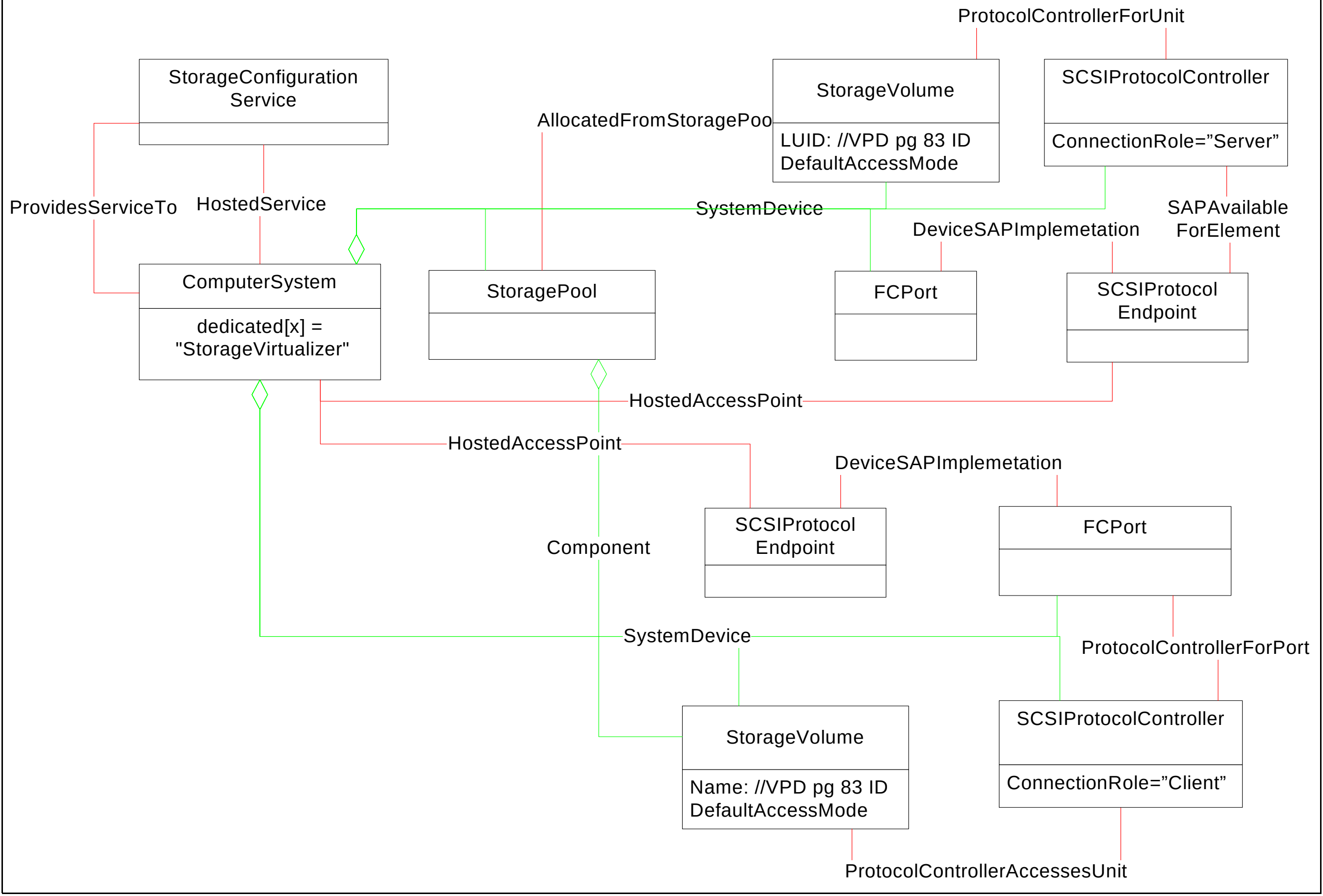
Figure 221: Array Instance



8.3.4.1 Storage Virtualization Model

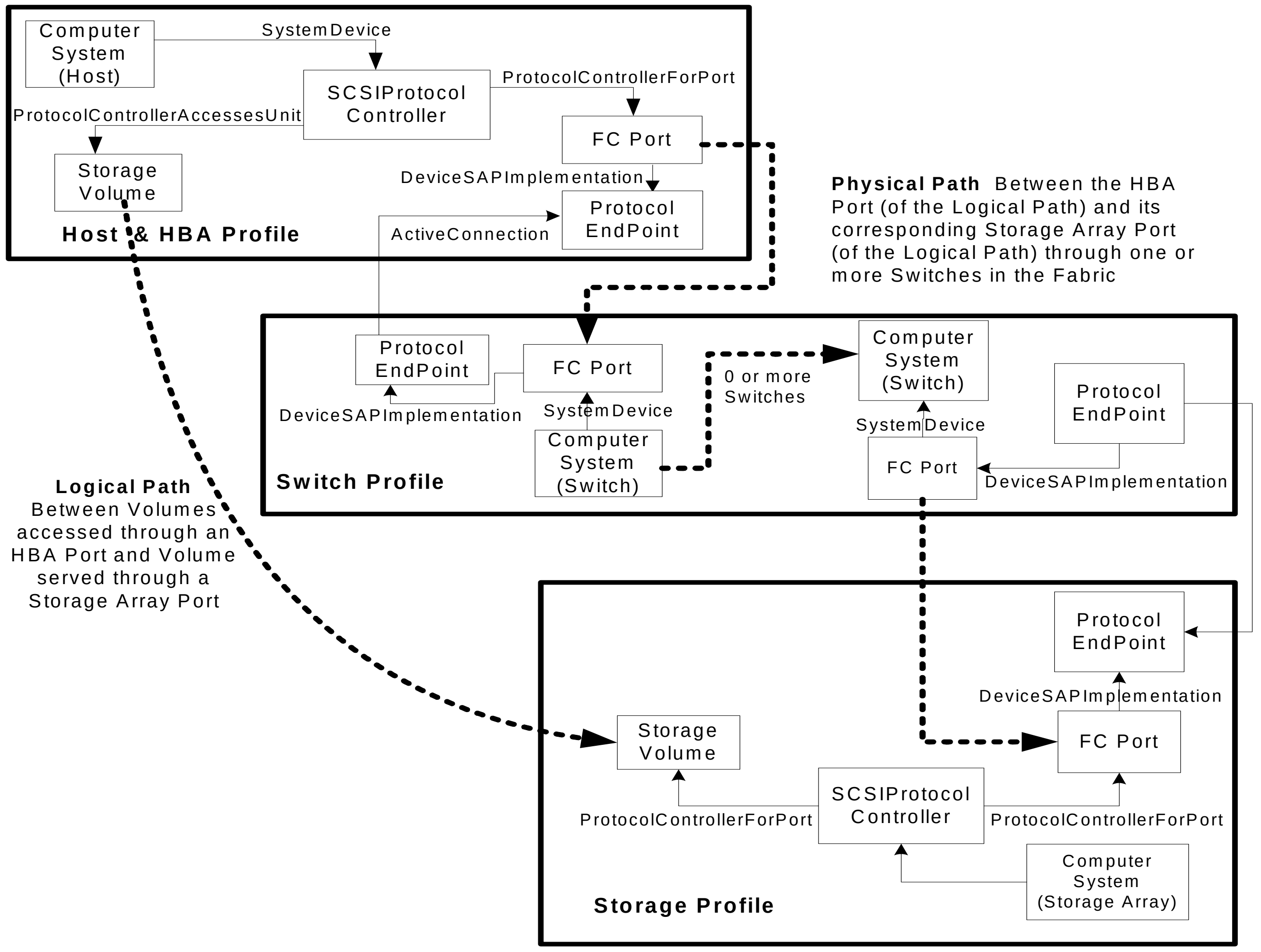
This is a simple model of a Storage Virtualizer. The model shows the basic controller and pool. The model also shows a single volume being used and a single volume being served to a host.

Figure 222: Virtualization Instance



8.3.5 Fabric Topology (HBA, Switch, Array)

Figure 223: Logical and Physical Topology Across Components of a Fabric



A map of a SAN that shows all the elements and the connections between them is very useful. To create the map all the elements in the SAN with their Fibre channel ports are first located. Next the ports are linked together.

To locate all the elements in a SAN, you start by locating the agents. SMI-S agents are located using SLP. Once the agents are located, intrinsic methods are used to enumerate ComputerSystem objects. Each ComputerSystem object represents an element in the SAN. The ComputerSystem object's "Dedicated" attribute can be used to identify the type of the element.

After the elements are located, Fibre channel ports for each element are discovered. For each ComputerSystem object follow SystemDeviceFCPort objects and ProtocolController objects. For each ProtocolController object follow the ProtocolControllerForPort associations to FCPort objects. Use the information in the FCPort objects found to determine the Durable Name for the FCPort object. The Durable Name is used to match the ports to objects in other profiles.

Now to link the elements' ports together find the Switch elements. Switches know about ports on elements logged into their ports. To find this information start by locating the ComputerSystem objects that represents switches. Switches can be identified by the "Dedicated" attribute of the ComputerSystem object being set to "Switch". For each switch follow the SystemDeviceFCPort objects that represent the ports of the switch. Next look for ActiveConnection ActiveConnectionFCPort objects. These FCPort objects represent the ports on the other side of a link. Use attributes from the FCPort object to determine the Durable Name. These identifiers are then matched to identifiers found in other profiles to complete the connections.

8.3.5.1 Overview

The Logical Disk Composition Recipe traces the objects and associations that make up a LogicalDisk across profile boundaries. It serves performance and fault identification use-cases by allowing the user to map out all the objects in the I/O stack that may contribute to the storage services a LogicalDisk provides to applications. It covers the Disk Partition, Volume Manager, Disk,Multipath, Common Initiator, Fabric, iSCSI Target, Storage Virtualizer and Array profiles and subprofiles. This recipe also shows how Correlatable Naming conventions may be used to identify and correlate instances of objects within, and across profiles.

8.3.5.2 Main Recipe

```
Logical Disk Composition Recipe

This main recipe is profile-independent.  It
uses subroutines which are profile-dependent.

Description:

By stitching together information from
multiple profiles, determine the logical composition
a host LogicalDisk in terms of its constituent
LogicalDevices, ProtocolControllers, Ports, StoragePools,
etc. and the associations between them.
Collect sufficient information to allow a graph to be drawn.
Where possible, allow network topologies to be attached.

Preexisting Conditions and Assumptions:

That all providers relevant to the logical composition
of the disk have been discovered (see the Server Profile
recipe “Find Servers Supporting a Given Profile),
can be queried for the information they have to contribute,
and follow SMI-S 1.1

Durable Names naming conventions to allow stitching
across profiles and providers. Correlatable, unique
and durable names are assumed if this is to work.
In particular, this must be true of instances of:

CIM_LogicalDisk
CIM_FCPort
CIM_SCSIProtocolEndpoint

Which are node objects, included in multiple profiles.
```

```
Subroutines:

Each subroutine of this recipe has access to all
providers relevant to the path under consideration.


// Add $node to $nodes[] if it has not already been added.
// If a new node was added, set #new_added to true.
sub AddIfNotAlreadyAdded(IN     CIM_LogicalElement   $node,
                         IN/OUT CIM_LogicalElement[] $nodes[],
                         OUT boolean #new_added,
                         OUT int #error_code);

// Add $link to $links[] if it has not already been added.
// If a new link was added, set #new_added to true.
sub AddLinkIfNotAlreadyAdded(IN     CIM_Dependency   $link,
                             IN/OUT CIM_Dependency[] $links[],
                             OUT boolean #new_added,
                             OUT int #error_code);

// Compare two LogicalElement references to determine if
// they represent the same modeled object. The two nodes
// may come from entirely different providers/profiles.
// This method uses correlatable naming conventions defined
// for the classes in question by the Profiles/SubProfiles.
sub RepresentsTheSameObject(IN     CIM_LogicalElement   $node1,
                            IN     CIM_LogicalElement   $node2,
                            OUT int #error_code);

// Compare two Dependency references to determine if
// they represent the same modeled association. The two links
// may come from entirely different providers/profiles.
// This method uses correlatable naming conventions defined
// for the endpoints in question by the Profiles/SubProfiles.
sub RepresentsTheSameAssociation(IN     CIM_Dependency   $link1,
                                 IN     CIM_Dependency   $link2,
                                 OUT int #error_code) {

// Given the Names and NameFormats of two object instances,
// determine if the two instances represent the same modeled object
// unambiguously according to Correlatable names semantics.
// Return true only if the match is unambiguous.
sub MatchUnambiguouslyByNameNameFormat(string name1,
                                       string nameFormat1,
                                       string name2,
                                       string nameFormat2
```

```
                                    );

// Given the IdentifyingDescriptions and OtherIDentifyingInfo
// arrays of two object instances,
// determine if the two instances represent the same object
// unambiguously according to Correlatable names semantics
// for ComputerSystem names.
// Return true only if the match is unambiguous.
sub MatchUnambiguouslyByIdentifyingInfo(string info1[],
                                        string desc1[],
                                        string info2[],
                                        string desc2[]
                                       );

// Given an instance of an object from one provider,
// find the instance of the same object in the current
// provider.  Return null if not found.
sub GetProviderInstanceOf(IN CIM_LogicalElement  $that_node,
                          OUT CIM_LogicalElement $this_node,
                          OUT int #error_code);



// In this function, the layer is passed references
// to the working graph.  It is expected that the layer
// will search the structures for objects it recognizes
// and can add new objects and associations to the graph.
// If the layer does not exist or does not recognize
// any of the objects or associations in the graph
// as objects it manages or knows about, it adds nothing.
// Set #new_added to true if the layer contributed anything new to
// contribute to the graph.

sub AddToGraphFromLayerXXX( IN/OUT CIM_LogicalElement $nodes[],
                            IN/OUT CIM_Dependency     $links[],
                            OUT boolean #new_added,
                            OUT int #error_code
                          );

// This fictitious function would draw a node in
// this logical composition on a canvas.  The net effect
// of drawing all the nodes would be
// an arrangement of boxes containing CIM class names
// and identifiers of those objects.
sub DrawNode($node);

// This fictitious function would draw a line representing
// the specified association between two nodes.  The net
// result would be a graph directed graph of the nodes
// with their associations.
sub DrawLinkBetweenNodes($link);


Main Recipe:

// Begin with a CIM_LogicalDisk reference representing a
// volume on which a filesystem has been placed or is
// being used “raw” by an application managing its own
// block structures.

$logicaldisk;

// The goal is to build two arrays: An array of objects
// representing nodes in the logical topology graph, and
// an array of Associations linking those objects to form
// a directed graph.

CIM_LogicalElement[] $nodes[];
CIM_Dependency[]     $links[];

// Define some other flow control variables.
boolean #new_objects_added = true;
int #error_code = 0;

// Start by adding the top level volume.

$node[0] = $logicaldisk;
#new_objects_added = true;

// Now, build down through the layers building what
// should be a breadth-first traversal of the tree graph.
// Repeatedly cycle through the layers until no new objects
// have been added. This allows for multiple layers of
// virtualization and network to kick in if new objects are found
// from the layers above added in previous passes.

while (#new_objects_added) {

   boolean #added;
   #new_objects_added = false;

   &AddToGraphFromLayerVolumeManager($nodes[],
                                     $links[],
                                     #added,
                                     #error_code);
```

```
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerDiskPartitioning($nodes[],
                                     $links[],
                                     #added,
                                     #error_code);
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerLocalDiskDrive($nodes[],
                                   $links[],
                                   #new_objects_added,
                                   #error_code);
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerMultipath($nodes[],
                              $links[],
                              #added,
                              #error_code);
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerCommonInitiator($nodes[],
                                    $links[],
                                    #added,
                                    #error_code);
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerFabric($nodes[],
                           $links[],
                           #added,
                           #error_code);
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerIPNetwork($nodes[],
                              $links[],
                              #added,
                              #error_code);
#new_objects_added |= #added;
if (0 != #error_code) { return #error_code; }

&AddToGraphFromLayerStorageVirtualizer($nodes[],
                                       $links[],
                                       #added,
                                          #error_code);
   #new_objects_added |= #added;
   if (0 != #error_code) { return #error_code; }

   &AddToGraphFromLayerArray($nodes[],
                             $links[],
                             #added,
                             #error_code);
   #new_objects_added |= #added;
   if (0 != #error_code) { return #error_code; }

} // while.

// Now “draw” the logical disk composition. In reality these functions
// would need to rather sophisticated with geometric constraints
// to draw a nice looking graph.


for #i in $nodes[] {
   &DrawNode($nodes[#i]);
}

for #i in $links[] {
   &DrawLinkBetweenNodes($link[#i]);
}


sub AddIfNotAlreadyAdded(IN     CIM_LogicalElement   $node,
                         IN/OUT CIM_LogicalElement[] $nodes[],
                         OUT boolean #new_added,
                         OUT int #error_code) {
    boolean #wasFound = false;
    boolean #new_added = false;

    // Search through the nodes looking for a match.
    // Not a particularly efficient way of doing it, but functional.
    for #i in $nodes[] {
        if (&RepresentsTheSameObject($node, $nodes[i], #error_code)) {
            if (#error_code == 0) {
               #wasFound = true;
            }
            break;
        }
    }

    // If we did not find a match, and there were no errors, add it.
    if ( (!#wasFound) && (#error_code == 0)) {
        $nodes[].add($node);
        #new_added = true;
    }
} // AddIfNotAlreadyAdded.

sub AddIfNotAlreadyAdded(IN     CIM_Dependency   $link,
                         IN/OUT CIM_Dependency[] $links[],
                         OUT int #error_code) {
    boolean #wasFound = false;
    #new_added = false;

    // Search through the nodes looking for a match.
    // Not a particularly efficient way of doing it, but functional.
    for #i in $links[] {
        if (&RepresentsTheSameAssociation($link, $links[#i], #error_code)) {
            if (#error_code == 0) {
               #wasFound = true;
            }
            break;
        }
    }

    // If we did not find a match, and there were no errors, add it.
    if ( (!#wasFound) && (#error_code == 0)) {
        $links[].add($link);
        #new_added = true;
    }
} // AddIfNotAlreadyAdded.

sub RepresentsTheSameAssociation(IN     CIM_Dependency   $link1,
                                 IN     CIM_Dependency   $link2,
                                 OUT int #error_code) {

    // Determine if the links are the same by comparing thier class
    // and the correlatable identifiers of their endpoints.

    if ( ($link1.getObjectClass() == $link2.getObjectClass()
         (&RepresentsTheSameObject($link1.Antecedent, $link2.Antecedent,
#error_code) &&
         (&RepresentsTheSameObject($link1.Dependent, $link2.Dependent,
#error_code)
        ) {
        return true;
    } else {
        return false;
    }
}
sub RepresentsTheSameObject(IN     CIM_LogicalElement   $node1,
                            IN     CIM_LogicalElement   $node2,
                            OUT int #error_code) {

    int #error_code = 0;
    boolean #result;

    // First, check if this is the same instance by checking object path.
    if ($node1.getObjectPath() == $node2.getObjectPath()) {
       return true;
    }

    // These objects can be handled by Name&NameFormat.
    if ( (($node1 ISA CIM_LogicalDisk) && ($node2 ISA CIM_LogicalDisk)) ||
          ($node1 ISA CIM_StorageVolume) && ($node2 ISA CIM_StorageVolume)) ||
          ($node1 ISA CIM_StorageExtent) && ($node2 ISA CIM_StorageExtent)) ||
          ($node1 ISA CIM_SCSIProtocolEndpoint) && ($node2 ISA
CIM_SCSIProtocolAEndpoint)) ||
       ) {
       #result = &MatchUnambiguouslyByNameNameFormat($node1.Name,
$node1.NameFormat,
                                                  $node2.Name, $node2.NameFormat);

    // ComputerSystems are compared by two methods.
  } else if ($node1 ISA CIM_ComputerSystem) && ($node2 ISA CIM_ComputerSystem)) {
         #result = &MatchUnambiguouslyByNameNameFormat($node1.Name,
$node1.NameFormat,
                                                  $node2.Name, $node2.NameFormat);
       if (!#result) {
         #result =
&MatchUnambiguouslyByIdentifyingInfo($node1.OtherIdentifyingInfo[],

$node1.IdentifyingDescriptions[],
                                                   $node2.OtherIdentifyingInfo[],

$node2.IdentifyingDescriptions[]);
       }

    // These objects are compared by name.
    } else if (($node1 ISA CIM_GenericDiskPartition) && ($node2 ISA
CIM_GenericDiskPartition)) {
       #result = ($node1.Name == $node2.Name);
    } else if (($node1 ISA CIM_FCPort) && ($node2 ISA CIM_FCPort)) {
       #result = ($node1.Name == $node2.Name);

    // These DiskDrive and StoragePool have their own monikers.
    } else if (($node1 ISA CIM_DiskDrive) && ($node2 ISA CIM_DiskDrive)) {
       #result = ($node1.DeviceID == $node2.DeviceID);
    } else if (($node1 ISA CIM_StoragePool) && ($node2 ISA CIM_StoragePool)) {
       #result = ($node1.InstanceID == $node2.InstanceID);
    } else {
       < this method can’t handle this type >
       #error_code = 1;
       return false;
    }
    return #result;

} // RepresentsTheSameObject.


sub MatchUnambiguouslyByNameNameFormat(string name1,
                                        string nameFormat1,
                                        string name2,
                                        string nameFormat2
                                       ) {

    if (nameFormat1 != nameFormat2) {
        return false;
    } else {
        if (name1 == name2) {
           return true;
        }
    }
    return false;
}

sub MatchUnambiguouslyByIdentifyingInfo(string info1[],
                                         string desc1[],
                                         string info2[],
                                         string desc2[]
                                        ) {
    boolean #matchFound = false;

    // Loop through both arrays looking for a match.
    for (#i=0; #i<info1[].length; #i++) {
        for (#j=0; #j<info2[].length; #j++) {
            if (MatchUnambiguouslyByNameNameFormat(desc1[#i],
                                                   info1[#i],
                                                   desc2[#j],
                                                   info2[#j]
                                                  )
               ) {
                #matchFound = true;
                break;
        }
        if (#matchFound) {
           break;
        }
    }
    return #matchFound;
}

sub GetProviderInstanceOf(IN CIM_LogicalElement  $that_node,
                          OUT CIM_LogicalElement $this_node,
                          OUT int #error_code) {

    CIM_LogicalElement $possible_matches[];

    $this_node = null;

    // Enumerate through all the instances of this class in this provider
    // looking for a match to $that_node.

    $possible_matches = EnumInstances($that_node.getClass(), false, false);
    for #i in $possible_matches[] {
        if ( &RepresentsTheSameObject($that_node, $possible_matches[#i],
#error_code)
             && !#error_code) {
           $this_node = $possible_matches[#i];
        )
    }
}  // GetProviderInstanceOf.
```

8.3.5.3 Array paths

```
Array layer piece of the Logical Disk Composition Recipe

This is based on the
Array Profile, which uses the Target Port Subprofile.
It connects StorageVolumes left by the SCSI initiator
side to their FCPorts on the array side to allow
network and logical disk topologies to be correlated.

sub AddToGraphFromLayerArray(IN/OUT CIM_LogicalElement $nodes[],
                             IN/OUT CIM_Dependency $links[],
                             OUT    boolean #new_added,
                             OUT    int     #error_code) {
```

```
CIM_SCSIProtocolController      $found_protocol_controllers[];
CIM_ProtocolControllerForUnit   $found_for_unit_associations[];
CIM_ProtocolEndpoint            $found_protocol_endpoints[];
CIM_DeviceSAPImplementation     $found_sap_associations[];
CIM_LogicalPort                 $found_ports[];
CIM_SAPAvailableForElement      $found_available_associations[];

boolean #added = false;

#new_added = false;

for #i in $nodes[] {

   if ($nodes[#i] ISA CIM_StorageVolume) {

    &GetProviderInstanceOf($nodes[#i], $node, #error_code);
    if (#error_code) { return;}

    if ($node != null) {

      // First, work up the path to include the network port
      // for stitching in the network topology.

      // Question: Could there be multiple ports?

      // Follow an ProtocolControllerForUnit to a SCSIProtocolController.
      $found_protocol_controllers[] = Associators(
                                 $node.getObjectPath(),
                                 “CIM_SCSIProtocolControllerForUnit”,
                                 “CIM_SCSIProtocolController”,
                                 “Dependent”,
                                 “Antecedent”
                                       );

      $found_for_unit_associations[] = References($node.getObjectPath(),
                                                  “CIM_SCSIProtocolController”,
                                                   “Dependent”
                                                  );

      &AddIfNotAlreadyAdded($found_protocol_controllers[0],
                              $nodes[], #added, #error_code);
      #new_added |= #added;
      &AddLinkIfNotAlreadyAdded($found_for_unit_associations[0],
                                  $links[], #added, #error_code);
      #new_added |= #added;

      // Follow an SAPAvailableForElement to a SCSIProtocolEndpoint.
```

```
         $found_protocol_endpoints[] = Associators(
                                 $found_protocol_controllers[0],
                                 “CIM_SAPAvailableForElement”,
                                 “CIM_ProtocolEndpoint”,
                                 “ManagedElement”,
                                 “AvailableSAP”
                                       );

         $found_available_associations[] =
References($found_protocol_controllers[],
                                                     “CIM_ProtocolEndpoint”,
                                                     “ManagedElement”
                                                    );

         &AddIfNotAlreadyAdded($found_protocol_endpoints[0],
                              $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_available_associations[0],
                                  $links[], #added, #error_code);
         #new_added |= #added;

         // Follow the DeviceSAPImplementation to a LogicalPort.
         $found_ports[] =
Associators($found_protocol_endpoints[0].getObjectPath(),
                                                “CIM_DeviceSAPImplementation”,
                                                “CIM_LogicalPort”,
                                                “Dependent”,
                                                “Antecedent”
                                               );

         $found_sap_associations[] =
References($found_protocol_endpoints.getObjectPath(),
                                                     “CIM_LogicalPort”,
                                                     “Dependent”
                                                    );
         &AddIfNotAlreadyAdded($found_ports[0],
                              $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_sap_associations[0],
                                  $links[], #added, #error_code);
         #new_added |= #added;

   } // for #i.


} // AddToGraphFromLayerArray.
```

8.3.5.4 Host Discovered Resource

```
// Host Discovered Resources layer piece of the Logical Disk Composition Recipe

// It uses the Host Discovered Resources Profile.

sub AddToGraphFromLayerHostDiscoveredResources(
                                  IN/OUT CIM_LogicalElement $nodes[],
                                  IN/OUT CIM_LogicalElement $links[],
                                  OUT boolean #new_added,
                                  OUT int     #error_code) {
boolean #added = false;
#new_added = false;

for #i in $nodes[] {

   // CIM_SCSIInitiatorTargetLogicalUnitPath $scsi_paths[];
   // CIM_SCSIProtocolEndpoint $initiator_endpoint;
   // CIM_SCSIProtocolEndpoint $target_endpoint;

   #i =0;

   if ($nodes[#i] ISA CIM_LogicalDisk) {

       &GetProviderInstanceOf($nodes[#i], $node, #error_code);
       if (#error_code) { return; }

       if ($node != null) {

          // Find all CIM_SCSIInitiatorTargetLogicalUnitPath
          // with $node as the LogicalUnit reference.
          $scsi_paths[] = References(
                            $node.getObjectPath(),
                            “CIM_SCSIInitiatorLogicalUnitPath”,  //ResultClass
                            “LogicalUnit”                        // Role
                                    );

          for (#j=0; #j<$scsi_paths.length; #j++) {
              &AddLinkIfNotAlready($scsi_paths[#j], $links[],
                                    #added, #error_code);
              #new_added |= #added;
              $initiator_endpoint = $scsi_paths[#j].Initiator;
              $target_endpoint = $scsi_paths[#j].Target;
              &AddIfNotAlreadyAdded($initiator_endpoint, $nodes[],
                                     #added, #error_code);
              #new_added |= #added;
              &AddIfNotAlreadyAdded($target_endpoint, $nodes[],
                                    #added, #error_code);
              #new_added |= #added;
          }

      } // if $node != null.
   } // if $node ISA.

}  // For #i.


} // AddToGraphFromLayerHostDiscoveredResources.
```

8.3.5.5 Common Initiator Port

```
Common Initiator layer piece of the Logical Disk Composition Recipe

It uses the Common Initiator Port Subprofile.

sub AddToGraphFromCommonInitiator(IN/OUT CIM_LogicalElement $nodes[],
                                  IN/OUT CIM_LogicalElement $links[],
                                  OUT boolean #new_added,
                                  OUT int     #error_code) {
boolean #added = false;
#new_added = false;

for #i in $nodes[] {

   CIM_SCSIInitiatorTargetLogicalUnitPath $scsi_paths[];
   CIM_SCSIProtocolEndpoint $initiator_endpoint;
   CIM_SCSIProtocolEndpoint $target_endpoint;

   #i =0;

   if ($nodes[#i] ISA CIM_LogicalDisk) {

       &GetProviderInstanceOf($nodes[#i], $node, #error_code);
       if (#error_code) { return; }

       if ($node != null) {

          // Find all CIM_SCSIInitiatorTargetLogicalUnitPath
          // with $node as the LogicalUnit reference.
          $scsi_paths[] = References($node.getObjectPath(),
                                     “CIM_SCSIInitiatorLogicalUnitPath”,  //
ResultClass
                                     “LogicalUnit”                        // Role
```

```
                                    );

                for (#i=0; #i<$scsi_paths.length; #i++) {
                   &AddLinkIfNotAlready($scsi_paths[#i], $links[], #added, #error_code);
                    #new_added |= #added;
                    $initiator_endpoint = $scsi_paths[#i].Initiator;
                    $target_endpoint = $scsi_paths[#i].Target;
                    &AddIfNotAlreadyAdded($initiator_endpoint, $nodes[], #added,
#error_code);
                    #new_added |= #added;
                    &AddIfNotAlreadyAdded($target_endpoint, $nodes[], #added,
#error_code);
                    #new_added |= #added;
                }

            } // if $node != null.
         } // if $node ISA.

      }  // For over nodes.

      // Relate the protocol endpoints to ports.
      for #i in $nodes[] {

         CIM_LogicalPort $ports[];
         CIM_DeviceSAPImplementation $sap_associations[];

         if ($nodes[#i] ISA CIM_SCSIProtocolEndpoint) {

            &GetProviderInstanceOf($nodes[#i], $node, #error_code);
            if (#error_code) { return; }

            if ($node != null) {

            // Follow the DeviceSAPImplementation assocation
            // to the LogicalPort object
            $ports[] = Associators($node.getObjectPath(),
                                   “CIM_DeviceSAPImplementation”,
                                   “CIM_LogicalPort”,
                                   “Dependent”,
                                   “Antecedent”
                                   );

            $sap_associations[] = References($node.getObjectPath(),
                                             “CIM_LogicalPort”,
                                             “Dependent”
                                            );

            // Add the port objects and associations to the graph.
            for #j in $ports[] {
               if ((null != $sap_associations[#j]) &&
                   (null != $ports[#j])
                  ) {
                  &AddLinkIfNotAlreadyAdded($sap_associations[#j], $links[], #added,
      #error_code);
                  #new_added |= #added;
                  &AddIfNotAlreadyAdded($ports[#j], $nodes[], #added, #error_code);
                  #new_added |= #added;
               }
            } for #j.

            } // if $node != null.

         } // if $node ISA.

      } // AddToGraphFromLayerCommonInitiator.
```

8.3.5.6 Fabric Layer

```
Fibre Channel Fabric layer piece of the Logical Disk Composition Recipe

It uses the Fabric profile.

Sub AddToGraphFromLayerFabric($nodes, $links, #error_code){

// This function does the following
//
// 1. Identifies all the Switches and adds their objects paths and the object
// paths of the FC Ports belonging to these Switches to the $nodes array
//
// 2. Creates a suitable Association instance (e.g. a SystemDevice Association
// instance between a Switch and a FC Port), setting its GroupComponent and
// PartComponent. Adds the object path of the Association to the $links array
//
// 3. Creates a map of all connected FC Ports (i.e., belonging to Switches
// that are ISL'd together and to Host HBAs and Storage System Front End
// Controllers)
//
// In this map, the FC Ports (i.e., the ones that are connected) are
// cross-connected.
//
// e.g., For a pair of FC Ports, one belonging to a Switch and the other
// belonging to a Host (HBA), the map indexed by the Switch Port WWN returns
// the Host (HBA) FC Port object path and the map indexed by the Host (HBA) FC //
Port WWN returns the Switch FC Port object path.
```

```
//
// The Object stored in this Map is a composite of five objects and four
// associations. They are Switch, Switch FC Port, Switch end Protocol End Point,
// Attached Protocol End Point and Attached FC Port. The Associations are
// System Device, Device SAPImplementation, ActiveConnection, The attached side
// DeviceSAPImplementation.
// This information is kept in the Map. While traversing the Host-HBA part of
// the topology, the HBA FC Ports are matched in this Map to find out if there
// is a corresponding Switch side FC Port. If yes, only then all the objects
// are that lie on that path are saved in the Nodes Array and the corresponding
// Associations that lie on the path are stored in the Links Array.
//
// Similar relationship exists between the pairs of FC Ports where one belongs //
to a Switch and the other belonging belongs to a Storage System Front End
// Controller and for FC Ports each of which belongs to a Switch.
//
// 4. Identifies all the Hosts and adds their objects paths to the $nodes array.
// Note that the object paths of the FC Ports (HBA Ports) belonging to these
// Hosts are already added to the $nodes array in step-3.
//
// 5. Creates a suitable Association instance (e.g. a SystemDevice Association
// instance between a Host and a FC Port), setting its GroupComponent and
// PartComponent. Adds the object path of the Association to the $links array.
//
// 6. Identifies all the Storage Systems and adds their objects paths to the
// $nodes array.
// Note that the object paths of the FC Ports (i.e., Front End Controller FC
// Ports) belonging to these Storage Systems are already added to the $nodes
// array in step-3.
//
// 7. Creates a suitable Association instance (e.g. a SystemDevice Association
// instance between a Storage System and a FC Port), setting its GroupComponent //
and PartComponent. Adds the object path of the Association to the $links
// array.
//
// First find all the switches in a SAN. Get all the FC Ports for each
// switch and get the Attached FC Ports for each Switch FC Port. Save these
// device FC ports in the map described above.

// PREEXISTING CONDITIONS AND ASSUMPTIONS
// 1. All agents/namespaces supporting Fabric Profile previously identified
// using SLP. Do this for each CIMOM supporting Fabric Profile

// A composite elementsOnPath object is created. This object will be populated
// as we go along and will be stored in elementsOnPathMap with the index
// of attached FC Port WWN

ElementsOnPath #elementsOnPath = new ElementsOnPath();
```

```
ElementsOnPathMap #elementsOnPathMap = new ElementsOnPathMap();

switches[] = enumerateInstances(“CIM_ComputerSystem”, true, false, true,
true, null)
for #i in $switches[]
{
if (!contains(5, $switches[#i].Dedicated))
continue // only process switches, not other computer systems

// Add the switch to the elementsOnPath object

#elementsOnPath.switch = $switches[#i];

// Get all the SystemDevice associations between this switch and its FC Ports

$sysDevAssoc[] = ReferenceNames($switches[#i],
                                “CIM_FCPort”,
                                “GroupComponent”);

// Add the system device associations to the links array

for #a in $sysDevAssoc-[]
$links.addIfNotAlreadyAdded ($sysDevAssoc[#a];

$fcPorts->[] = AssociatorNames(
$switches[#i].getObjectPath(),
“CIM_SystemDevice”,
“CIM_FCPort”,
“GroupComponent”,
“PartComponent”)
for #j in $fcPorts->[]
{
// Add the FC Port to the elementsOnPathObject

#elementsOnPath.swFCPort = fcPorts->[#j];

$protocolEndpoints->[] = AssociatorNames(
fcPorts->[#j],
“CIM_DeviceSAPImplementation”,
“CIM_ProtocolEndpoint”,
“Antecedent”,
“Dependent”);

// NOTE - It is possible for this collection to be empty (i.e., ports that are not
// connected). It is NOT possible for this collection to have more than one
// element
```

```
if ($protocolEndpoints->[].length == 0)
continue

// Add the Protocol End Point to the elementsOnPathObject

#elementsOnPath.prorEP = protocolEndpoints[0];

// Add the associations between the fcPort and the Protocol end point to the
// links array

$devSAPImplassoc[]  = ReferenceNames($fcPorts->[#j],
                                     “CIM_ProtocolEndpoint”,
                                     “Antecedent”);
for #a in $devSAPImplassoc->[]
$links.addIfNotAlreadyAdded ($devSAPImplassoc->[#a];

$attachedProtocolEndpoints->[] = AssociatorNames(
$protocolEndpoints->[0],
“CIM_ActiveConnection”,
“CIM_ProtocolEndpoint”,
 null, null)

//Add the AttachedProtocolEndPoint to the elementsOnPath object

elementsOnPath.attachedPEP = attachedProtocolEndpoints->[0];

// Get the associations between the Protocol end point and the Attached
// protocol endpoint

$actConnassoc[]  = ReferenceNames($protocolEndpoint->[#0],
                                  “CIM_ActiveConnection”,
                                   “Antecedent”);

// Add it to the elementsOnPath object
elementsOnPath.actConn = actConnAssoc->[0];


// NOTE: role & resultRole are null as the direction of the association is not
// dictated by the specification

// $attachedFcPort is either a device FC port or an ISL’d switch FC port from
// another switch. We store this result is stored (i.e. which device
// FC Port is connected // to which switch FC Port) in a suitable data
// structure for subsequent correlation to ports discovered on devices.

for #k in $attachedProtocolEndpoints->[] {
$attachedFcPorts->[] = Associators(
$attachedProtocolEndpoints->[#k],
“CIM_DeviceSAPImplementation”,
“CIM_FCPort”,
“Dependent”,
“Antecedent”,
false,
false,
[“PermanentAddress”]);

$attachedFcPort = $attachedFcPorts[0] // Exactly one member guaranteed by model

// Add the attached FC Port to the elementsOnPath object

if $attachedFcPort != null
  #elementsOnPath.attFCPort = $attachedFcPort);

// Save the elementsOnPath object in elementsOnPath Map with the index of
// wwn of the attached fc port

elementsOnPathMap.put ($attachedfcPort.PermanentAddress, elementsOnPath);
}
}
}

 //   HBA to switch paths
// DESCRIPTION
// Determine physical path from HBA to switch.
//
// For each HBA FC port on every host, determine the connected switch
//FC port. NOTE: Not every HBA FC port will be connected to a switch FC port,
// and not every switch FC port will be connected to a device FC port. Only
// the connections between HBA FC ports and switch FC ports are discovered
// by this recipe
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1. All agents/namespaces supporting HBA Profile previously identified
// using SLP
// 2. Array $attachedFcPorts->[] is a map of how elements in a SAN are
// connected together via Fibre-ChannelFC ports. Each index is a WWN of
// any device port on the SAN, and the value at that index is the
// connected switch FC port.

// Do this for each CIMOM supporting HBA Profile

$hosts[] = enumerateInstances(“CIM_ComputerSystem”)
for #i in $hosts->[]
{
```

```
if (!contains(0, $hosts[#i].Dedicated))
continue // only process systems that are “not dedicated”
$fcPorts[] = Associators(
$hosts[#i].getObjectPath(),
“CIM_SystemDevice”,
“CIM_FCPort”,
“GroupComponent”,
“PartComponent”,
false,
false,
[“PermanentAddress”])

// If the Host has FC Ports, add the Host to the $nodes array

if $fcPorts[] != null
$nodes.addIfNotAlreadyAdded ($hosts[#i]);

// Get all the SystemDevice associations between this host and its FC Ports

$sysDevAssoc[] = ReferenceNames($hosts[#i],
                                “CIM_FCPort”,
                                “GroupComponent”);

// Add these associations to the $links array

for #a in $sysDevAssoc-[]
$links.addIfNotAlreadyAdded ($sysDevAssoc[#a];

for #j in $fcPorts[]
{
// Get the FCPort WWN

#wwn = $fcPorts[#j].PermanentAddress

// Match this device port WWN to one (or less) switch FC ports, by using the
// mapping table built above

$elementsOnPath = elementsOnPathMap.get(#wwn);

// If a match is found, then add all the elements from the elementsOnPath
// object to nodes and links array.

// This will ensure that only those Switches and Switch FC Ports etc that are on a
path will be entered in the nodes and links array

if elementsOnPath != null
{
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getSwitch());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getswFCPort());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getPEP());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.geAttPEP());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getAttFCPort());

   $links.addIfNotAlreadyAdded (elementsOnPath.getDevSAPImpl());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getActConn());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getAttDevSAPImpl());
}

}
//    Determine physical path from Switch to Storage Arrays
// DESCRIPTION
// Determine physical path from Storage Arrays to Switches
//
// For each fibre-channelFC port on every array, determine the connected
// switch FC port. NOTE: This identifies the FrontEnd I/O Controllers
// (and Storage Arrays) whose FC ports are physically connected to
// some of the FC ports of some of the Switches. This recipe does not
// distinguish and does not filter the front-end FC Port from the
// back-end FC Ports.
//
// PRE-EXISTING CONDITIONS AND ASSUMPTION
// 1. All agents/namespaces conforming to the Array profile previously
// identified
// 2. Array $attachedFcPorts[] is a map of how elements in a SAN are
// connected together via Fibre-ChannelFC ports. Each index is a WWN of
// any device FC port on the SAN, and the value at that index is the
// connected switch FC port.

// Do this for each CIMOM supporting the Array Profile:
// First identify upper-level computer systems for storage arrays -
// see the Server Profile clause  for how to use the Server profile to do this,
// or (as here) enumerate all systems within a conforming namespace
$computerSystems[] = enumerateInstances(“CIM_ComputerSystem”);
#n = 0
for #i in $computerSystems[]
{
if (!contains(3, $computerSystems[#i].Dedicated))
continue // only process systems that are dedicated “storage”
if (!contains(15, $computerSystems[#i].Dedicated))
continue // only process systems that are dedicated “block server”
$storageSystems[#n++] = $computerSystems[#i]
}
// Now accumulate all subsidiary computerSystems (cluster members or
// storage controllers) - treat $storageSystems[] as a queue and stuff
// newly discovered subsidiaries onto the end, so that ComponentCS
// associations are followed to arbitrary depth
#i = 0
while (#i < #n)
{
$subsidiaries[] = Associators(
$storageSystems[#i].getObjectPath(),
“CIM_ComponentCS”,
“CIM_ComputerSystem”,
“GroupComponent”,
“PartComponent”,
false,
false,
null)
for #j in $subsidiaries[]
{
$storageSystems[#n++] = $subsidiaries[#j]
}
#i++;
}
// Now get scoped FC ports for all the systems that have been accumulated
// NOTE: Some of the FC ports contained will be back-end ports, but they will
// have no connectivity to switches, so we won’t distinguish them
// from unconnected front-end FC ports

for #i in $storageSystems[]
{
$fcPorts[] = Associators(
$storageSystems[#i].getObjectPath(),
“CIM_SystemDevice”,
“CIM_FCPort”,
“GroupComponent”,
“PartComponent”,
false,
false,
[“PermanentAddress”])
for #j in $fcPorts[]
{
// Get the FCPort WWN
#wwn = $fcPorts[#j].PermanentAddress

// If the Storage System has FC Ports, add the storage system to the $nodes array

if $fcPorts[] != null
$nodes.addIfNotAlreadyAdded ($storageSystems[#i]);

// Get all the SystemDevice associations between this host and its FC Ports
$sysDevAssoc[] = ReferenceNames($storageSystems[#i],
                                "CIM_FCPort”,
                                “GroupComponent”);

// Add these associations to the $links array

for #a in $sysDevAssoc-[]
$links.addIfNotAlreadyAdded ($sysDevAssoc[#a];

for #j in $fcPorts[]
{
// Get the FCPort WWN

#wwn = $fcPorts[#j].PermanentAddress

// Match this device port WWN to one (or less) switch FC ports, by using the
// mapping table built above

$elementsOnPath = elementsOnPathMap.get(#wwn);

// If a match is found, then add all the elements from the elementsOnPath
// object to nodes and links array.

// This will ensure that only those Switches and Switch FC Ports etc that are on a
path will be entered in the nodes and links array

if elementsOnPath != null
{
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getSwitch());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getswFCPort());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getPEP());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.geAttPEP());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getAttFCPort());

   $links.addIfNotAlreadyAdded (elementsOnPath.getDevSAPImpl());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getActConn());
   $nodes.addIfNotAlreadyAdded (elementsOnPath.getAttDevSAPImpl());
}

}
}
}
```

8.3.5.7 IP Network Layer

```
IP Network piece of the Logical Disk Composition Recipe

It uses the iSCSITarget profile(s).

sub AddToGraphFromLayerIPNetwork(IN/OUT CIM_LogicalElement $nodes[],
                                 IN/OUT CIM_Dependency $links[],
                                 OUT boolean #new_added,
                                 OUT int     #error_code) {

CIM_EnpointOfNetworkPipe    $found_endpoints_of_pipe[];
CIM_iSCSISession            $found_sessions[];
CIM_EthernetPort            $found_ports[];
CIM_DeviceSAPImplementation $found_sap_associations[];

boolean #added;

for #i in $nodes[] {

    if ($nodes[#i] instanceof iSCSIProtocolEndpoint) {

       // Find the iSCSIProtocolEndpoints left for us by the iSCSI
       // Initiator Port subprofile. These are correlated by Name-NameFormat.
       &GetProviderInstanceOf($nodes[#i], $node, #error_code);

       if ($node != null) {

       // Using the EndpointOfNetworkPipe, follow the association
       // to an iSCSISession.  This represents the topology contribution
       // if the IP Network.

         $found_sessions[] = Associators(
                                  $node.getObjectPath(),
                                  “CIM_EndpointOfNetworkPipe”,
                                  “CIM_iSCSISession”,
                                  “Antecedent”,
                                  “Dependent”
                                        );

         $found_endpoints_of_pipe[] = References($node.getObjectPath(),
                                                    “CIM_iSCSISession”,
                                                    “Antecedent”
                                                   );

         &AddIfNotAlreadyAdded($found_protocol_endpoints[0],
                               $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_available_associations[0],
                                  $links[], #added, #error_code);
         #new_added |= #added;

         // Also follow the DeviceSAPImplementation association
         // from the protocol endpoint to the EthernetPort for completeness.

         $found_ports[] = 
Associators($found_protocol_endpoints[0].getObjectPath(),
                             “CIM_DeviceSAPImplementation”,
                             “CIM_EthernetPort”,
                             “Dependent”,
                             “Antecedent”
                             );

         $found_sap_associations[] = 
References($found_protocol_endpoints[0].getObjectPath(),
                                        “CIM_EthernetPort”,
                                        “Dependent”
                                       );

         // Add the ports and sap associations.
         &AddIfNotAlreadyAdded($found_ports[0],
                              $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_sap_associations[0],
                                  $links[], #added, #error_code);
         #new_added |= #added;

       } // if $node != null.

    } // if $nodes[#i] instanceof.

} // for #i.

} // AddToGraphFromLayerIPNetwork.
```

8.3.5.8 Local Disk Layer

```
Local Disk layer piece of the Logical Disk Composition Recipe

It uses the Disk Subprofile.


// Given a CIM_StorageExtent, recursively traverse the CIM_BasedOn
// associations finding other CIM_StorageExtents on which this extent is based
// and adding the extents and associations to the found_extents
// and found_basedon_associations as you go.

sub RecursivelyAddStorageExtents( IN CIM_StorageExtent $found_extent,
                                  IN/OUT CIM_StoragePool $found_extents[],
                                  IN/OUT CIM_AllocatedFromStoragePool
$found_basedon_associations[],
                                  OUT boolean #new_added
                                );


sub AddToGraphFromLayerLocalDiskDrive(IN/OUT CIM_LogicalElement $nodes[],
                                      IN/OUT CIM_Dependency     $links[],
                                      OUT boolean               #new_added,
                                      OUT int                   #error_code) {

// Make sure we've recursively tracked down all the StorageExtents.

boolean           #added = false;
CIM_StorageExtent $found_extents[];
CIM_BasedOn       $found_associations[];

#new_added = false;

for #i in $nodes[] {

   if ($nodes[#i] ISA CIM_StorageExtent) {

      &GetProviderInstanceOf($nodes[#i], $node, #error_code);
      if (#error_code) { return; }

      if ($node != null) {
         &RecursivelyAddStorageExtents($node,
                                       $found_extents[],
                                       $found_associations[],
                                       #added);
         #new_added |= #added;
      }

      } // if $node != null.
   } // if $node ISA.

}  // For over nodes.

// Add the newly found extents to the master nodes and links.
for (#i=0; #i<$found_extents.length; #i++) {
  &AddIfNotAlreadyAdded($found_extents[#i], $nodes[],
                        #added, #error_code);
  #new_added |= #added;
  &AddLinkIfNotAlreadyAdded($found_associations[#i], $links[],
                            #added, #error_code);
  #new_added |= #added;
}


// Now see if there are any local disk drives making
// up those extents through the MediaPresent association.

CIM_DiskDrive $disk_media[];
CIM_MediaPresent $mediapresent_associations[];

for #i in $nodes[] {

   if ($nodes[#i] ISA CIM_StorageExtent) {

      &GetProviderInstanceOf($nodes[#i], $node, #error_code);
      if (#error_code) { return;}

      if ($node != null) {

         $disk_media[] = Associators($node.getObjectPath(),
                                     “CIM_MediaPresent”,
                                     “CIM_DiskDrive”,
                                     “Dependent”,
                                     “Antecedent”
                                    );

         $mediapresent_associations[] = References($node.getObjectPath(),
                                                   “CIM_DiskDrive”,
                                                   “Dependent”
                                                  );
      }

      // There should be only one asociation found for each extent.
      if (0 != $disk_media.length) {
         &AddIfNotAlreadyAdded($disk_media[0], $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($mediapresent_associations[0], $links[],
                                   #added, #error_code);
         #new_added |= #added;
      }

      } if $node != null.
   } if $node ISA .
} // for.


} // AddToGraphFromLayerLocalDiskDrive.



sub RecursivelyAddExtents( IN CIM_StorageExtent $found_extent,
                           IN/OUT CIM_StorageExtent $found_extents[],
                           IN/OUT CIM_BasedOn $found_basedon_associations[],
                           OUT boolean #new_added
                          ) {

      CIM_StorageExtent $new_found_extents[];
      CIM_BasedOn $new_found_associations;

      #new_added = false;

      $new_found_extents[] = Associators($found_extent.getObjectPath(),
                                          “CIM_BasedOn”,
                                          “CIM_StorageExtent”,
                                          “Dependent”,
                                          “Antecedent”
                                         );

      $new_found_associations[] = References($node.getObjectPath(),
                                              “CIM_StorageExtent”,
                                              “Dependent”
                                             );
      for #i in $new_found_associations[] {
          $found_basedon_associations[].add($new_found_associations[#i]);
          #new_added = true;
      }
      for #i in $new_found_extents[] {
          $found_extents[].add($new_found_extents[#i]);
          &RecursivelyAddExtents($new_found_extents[#i], $found_extents[],
$found_basedon_associations[]);
          #new_added = true;
      }

}  // RecursivelyAddExtents.
```

8.3.5.9 Logical Disk Layers

```
Logical Disk Partitioning piece of the Logical Disk Composition Recipe

It uses the Disk Partition Subprofile.


// Given a CIM_GenericDiskPartition, recursively traverse the CIM_BasedOn
// associations finding other CIM_GenericDiskPartitions on which this partition is
based
// and adding the partitions and associations to the found_partitions
// and found_partition_associations as you go.

sub RecursivelyAddPartitions( IN CIM_GenericDiskPartition $found_partition,
                            IN/OUT CIM_GenericDiskPartition[] $found_partitions[],
                             IN/OUT CIM_BasedOn[] $found_partition_associations[],
                              OUT boolean #new_added
                            );

sub AddToGraphFromLayerDiskPartitioning(IN/OUT CIM_LogicalElement $nodes[],
                                        IN/OUT CIM_LogicalElement $links[],
                                        #new_added,
                                        #error_code) {

CIM_GenericDiskPartition        $found_partitions[];
CIM_LogicalDiskBasedOnPartition $found_partition_associations[];
CIM_StorageExtent               $found_extents[];
CIM_BasedOn                     $found_extent_associations[];

boolean $added = false;

#new_added = false;

for #j in $nodes[] {

    // In the Disk Partitioning Profile
    // start with a LogicalDisk DiskPartition object, as it is defined
    // as that on which storage applications (volume managers or
    // filesystems) may be placed.
    // The LogicalDisk object has DeviceID and Name attributes
    // that should be set to OS device names like
    // (/dev/sda1 on Linux or \\.\PHYSICALDRIVEX  on windows)

    if ($nodes[#j] ISA CIM_LogicalDisk) {

    &GetProviderInstanceOf($nodes[#j], $node, #error_code);
    if (#error_code) { return; }
if ($node != null) {
   // One would then follow the LogicalDiskBasedOn Partition
   // association to a GenericDiskPartition object.


  $found_partitions[] = Associators($node.getObjectPath(),
                                    “CIM_LogicalDiskBasedOnPartition”,
                                    “CIM_GenericDiskPartition”,
                                    “Dependent”,
                                    “Antecedent”
                                   );

  $found_partition_associations[] = References($node.getObjectPath(),
                                            “CIM_GenericDiskPartition”,
                                            “Dependent”
                                           );

  // To found paritions, add all recursive BasedOn associations to
  // and their partitions.
  for (#i=0; #i<$found_partitions[].length; #i++) {
     &RecursivelyAddPartitions($found_partitions[#i],
                              $found_partitions[],
                              $found_partition_associations[]);
  }

 // Now add all parititons and associations found so far.
 for (#i=0; #i<$found_partitions[].length; #i++) {
     &AddIfNotAlreadyAdded($found_partitions[#i], $nodes[],
                          #added, #error_code);
     #new_added |= #added;
     &AddLinkIfNotAlreadyAdded($found_partition_associations[#i],
                              $links[], #added, #error_code);
     #new_added |= #added;
 }

// Now follow the BasedOn associations from partitions
// to extents.

for #k in $found_partitions[] {

   // look for a BasedOn association that
   // leads to a StorageExtent.

   $found_extents[] = Associators($found_partitions[#k].getObjectPath(),
                                  “CIM_BasedOn”,
                                  “CIM_StorageExtent”,
                                        “Dependent”,
                                        “Antecedent”
                                       );

            $found_extent_associations[] = References($node.getObjectPath(),
                                                    “CIM_StorageExtent”,
                                                    “Dependent”
                                                   );

            if (  ($found_extents[0] != null) &&
                  ($found_extent_associations[0] != null) &&
               ) {
                &AddLinkIfNotAlreadyAdded($found_extent_associations[0], $links[],
                                          #added, #error_code);
                #new_added |= #added;
                &AddIfNotAlreadyAdded($found_extents[0], $nodes[],
                                      #added, #error_code);
                #new_added |= #added;
            }

         }  // For over partitions.

         // The DeviceID field of those StorageExents that are
         // StorageVolumes should be correlatable
         // to a StorageVolume object maintained by the Array profile.
         // (see Host Discovered Resources profile).


         } // if $null != $node.

         } // if $node ISA.

     }  // For over nodes.

     } // AddToGraphFromLayerDiskPartitioning.



     sub RecursivelyAddPartitions( IN CIM_GenericDiskPartition $found_partition,
                                  IN/OUT CIM_GenericDiskPartition[] $found_partitions[],
                                   IN/OUT CIM_BasedOn[] $found_partition_associations[],
                                    OUT #new_added
                                  ){

           CIM_GenericDiskPartition $new_found_partitions[];
           CIM_BasedOn $new_found_associations;
```

```
        $new_found_partitions[] = Associators($found_partition.getObjectPath(),
                                          “CIM_BasedOn”,
                                          “CIM_GenericDiskPartition”,
                                          “Dependent”,
                                          “Antecedent”
                                         );

        $new_found_associations[] = References($node.getObjectPath(),
                                              “CIM_GenericDiskPartition”,
                                              “Dependent”
                                             );

        for #i in $new_found_associations[] {
            $found_basedon_associations[].add($new_found_associations[#i]);
            #new_added = true;
        }

        for #i in $new_found_partitions[] {
            $found_partitions[].add($new_found_partitions[#i]);
            &RecursivelyAddPartitions($new_found_partitions[#i],
                                      $found_partitions[],
                                      $found_partition_associations[]);
            #new_added = true;
        }


  }  // RecursivelyAddPartitions.
```

8.3.5.10 Multipath Layer

```
Multipath layer piece of the Logical Disk Composition Recipe

It uses the SCSI Multipath Management Subprofile

sub AddToGraphFromLayerMultipath(IN/OUT CIM_LogicalElement $nodes[],
                                 IN/OUT CIM_Dependency $links[],
                                 OUT boolean #new_added,
                                 OUT int    #error_code) {
boolean #added = false;
#new_added = false;

for #j in $nodes[] {

   CIM_SCSIInitiatorTargetLogicalUnitPath $scsi_paths[];
   CIM_SCSIProtocolEndpoint $initiator_endpoint;
   CIM_SCSIProtocolEndpoint $target_endpoint;

   #i =0;
   if ($nodes[#j] ISA CIM_LogicalDisk) {

          &GetProviderInstanceOf($nodes[#j], $node, #error_code);
          if (#error_code) { return; }

          if ($node != null) {

          // Find all CIM_SCSIInitiatorTargetLogicalUnitPath
          // with $node as the LogicalUnit reference.
          $scsi_paths[] = References($node.getObjectPath(),
                                     “CIM_SCSIProtocolEndpoint”,  // ResultClass
                                     “LogicalUnit”                // Role
                                    );

          for (#i=0; #i<$scsi_paths.length; #i++) {
              &AddLinkIfNotAlreadyAdded($scsi_paths[#i], $links[], #added,
#error_code);
              #new_added |= #added;
              $initiator_endpoint = $scsi_paths[#i].Initiator;
              $target_endpoint = $scsi_paths[#i].Target;
              &AddIfNotAlreadyAdded($initiator_endpoint, $nodes[], #added,
#error_code);
              #new_added |= #added;
              &AddIfNotAlreadyAdded($target_endpoint, $nodes[], #added,
#error_code);
              #new_added |= #added;
          }    // for.

          } // if $node != null.

   } // if $node ISA.

}  // For over nodes.

} // AddToGraphFromLayerMultipath.
```

8.3.5.11 Virtualizer Layer

```
Multipath layer piece of the Logical Disk Composition Recipe

It is based on the Storage Virtualizer Profile,
 which includes the Target Port Subprofile,
the Block Services Package, and the
Initiator Port Subprofile.  It stitches StorageVolumes
it finds up to their ingress ports, across the layers
of virtualization, and out their egress ports.

// Given a CIM_StoragePool, recursively traverse the CIM_AllocatedFromStoragePool
// associations finding other CIM_StoragePools on which this pool is based
// and adding the pools and associations to the found_pools
// and found_allocated_associations as you go.  This method is implemented
// in with the LayerVolumeManager subroutine of this recipe.
sub RecursivelyAddPools( IN CIM_StoragePool$found_pool,
                         IN/OUT CIM_StoragePool $found_pools[],
                         IN/OUT CIM_AllocatedFromStoragePools
$found_allocated_associations[],
                         OUT boolean #new_added
                       );

sub AddToGraphFromLayerStorageVirtualizer(IN/OUT CIM_LogicalElement $nodes[],
                                          IN/OUT CIM_Dependency $links[],
                                          OUT    boolean #new_added,
                                          int    #error_code) {

   CIM_SCSIProtocolController      $found_protocol_controllers[];
   CIM_ProtocolControllerForUnit   $found_for_unit_associations[];
   CIM_ProtocolEndpoint            $found_protocol_endpoints[];
   CIM_DeviceSAPImplementation     $found_sap_associations[];
   CIM_LogicalPort                 $found_ports[];
   CIM_SAPAvailableForElement      $found_available_associations[];
   CIM_StorageVolume               $found_storage_volumes[];
   CIM_StoragePool                 $found_storage_pools[];
   CIM_AllocatedFromStoragePool    $found_allocated_associations[];
   CIM_StorageExtent               $found_component_disks[];
   CIM_ConcreteComponent           $found_component_associations[];

   boolean #added = false;

   #new_added = false;

   for #i in $nodes[] {

      if ($nodes[#i] ISA CIM_StorageVolume) {

       &GetProviderInstanceOf($nodes[#i], $node, #error_code);
       if (#error_code) { return;}

       if ($node != null) {

         // First, work up the path to include the network port
         // for stitching in the network topology.

         // Question: Could there be multiple ports?

         // Follow an ProtocolControllerForUnit to a SCSIProtocolController.
         $found_protocol_controllers[] = Associators(
                                  $node.getObjectPath(),
                                  “CIM_SCSIProtocolControllerForUnit”,
                                  “CIM_SCSIProtocolController”,
                                  “Dependent”,
                                  “Antecedent”
                                        );

         $found_for_unit_associations[] = References($node.getObjectPath(),
                                                     “CIM_SCSIProtocolController”,
                                                      “Dependent”
                                                     );

         &AddIfNotAlreadyAdded($found_protocol_controllers[0],
                               $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_for_unit_associations[0],
                                   $links[], #added, #error_code);
         #new_added |= #added;

         // Follow an SAPAvailableForElement to a SCSIProtocolEndpoint.
         $found_protocol_endpoints[] = Associators(
                                  $found_protocol_controllers[0],
                                  “CIM_SAPAvailableForElement”,
                                  “CIM_ProtocolEndpoint”,
                                  “ManagedElement”,
                                  “AvailableSAP”
                                        );

         $found_available_associations[] =
References($found_protocol_controllers[],
                                                      “CIM_ProtocolEndpoint”,
                                                      “ManagedElement”
                                                     );

         &AddIfNotAlreadyAdded($found_protocol_endpoints[0],
                               $nodes[], #added, #error_code);
         #new_added |= #added;
         &AddLinkIfNotAlreadyAdded($found_available_associations[0],
                                   $links[], #added, #error_code);
         #new_added |= #added;

         // Follow the DeviceSAPImplementation to a LogicalPort.
         $found_ports[] =
Associators($found_protocol_endpoints[0].getObjectPath(),
                                                 “CIM_DeviceSAPImplementation”,
                                                 “CIM_LogicalPort”,
                                                 “Dependent”,
                                                 “Antecedent”
```

```
                                              );

          $found_sap_associations[] =
References($found_protocol_endpoints.getObjectPath(),
                                                  “CIM_LogicalPort”,
                                                  “Dependent”
                                                 );
          &AddIfNotAlreadyAdded($found_ports[0],
                                $nodes[], #added, #error_code);
          #new_added |= #added;
          &AddLinkIfNotAlreadyAdded($found_sap_associations[0],
                                    $links[], #added, #error_code);
          #new_added |= #added;

          // Now, work down the path through the virtualization layer
          // and out the other side to the network ports.

          // Follow the AllocatedFromStoragePool to a StoragePool.
          $found_storage_pools[] = Associators(
                                   $node.getObjectPath(),
                                   “CIM_AllocatedFromStoragePool”,
                                   “CIM_StoragePool”,
                                   “Dependent”,
                                   “Antecedent”
                                         );

          $found_allocated_associations[] = References($node.getObjectPath(),
                                                  “CIM_StoragePool”,
                                                  “Dependent”
                                                 );

          // Recursively add other StoragePools by following additional
          // AllocatedFromStoragePool associations.
          for #j in $found_storage_pools[] {
                 &RecursivelyAddPools($found_storage_pools[#j],
                                      $found_storage_pools[],
                                      $found_allocated_associations[],
                                      #added
                                     );
                 #new_added |= #added;
          } // for #j.

          for #j in $found_storage_pools[] {

              // Add the pools and allocated associations.
              &AddIfNotAlreadyAdded($found_storage_pools[#j],
                                    $nodes[], #added, #error_code);
              #new_added |= #added;
```

```
             &AddLinkIfNotAlreadyAdded($found_allocated_associations[#j],
                                          $links[], #added, #error_code);
             #new_added |= #added;

             // Follow the ConcreteComponent associations to StorageExtents.
             $found_component_disks[] =
Associators($found_storage_pools[#j].getObjectPath(),
                                                  “CIM_ConcreteComponent”,
                                                  “CIM_CIMLogicalDisk”,
                                                  “Dependent”,
                                                  “Antecedent”
                                                 );
             $found_component_associations[] =
References($found_storage_pools[#j].getObjectPath(),
                                                        “CIM_LogicalDisk”,
                                                        “Dependent”
                                                       );

             for #k in $found_component_disks {

                // Add the disks and component associations.
                &AddIfNotAlreadyAdded($found_component_disks[#k],
                                      $nodes[], #added, #error_code);
                #new_added |= #added;
                &AddLinkIfNotAlreadyAdded($found_component_associations[#k],
                                          $links[], #added, #error_code);
                #new_added |= #added;

                // Follow the SAPAvailableForElement to
                // a ProtocolEndpoint.
                $found_protocol_endpoints[] = Associators(
                                 $found_component_disks[#k].getObjectPath(),
                                 “CIM_SAPAvailableForElement”,
                                 “CIM_ProtocolEndpoint”,
                                 “ManagedElement”,
                                 “AvailableSAP”
                                       );

               $found_available_associations[] = References($node.getObjectPath(),
                                                        “CIM_ProtocolEndpoint”,
                                                        “ManagedElement”
                                                       );


                // Add the endpoints and available associations.
                &AddIfNotAlreadyAdded($found_protocol_endpoints[0],
                                      $nodes[], #added, #error_code);
                #new_added |= #added;
                &AddLinkIfNotAlreadyAdded($found_available_associations[0],
                                          $links[], #added, #error_code);
                #new_added |= #added;


                // Follow the DeviceSAPImplementation assocation
                // to the LogicalPort object.
                $found_ports[] =
Associators($found_protocol_endpoints[0].getObjectPath(),
                             "CIM_DeviceSAPImplementation",
                             "CIM_LogicalPort",
                             "Dependent",
                             "Antecedent"
                             );

                $found_sap_associations[] =
References($found_protocol_endpoints[0].getObjectPath(),
                                        "CIM_LogicalPort",
                                        "Dependent"
                                       );

                // Add the ports and sap associations.
                &AddIfNotAlreadyAdded($found_ports[0],
                                      $nodes[], #added, #error_code);
                #new_added |= #added;
                &AddLinkIfNotAlreadyAdded($found_sap_associations[0],
                                          $links[], #added, #error_code);
                #new_added |= #added;

             } // for #k.

         } // for #j.
      } // if $node ISA.
      } // if $node != null.
   } // for #i.

} // AddToGraphFromLayerStorageVirtualizer.
```

8.3.5.12 Volume Manager Layer

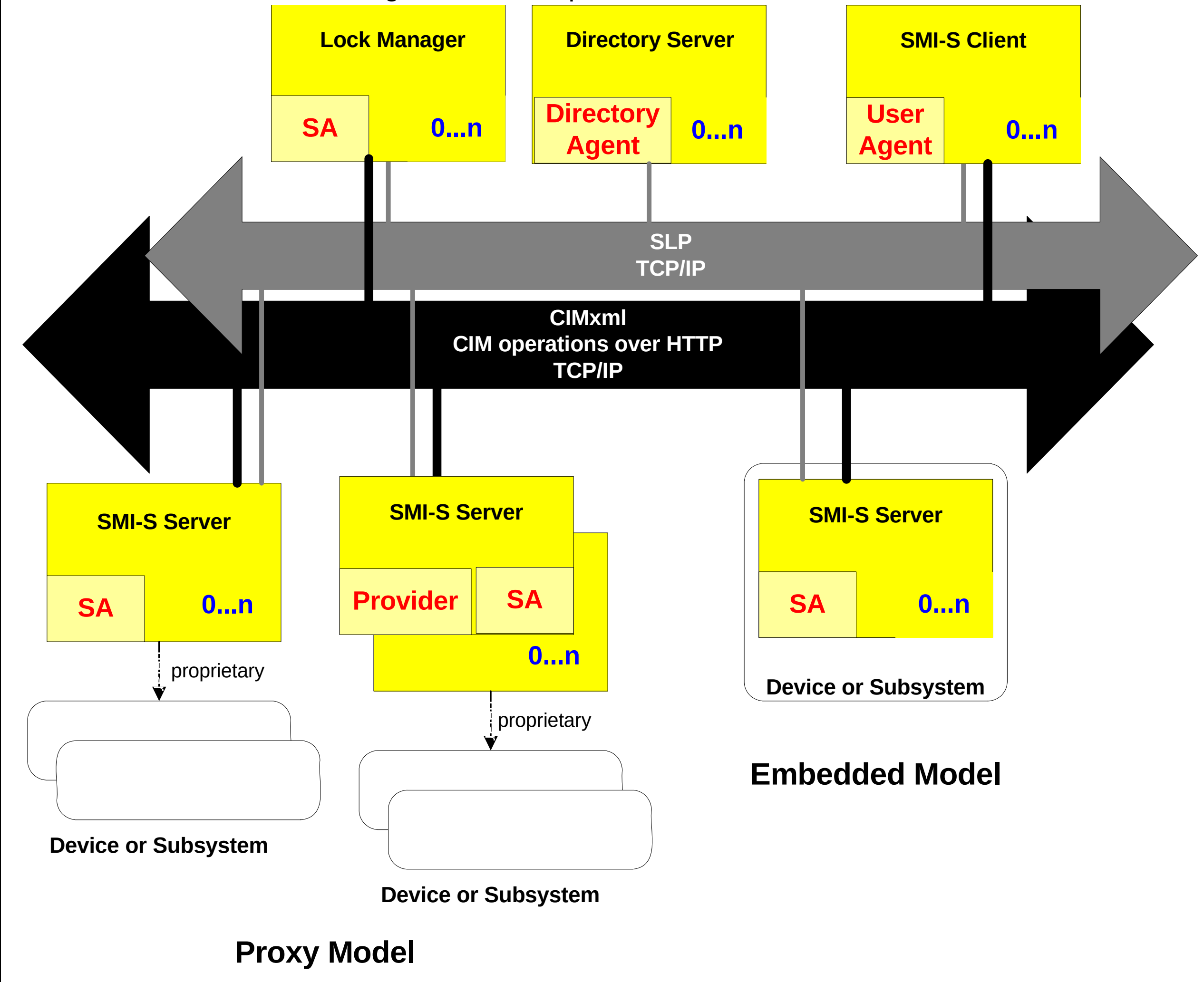
```
Volume Manager layer piece of the Logical Disk Composition Recipe

It uses the Volume Management Profile.


// Given a CIM_StoragePool, recursively traverse the CIM_AllocatedFromStoragePool
// associations finding other CIM_StoragePools on which this pool is based
// and adding the pools and associations to the found_pools
// and found_allocated_associations as you go.

sub RecursivelyAddPools( IN CIM_StoragePool$found_pool,
                         IN/OUT CIM_StoragePool $found_pools[],
                         IN/OUT CIM_AllocatedFromStoragePools
$found_allocated_associations[],
                         OUT boolean #new_added
                       );

// Given a CIM_LogicalDisk, recursively traverse the CIM_BasedOn
// associations finding other CIM_LogicalDisks on which this pool is based
// and adding the disks and associations to the found_disks
// and found_basedon_associations as you go.

sub RecursivelyAddDisks( IN CIM_LogicalDisk $found_disk,
                         IN/OUT CIM_LogicalDisk $found_disks[],
                         IN/OUT CIM_AllocatedFromStoragePools
$found_basedon_associations[],
                         OUT boolean #new_added
                       );

// We want the CIM_StoragePools to be part of the
// composition topology if they exist.

sub AddToGraphFromLayerVolumeManager(IN/OUT CIM_LogicalElement $nodes[],
                                     IN/OUT CIM_Dependency     $links[],
                                     OUT    boolean #new_added,
                                     OUT    int #error_code) {

#added = false;

for #j in $nodes[] {

   CIM_StoragePool $found_storage_pools[];
   CIM_AllocatedFromStoragePool $found_allocated_associations[];

   if ($nodes[#j] ISA CIM_LogicalDisk) {

   &GetProviderInstanceOf($nodes[#j], $node, #error_code);
   if (#error_code) { return;}

   if (($node != null)) {

      // This first method looks for cases where volume groups
      // have been created as StoragePools.

      // Follow the CIM_AllocatedFromStoragePool association
      // to a CIM_StoragePool.
      $found_storage_pools[] = Associators($node.getObjectPath(),
                                          “CIM_AllocatedFromStoragePool”,
                                          “CIM_StoragePool”,
                                          “Dependent”,
                                          “Antecedent”
                                         );

      $found_allocated_associations[] = References($node.getObjectPath(),
                                                  “CIM_StoragePool”,
                                                  “Dependent”
                                                 );

      // Then, recursively follow any CIM_AllocatedFromStoragePool
      // associations to other CIM_StoragePools, adding associations
      // and strorage pools as you go.
      for (#i=0; #i<$found_storage_pools[].length, #i++) {
          &RecursivelyAddPools( $found_storage_pools[#i],
                                $found_storage_pools[],
                                $found_allocated_associations,
                                #added
                              );
          #new_added |= #added;
      }

      for #k in $found_allocated_associations[] {
          &AddLinkIfNotAlreadyAdded($found_allocated_association[#k], $links[],
                                    #added, #error_code);
          #new_added |= #added;
      }
      for #k in $found_storage_pools[] {
          &AddIfNotAlreadyAdded($found_pool_storage_pools[#k],
                                $nodes[], #added, #error_code);
          #new_added |= #added;
      }

      // Now find the component disks of the storage pools.
      CIM_LogicalDisk[] $found_component_disks[];
      for #k in $found_storage_pools[] {
         $found_component_disks[] =
Associators($found_storage_pools[#k].getObjectPath(),
                                                 “CIM_ConcreteComponent”,
                                                 “CIM_CIMLogicalDisk”,
                                                 “Dependent”,
                                                 “Antecedent”
                                                );
         $found_component_associations[] =
References($found_storage_pools[#k].getObjectPath(),
                                                       “CIM_LogicalDisk”,
                                                       “Dependent”
                                                      );


      }

      for (#i=0; i < $found_component_disks[].length; #i++) {
            &AddLinkIfNotAlreadyAdded($found_component_associations[#i],
                                      $links[],
                                      #added,
                                      #error_code);
            #new_added |= #added;
      }

      // If this implementation does not use volume groups,
      // look for the BasedOn associations to find the disks.

      CIM_LogicalDisk[] $found_logical_disks[];
      CIM_BasedOn[] $found_basedon_associations[];

      $found_logical_disks[] = Associators($node.getObjectPath(),
                                          “CIM_BasedOn”,
                                          “CIM_LogicalDisk”,
                                          “Dependent”,
                                          “Antecedent”
                                           );

      $found_basedon_associations[] = References($node.getObjectPath(),
                                                  “CIM_LogicalDisk”,
                                                  “Dependent”
                                                 );

      // Add these disks to the component_disks.
      for (#i=0; #i<$found_basedon_associations[].length; #i++) {
         &AddLinkIfNotAlreadyAdded($found_basedon_associations[#i], $links[].
                                   #added, #error_code);
         #new_added |= #added;
         &AddIfNotAlreadyAdded($found_logical_disks[#i],
                               $found_component_disks,
                               #added, #error_code);
         #new_added |= #added;

      }

      // Follow all BasedOn associations to find more component disks
      // recursively.
      for (#i=0; #i<$found_component_disks[].length; #i++) {
```

```
         &RecursivelyAddDisks($found_component_disks[#i],
                              $nodes[],
                              $links[],
                              #added);
      }

   } // if $node != null.

   } // if $node ISA.

}  // For over nodes.

} // AddToGraphFromLayerVolumeManager.


sub RecursivelyAddPools( IN CIM_StoragePool$found_pool,
                         IN/OUT CIM_StoragePool $found_pools[],
                         IN/OUT CIM_AllocatedFromStoragePools
$found_allocated_associations[],
                         OUT boolean #new_added
                       ) {

      CIM_StoragePool $new_found_pools;
      CIM_AllocatedFromStoragePool $new_found_associations;

      #new_added = false;

      $new_found_pools[] = Associators($found_pool.getObjectPath(),
                                        “CIM_AllocatedFromStoragePool”,
                                        “CIM_StoragePool”,
                                        “Dependent”,
                                        “Antecedent”
                                       );

      $new_found_associations[] = References($node.getObjectPath(),
                                            “CIM_StoragePool”,
                                            “Dependent”
                                           );
      for #i in $new_found_associations[] {
          $found_allocated_associations[].add($new_found_associations[#i]);
          #new_added = true;
      }
      for #i in $new_found_pools[] {
          $found_pools[].add($new_found_pools[#i]);
          &RecursivelyAddPools($new_found_pools[#i], $found_pools[],
$found_allocated_associations[], #new_added);
          #new_added = true;
      }
```

```
}  // RecursivelyAddPools.


sub RecursivelyAddDisks( IN CIM_LogicalDisk $found_disk,
                         IN/OUT CIM_LogicalDisk $found_disks[],
                         IN/OUT CIM_BasedOn $found_basedon_associations[],
                         OUT boolean #new_added
                       ) {

      CIM_StoragePool $new_found_disks[];
      CIM_AllocatedFromStoragePool $new_found_associations;

      #new_added = false;

      $new_found_disks[] = Associators($found_disk.getObjectPath(),
                                        "CIM_BasedOn",
                                        "CIM_LogicalDisk",
                                        "Dependent",
                                        "Antecedent"
                                       );

      $new_found_associations[] = References($node.getObjectPath(),
                                              "CIM_LogicalDisk",
                                              "Dependent"
                                             );
      for #i in $new_found_associations[] {
          $found_basedon_associations[].add($new_found_associations[#i]);
          #new_added = true;
      }
      for $new_found_disk in $new_found_disks[] {
          $found_disks[].add($new_found_disks[#i]);
          &RecursivelyAddDisks($new_found_disks[#i], $found_disks[],
$found_basedon_associations[], #new_added);
          #new_added = true;
      }


}  // RecursivelyAddDisks.
```

Clause 9: SMI-S Roles

Figure 224: Complete Reference Model



9.1 Introduction

As shown in Figure 224: "Complete Reference Model" above, the complete reference model shows the roles for the various entities of the management system. Any given host, network device or storage device may implement one or more of these roles as described later in this clause.

Here we present a concise definition of each of these roles and the requirements on implementations of these roles in a management system. For each of these roles, specific functions are required to be implemented in one or more functional areas:

a) SLP Discovery Functions – the required discovery capabilities that the role performs in the overall management system;

b) Basic CIM-XML Operations – the management model operations that the role performs;

c) Security – the security requirements that the role is expected to satisfy;

d) Lock Management Operations – the locking operations that the role is expected to perform.

The detail of these responsibilities for each of the roles is described in the following sections.

9.2 SMI-S Client

9.2.1 Overview

The SMI-S Client role in the overall management system is performed by software that is capable of performing management operations on the resources under management. This includes monitoring, configuration, and control of the operations on the resources. Typical clients include user interface consoles, complete management frameworks, and higher-level management applications and services such as policy based management systems.

There can be zero or more SMI-S clients in the overall management system. These clients can all coexist simultaneously and can perform independent or overlapping operations in the management system. It is outside the scope of this specification to specify client cooperation with other clients in any way. The semantics of the described management system is that the last successful client operation is valid and persists in the absence of any other client operations (last write wins).

It is expected that development kits for the management system will provide code for the required functions implemented in clients. Consoles, frameworks and management applications can then use this common code in order to comply with this specification. The specification of an API for this client code, and specific language bindings for applications is outside the scope of this specification, but is a candidate for follow-on work.

9.2.2 SLP Functions

The SMI-S Client role is required to implement SLP User Agent (UA) functionality as specified in 10.6, "User Agents (UA)". The Client discovers all SMI-S servers within its configured scope that are required for its operations by querying for service specific attributes that match the criteria for those operations.

9.2.3 CIM-XML Protocol Functions

The SMI-S Client role shall implement CIM-Client functionality as specified by CIM-XML standard and should implement CIM-Listener functionality as specified by CIM-XML standard.

9.2.4 Security Considerations

The SMI-S Client role shall implement security as specified in 8.2.4.1.1.4, "HTTP Security".

SMI-S Client support for HTTP security is *REQUIRED*. This includes the following requirements applicable to clients:

- SSL 3.0 and TLS shall be supported.
- HTTP Basic Authentication shall be supported. HTTP Digest Authentication should be supported.
- HTTP Realms shall be supported.
- All certificates, including CA Root Certificates used by clients for certificate validation, shall be replaceable.
- The DER encoded X.509, Base64 encoded X.509 and PKCS#12 certificate formats shall be supported.
- Certificate Revocation Lists shall be supported in the DER encoded X.509 and Base64 encoded X.509 formats.

The above list is not comprehensive; see 8.2.4.1.1.4, "HTTP Security" for the complete requirements. If there is any conflict between this text and 8.2.4.1.1.4, the text in 8.2.4.1.1.4 is the final specification of the requirements.

9.2.5 Lock Management Functions

There are no requirements for locking in this release of the specification.

9.3 Dedicated SMI-S Server

9.3.1 Overview

The intention of the SMI-S server role in a management system is to provide device management support in the absence of any other role. A simple management system could consist of just a SMI-S Client and a SMI-S Server and all management functions can be performed on the underlying resource. This means that a vendor can offer complete management for the resource by shipping a standalone client for the resource and not depend on any other management infrastructure. Although, at the same time, the SMI-S Server can participate in a more complex management environment through the use of the standard mechanisms described here.

- Embedded SMI-S Server – the SMI-S Server functions are incorporated into the resource directly and do not involve separate installation steps to become operational.
- Proxy SMI-S Server – the SMI-S Server is hosted on a system separate from the resource and communicates with the resource via either a standard or proprietary remote protocol. This typically involves an installation operation for the SMI-S Server and configuration for, or independent discovery of, the desired resource.

In order to minimize the footprint on the resource or proxy hosts, the required functions of the SMI-S Server role have purposely been scaled back from those of a typical general purpose CIM Server running on host with more significant resources. These required functions are described in the sections below.

9.3.2 SLP Functions

The SMI-S Server role is required to implement SLP Service Agent (SA) functionality as specified in 10.7, "Service Agents (SAs)". Optionally, it should implement Service Agent Server functionality or use an existing SA Server if one exists. The SMI-S Server shall advertise service-specific attributes that allow the Client to locate it based on its profile, as defined in 10.10, "'Standard WBEM' Service Type Templates".

9.3.3 CIM-XML Protocol Functions

9.3.3.1 General

The SMI-S Server role shall implement CIM-Server functionality as specified by the CIM-XML standard.

9.3.3.2 Required Intrinsic Methods

An SMI-S Server is required to implement a set of intrinsic methods as defined for each profile. The intrinsic methods are grouped by “functional profile” as specified in the CIM-XML standard:

Table 1228: Functional Profiles

Functional Group	Dependency	Methods
Basic Read	None	GetClass EnumerateClasses EnumerateClassNames GetInstance EnumerateInstances EnumerateInstanceNames GetProperty
Basic Write	Basic Read	SetProperty

Table 1228: Functional Profiles

Functional Group	Dependency	Methods
Instance Manipulation	Basic Write	CreateInstance ModifyInstance DeleteInstance
Schema Manipulation	Instance Manipulation	CreateClass ModifyClass DeleteClass
Association Traversal	Basic Read	Associators AssociatorNames References ReferenceNames
Query Execution	Basic Read	ExecQuery
Qualifier Declaration	Schema Manipulation	GetQualifier SetQualifier DeleteQualifier EnumerateQualifiers
Indication	None	

SMI-S Servers shall implement intrinsic methods as specified in the “CIM Server Requirements” section of the Profile specification.

9.3.3.3 Required Model Support

The SMI-S Server shall implement the Server Profile as detailed in the Server Profile (8.2.4.1).

9.3.4 Security Considerations

The SMI-S Server role shall implement security as specified in 8.2.4.1.1.4, "HTTP Security".

9.3.5 Lock Management Functions

There are no requirements for locking in this release of the specification.

9.4 General Purpose SMI-S Server

9.4.1 Overview

The General Purpose SMI-S Server role in an overall management system is intended to reduce the number of network connections needed by a Client to manage large numbers of resources. It is also envisioned as a convenient place to perform operations across multiple resources, further off-loading these from the Client as well.

In addition, the General Purpose SMI-S Server role can provide a hosting environment for the plug-in instrumentation of host-based resources and management proxies for resources with remote management protocols. These plug-ins are called providers and considered sub roles of the General Purpose SMI-S Server.

A General Purpose SMI-S Server is not required in a management system, but is expected to be deployed at least as a common infrastructure for host-based resources. In any large storage network, there may be several General Purpose SMI-S Servers (as many as one per host). Communication between General Purpose SMI-S Servers may be standardized in the future, but this capability is outside the scope of this specification. General Purpose SMI-S Servers may act as a point of aggregation for multiple SMI-S Profiles as described in the Server Profile (8.2.4.1) using existing standard mechanisms as specified here.

As General Purpose SMI-S Servers are expected to be deployed on hosts with more resources and less footprint concerns than other managed resources, the required functions, specified below, are more extensive that of an Dedicated SMI-S Server.

9.4.2 SLP Functions

The General Purpose SMI-S Server role is required to implement SLP Service Agent (SA) functionality as specified in 10.7, "Service Agents (SAs)". The General Purpose SMI-S Server shall advertise service specific attributes that allow the Client to locate it based on the profiles it supports, as defined in 10.10, "'Standard WBEM' Service Type Templates".

9.4.3 CIM-XML Protocol Functions

9.4.3.1 General

The General Purpose SMI-S Server role shall implement CIM-Server functionality as specified by the CIM-XML standard.

9.4.3.2 Required Intrinsic Methods

The General Purpose SMI-S Server is required to implement the minimum profile as specified in CIM-XML standard. In addition, it shall implement the intrinsic methods needed to support the Profiles that it supports.

9.4.3.3 Required Model Support

The General Purpose SMI-S Server shall implement the Server Profile as detailed in the Server Profile section (8.2.4.1, "Server Profile").

9.4.3.4 Security Considerations

The General Purpose SMI-S Server role shall implement security as specified in 8.2.4.1.1.4, "HTTP Security".

9.4.4 Lock Management Functions

There are no requirements for locking in this release of the specification.

9.4.5 Provider Subrole

9.4.5.1 Overview

A sub-role within a General Purpose SMI-S Server that can be used to provide management support for the resource, especially useful when the resource is host-based (i.e., HBA or Host Software) and the platform provides an CIM Server as part of its operating system.

9.4.5.2 Required Model Support

The Provider shall implement the Provider Subprofile as detailed in the object model shown in the Server Profile section (8.2.4.1, "Server Profile").

9.5 Directory Server

The Directory Server role is used to facilitate Discovery of instances of the various roles in a management system, but may also be used by management systems to store common configurations, user credentials and management policies. Functions outside of Discovery are outside the scope of this specification. The Directory Server role is optional for a compliant management system.

9.5.1 SLP Functions

The Directory Server role is required to implement SLP Directory Agent (DA) functionality as specified in 10.8, "Directory Agents (DAs)". The Directory registers all Agents and Object Managers within its configured scope and allows queries for their respective service specific attributes.

9.5.2 CIM-XML Protocol Functions

There are no additional CIM-XML requirements for this role.

9.5.3 Security Considerations

There are no additional security requirements for this role.

9.5.4 Lock Management Functions

There are no requirements for locking in this release of the specification.

9.6 Combined Roles on a Single System

9.6.1 Overview

As mentioned previously, the various roles of the management system can be deployed in different combinations to different systems throughout the managed environment. In general, there are no restrictions on what roles can be deployed on any given system, but some examples are given below to illustrate typical situations.

9.6.2 General Purpose SMI-S Server as a Profile Aggregator

9.6.2.1 SLP Functions

The General Purpose SMI-S Server role may implement SLP User Agent (UA) functionality as specified in 10.6, "User Agents (UA)". The General Purpose SMI-S Server discovers all Profiles within its configured scope that are aggregated by querying for service specific attributes that match the criteria for those aggregations.

9.6.2.2 CIM-XML Protocol Functions

The General Purpose SMI-S Server role may implement CIM-Client functionality as specified by CIM-XML standard and may implement CIM-Listener functionality as specified by CIM-XML standard. A General Purpose SMI-S Server may reflect instances and classes from the aggregated Profiles (perhaps by delegating operations to the Dedicated SMI-S Servers), but is not required to do so. The Profile’s Model instances should be reflected in the advertised default namespace of the General Purpose SMI-S Server. The hierarchy of General Purpose SMI-S Servers and Dedicated SMI-S Servers in a multi-level system needs to be reflected in the model such that it can be administrated.

9.6.2.3 Security Considerations

There are no requirements for security for this role.

9.6.2.4 Lock Manager Functions

There are no requirements for locking in this release of the specification.

Clause 10: Service Discovery

10.1 Objectives

Service discovery in the context of SMI-S refers to the discovery of dedicated SMI-S servers, general purpose SMI-S servers, and directory servers, and the functions they offer in an SMI-S managed environment. The specific objectives to be addressed by the discovery architecture are:

a) Provide a mechanism that allows SMI-S clients to discover the SMI-S constituents in a storage network environment so that they may communicate with these constituents using CIM Operations over HTTP protocol. This includes:

   1) Finding the address for the SMI-S constituent;

   2) Finding the capabilities of the server, including communications capabilities, security capabilities, CIM operational capabilities and the functional capabilities (CQL, Batch operations support, etc.);

b) Provide a mechanism that is efficient in the amount of information exchanged with minimal exchanges to acquire the information;

c) Provide a mechanism that accurately defines the services in the network, independent of whether or not those services are currently available;

d) Provide a mechanism that provides information on namespaces provided and the CIM Schema supported;

e) Provide a mechanism that allows SMI-S clients the profile(s) supported by agents and object managers;

f) Provide a mechanism that scales to enterprise environments;

g) Utilize existing standard mechanisms to effect the SMI-S service discovery to enable rapid deployment;

h) Provide a mechanism that allows SMI-S clients to determine the level of (SMI-S) support provided by the constituents (e.g., R1, R2, etc.

10.2 Overview

SMI-S Release 1 uses the Service Location Protocol Version 2 (SLPv2), as defined by IETF RFC2608, for its *basic* discovery mechanism. SLPv2 is used to locate constituents (agents, object managers, etc.), but complete discovery of all the services offered involves traversing the interoperability model for the SMI-S profile supported. This clause of the SMI-S specification deals primarily with the information discovered using SLPv2. There are references to information discovered by traversing the interoperability model, but details on this are provided in 9.3, "Dedicated SMI-S Server".

Note: SLPv1 is not supported in SMI-S as discovery mechanism. SMI-S requires capabilities that were introduced in SLPv2 in order to support the discovery of SMI-S agents and object managers. SLPv2 defines discovery protocols among three constituents:

User Agent (UA): A process that attempts to establish contact with one or more services. A User Agent retrieves service information from Service Agents or Directory Agents. In SMI-S, a “user agent” would be part of an SMI-S Client.

Service Agent (SA): A process working on behalf of one or more services to advertise the services. In SMI-S, a “service agent” would be supported by SMI-S dedicated or general purpose servers.

Directory Agent (DA): A process that caches SLP service advertisements registered by Service Agents and forwards the service advertisements to User Agents on demand. In SMI-S, the SLP “Directory agent” is defined as the main function of the “directory server” role in the SMI-S Reference Model.

SLPv2 provides a framework for client applications, represented by User Agents, to find and utilize services, represented by Service Agents. The Directory Agent represents an optional part that enhances the performance and scalability of the protocol by acting as a cache for all services that have been advertised. Directory Agents also reduce the load on Service Agents, making simpler implementations of Service Agents possible. User Agents can then query the Directory Agent for services. Service Agents register with Directory Agents and are required to re-register as the registrations expire. If no Directory Agent is present, User Agents may request service information directly from the Service Agents.

Using SLPv2, a client can discover SMI-S servers and SLPv2 Directory Agents in the storage network. In the case of SMI-S servers, the basic information discovered is the profiles supported and the URL of the service. Details on the specific services provided with the profile are then found by traversing the service structure modeled for the profile.

Using SLPv2, a “service agent” advertises its services. These advertisements have an expiration time period. To avoid getting an advertisement deleted, a service agent shall reregister before the time period expires. SMI-S servers may deregister as part of a graceful shutdown.

A service advertisement consists of file components:

- Service type name – describes the general type of service being advertised (ex. Printing, faxing, etc.). The working assumption is that DMTF wants “WBEM Servers” advertised with the service type WBEM. This is used by SMI-S servers (both dedicated and general purpose servers).;
- Attributes – The collection of attributes describes the particular instance of the service in more detail. For SMI-S, these would be the attributes defined by the service type template for WBEM. The attributes are defined in 10.5.2, "Service Attributes";
- Service Access point – the service access point defines the point of connection that the software client of the UA uses to connect to the service over the network.;
- Scopes – These are administrative groupings of services. The default value (“default”) should be used for SMI-S servers. Other scopes may be defined by the customer, but care must be taken when this is done. The administrator must do this correctly or SMI-S servers will not be visible. All the SMI-S recipes assume that DEFAULT is set for scopes;
- Language – Services advertisements contain human readable strings. These are provided in English, but may also be in other languages.

SLPv2 provides for authentication of service URLs and service attributes. This provides user agents (UAs) and directory agents (DAs) with assurances of the integrity of service URLs and attributes included in SLP messages. The only systems which can generate digital signatures are those which have been configured by administrators in advance. Agents that verify signed data may assume it is trustworthy inasmuch as administrators have assured trustworthiness through the cryptographic keying of SAs and DAs. The SLPv2 security model assumes that service information is public, and therefore does not require confidentiality.

Section 2.5 of RFC 3723 - *Securing Block Storage Protocols over IP* states that the SA advertisements as well as UA requests and/or responses are vulnerable to the following security threats:

1) An attacker could insert or alter service agent (SA) advertisements or responses to a UA requests in order to masquerade as the real peer or launch a denial of service attack.

2) An attacker could gain knowledge about an SA or a UA through sniffing, and launch an attack against the peer.

3) An attacker could spoof DA advertisements and thereby cause UAs and SAs to use a rogue DA.
Section 2.5 of RFC 3723 also outlines the capabilities required to address these threats, but notes that SLP (as defined in RFC 2608) does not satisfy these security requirements. SLPv2 only provides end-to-end authentication (i.e., does not support confidentiality), but with this authentication, there is no way to authenticate zero result responses. Thus an attacker could mount a denial of service attack by sending UAs a zero results Service Reply (SrvRply) or Attribute Reply (AttrRply) with a source address corresponding to a legitimate DA advertisement.

The RFC 3723 mitigation strategies include reliance on digital signatures for authentication of service URLs and attributes as well as IPsec. For SMI-S environments that require security in conjunction with the use of SLPv2, the major RFC 3723 recommendations are not necessary as long as the SLP messages are not fully trusted and SSL or TLS with server certificates are used. Additional security guidance is provided in the sections associated with UAs and SAs.

10.3 SLP Messages

SLP v2 divides the base set of SLP messages into required and optional subsets.

Note: SLP v2 also includes a new feature, an extension format. Extension messages are attached to base messages. SMI-S does not use extensions. The discussion of messages introduces the following terms that define the SLP services:

Attribute Reply (AttrRply): A reply to an Attribute Request. (optional)

Attribute Request (AttrRqst): A request for attributes of a given type of service or attributes of a given service. (optional)

DA Advertisements (DAAdvert): A solicited (unicast) or unsolicited (multicast) advertisement of Directory Agent availability.

SA Advertisement (SAAdvert): Information describing a service that consists of the Service Type, Service Access Point, lifetime, and Attributes.

Service Acknowledgement (SrvAck): A reply to a SrvReg request.

Service Deregister (SrvDereg): A request to deregister a service or some attributes of a service. (optional)

Service Register (SrvReg): A request to register a service or some attributes of a service.

Service Reply (SrvRply): A reply to a Service Request.

Service Request (SrvRqst): A request for a service on the network.

Service Type Reply (SrvTypeRply): A reply to a Service Type Request. (optional)

Service Type Request (SrvTypeRqst): A request for all types of service on the network. (optional)

Service Agents (SAs) and User Agents (UAs) shall support Service Request, Service Reply, and DAAdvertisement message types. Service Agents shall additionally support Service Registration, SA Advertisement, and Service Acknowledgement message types. The remaining message types may be supported by Service Agents and User Agents. Directory Agents (DAs) shall support all message types with the exception of SA Advertisement. Table 1229, “Message Types” lists each base message type, its abbreviation, function code, and required/optional status.

Table 1229: Message Types

Message Type	**Abbreviation**	**Function Code**	**Required (R)/ Optional (O)**		
			DAs	**SAs**	**UAs**
Service Request	SrvRqst	1	R	R	R
Service Reply	SrvRply	2	R	R	R
Service Registration	SrvReg	3	R	R	O
Service Deregistration	SrvDereg	4	R	O	O
Service Acknowledge-ment	SrvAck	5	R	R	O
Attribute Request	AttrRqst	6	R	R	R
Attribute Reply	AttrRply	7	R	R	R
DA Advertisement	DAAdvert	8	R	R	R
Service Type Request	SrvTypeRqst	9	R	O	O
Service Type Reply	SrvTypeRply	10	R	O	O
SA Advertisement	SAAdvert	11	N/A	R	O

Note: The requirements in Table 1229 extend the requirements defined for SLP V2. SMI-S adds additional requirements for AttrRqst and AttrRply beyond those defined by the RFC.

10.4 Scopes

SLPv2 defines a scope as follows:

Scope: A set of services, typically making up a logical administrative group.

Scopes are sets of service instances. The primary use of Scopes is to provide the ability to create administrative groupings of services. A set of services may be assigned a scope by network administrators. A User Agent (UA) seeking services is configured to use one or more scopes. The UA only discovers those services that have been configured for it to use. By configuring UAs and Service Agents with scopes, administrators may make services available. Scopes strings are case insensitive. The default SCOPE string is “DEFAULT”.

SMI-S does not dictate how Scopes are set. That is, scopes can be set by customers to match their needs. However, SMI-S requires that SMI-S servers use the “default” scope as a means of making SMI-S advertisements visible to SMI-S clients.

To be compliant with SMI-S, User Agents (SMI-S clients) and Service Agents (SMI-S servers) shall not require scope settings that interfere with administrative use of scopes. Specifically, this means:

- SMI-S clients and servers shall allow an administrator to set scopes to define what is to be searched, and,
- SMI-S clients and servers shall allow an administrator to configure scopes, including turning off the “default” scope.

10.5 Services Definition

Services definition uses the following terms defined in SLPv2:

Service Type Template: A formalized, computer-readable description of a Service Type. The template defines the format of the service URL and attributes supported by the service type.

Service URL: A Uniform Resource Locator for a service containing the service type name, network family, Service Access Point, and any other information needed to contact the service.

Services are defined by two components: the Service URL and the Service Type Template. The Service URL defines an access point for the service and identifies a unique resource in the network. Service URLs may be either existing generic URLs or URLs from the service: URL scheme.

The second component in a Service definition is a Service Type Template. Service Type Templates define the attributes associated with a service. These attributes, through inclusion in registrations and queries, allow clients to differentiate between similar services.

SMI-S servers use a Service Type Template defined by DMTF for advertising “WBEM Servers” (e.g., CIMOMs). The template name for WBEM Servers is “WBEM”.

10.5.1 Service Type

Service Type: The class of a network service represented by a unique string (for example a namespace assigned by IANA).

The service type describes a class of services that share the same attributes (e.g., the service printer or the service “WBEM”). DMTF is considering an SLP-based discovery mechanism that locates “WBEM” (e.g., CIMOMs). The SMI-S design builds on the DMTF proposal.

The basic function of SLP discovery is the identification of the service offered by a constituent. In the case of SMI-S, the service type advertised by all constituents is “WBEM.” This follows a DMTF proposal for advertising WBEM Servers. The only exception to this is the Directory Server, which advertises itself as a “directory-agent.” That is, SMI-S uses a standard SLP directory service. SMI-S does not require a unique SMI-S directory server.

For other roles (SMI-S servers) the role advertises its services as a WBEM services (e.g., “WBEM”).

10.5.2 Service Attributes

Attributes: A collection of tags and values describing the characteristics of a service.

SMI-S servers shall advertise a standard set of attributes. These attributes are the following:

- Service-hi-name – This is the name of the service for use in human interfaces.
- Service-hi-description – This is a description of the CIM service that is suitable for use in human interfaces.
- Service-id – A unique id for the CIM Server that is providing the service.
- Service-location-tcp – This is a list of TCP addresses that can be used to reach the service. NOTE: This need only be one (for CIM-XML). But it could hold others (for other communications protocols).
- CommunicationMechanism – “cim-xml” (at least). The SMI-S server could support others, but “cim-xml” is mandatory for SMI-S servers.
- OtherCommunicationMechanismDescription – used only if “other” is also specified for CommunicationMechanism.
- InteropSchemaNamespace – The Namespace within the SMI-S server where the CIM Interop Schema can be accessed. Each namespace provided shall contain the complete information and if multiple namespaces are provided they shall contain the same information. Even though multiple

InteropSchemaNamespaces may be provided, an SMI-S client may rely on the first namespace as the definitive namespace for accessing the Interop Schema (including the class instances of the Server Profile).

- ProtocolVersion – The Version of the cim-xml protocol if this is the defined. This is mandatory for SMI-S servers.
- FunctionalProfilesSupported: Permissible values are “Unknown”, “Other”, “Basic Read”, “Basic Write”, “Schema Manipulation”, “Instance Manipulation”, “Association Traversal”, “Query Execution”, “Qualifier Declaration”, “Indications”. This defines the CIM Operation Profiles supported by the SMI-S server. Can return multiple values.
- FunctionalProfileDescriptions - If the “other” value is used in the FunctionalProfilesSupported attribute, this shall be populated. If provided it shall be derived from the CommunicationMechanism.FunctionalProfileDescriptions property. Use of this attribute is not specified by SMI-S.
- MultipleOperationsSupported – A Boolean that defines whether the SMI-S server supports batch operations.
- AuthenticationMechanismsSupported – Permissible values are “Unknown”, “None”, “Other”, “Basic”, “Digest”. Defines the authentication mechanism supported by the SMI-S server. Can return multiple values.
- AuthenticationMechanismDescriptions - Defines other Authentication mechanism supported by the SMI-S server. The value shall be supplied if the “Other” value is set in the AuthenticationMechanismSupported attribute. This attribute is optional. It is to be provided only when the AuthenticationMechanismSupported attribute is “other”.
- Namespace - Namespace(s) supported on the SMI-S server. This attribute may have multiple values (one for each namespace defined in the SMI-S server), and is literal (L) because the namespace names may not be translated into other languages.
- Classinfo - The values are taken from the interop schema Namespace.classinfo property. The values represent the classinfo (CIM Schema version, etc.) for the namespaces defined in the corresponding namespace listed in the namespace attribute. Each entry in this attribute shall correspond to the namespace defined in the same position of the namespace attribute. There shall be one entry in this attribute for each entry in the namespace attribute.
- RegisteredProfilesSupported – The SMI-S profile(s) supported by the server, prefixed by “SNIA” (at least). An SMI-S server may also support other RegisteredProfiles, but it shall support at least one “SNIA” profile. In addition, this attributed can also be used to advertise subprofiles, when subprofiles are to be advertised. The RegisteredProfilesSupported is an array. Each entry includes a RegisteredOrganization (i.e., SNIA), a Profile name and an optional subprofile name. Each name is separated by a colon.

Note that a single SMI-S server can support multiple profiles. As a result, the profile attribute is an array of values.

Additional attributes, such as specific profile services supported, model subprofiles supported and the SMI-S release level are not discovered via SLP. They would be found by traversing the model presented by the SMI-S server.

10.6 User Agents (UA)

A User Agent is a Client process working on the user’s behalf to establish contact with some service. A User Agent retrieves service information from Service Agents (See 10.7, "Service Agents (SAs)") or

Directory Agents (10.8, "Directory Agents (DAs)"). Further description of a Client and its role may be found in 9.2, "SMI-S Client".

The only required feature of a User Agent is that it can issue SrvRqsts and interpret DAAdverts, SAAdverts and SrvRply messages. If Directory Agents exist, User Agents shall issue requests as Directory Agents are discovered.

An SMI-S Client should act as an SLP user agent (UA) using the query functions of SLP V2 to determine location and other attributes of the “WBEM” SLP Service Type Template defined in 10.10, "‘Standard WBEM’ Service Type Templates".

The basic search methodology for SMI-S clients is to search for directory agents and service agents within their scope. If all SMI-S servers are supported by a directory agent, then the search yields nothing but directory agents. The client can then obtain a list of services (and their URLs) for management of the SMI-S servers.

If any Service agents are not covered by a directory agent (i.e., are not within its scope), then the client obtains service replies from those service agents.

An client would typically search for all service types available in their scope(s). This returns a list of service types available in the network. However, an SMI-S client can be assumed to be searching for “WBEM” service types. If a client only manages selected devices (e.g., switches or arrays), the SMI-S client can issue a request for the specific services by using predicates on the “RegisteredProfilesSupported” attribute.

When a SMI-S client uses SLPv2 and security is an issue, the following should be considered:

- SSL and TLS should be used with a certificate-based cipher suite along with a certificate installed on each SMI-S server (SA) for communications with discovered SAs (SMI-S servers).
- SLPv2 Service Agents (SA) and Directory Agents (DA) may advertise (SAAdverts and DAAdverts, respectively) their presence on the network, using multicast; however, SMI-S clients should treat these advertisements as advisory (i.e., identity must be verified as described below).
- SMI-S clients should maintain and use a negative authentication cache to avoid repeatedly contacting an SMI-S server that fails to authenticate as part of the SSL or TLS handshake.

10.7 Service Agents (SAs)

A Service Agent supports an SMI-S server process working on behalf of one or more services to advertise the services.

See Clause 9:, "SMI-S Roles" for further description of SMI-S servers.

Service Agents shall accept multicast service requests and unicast service requests. SAs may accept other requests (Attribute and Service Type Requests). An SA shall reply to appropriate SrvRqsts with SrvRply or SAAdvert messages. The SA shall also register with all DAs as they are discovered.

To provide for SMI-S Client discovery of SMI-S servers, a CIM Server shall act as a Service agent (SA) for the IETF Service Level Protocol (SLP) V2 as defined in IETF RFC 2608. The service shall correspond to V2 of SLP (IETF RFC 2608 and 2609) and shall use the Service Templates defined in 10.10, "‘Standard WBEM’ Service Type Templates" of this specification for advertisements. An SMI-S server acting as an SA shall provide a separate SLP advertisement for each address/port that the CIM Server advertises.

When a SMI-S server uses SLPv2 and security is an issue, the following should be considered:

- SMI-S servers should accept SSL and TLS unicast connections from SMI-S clients as well as selecting a certificate-based cipher suite.
- SMI-S servers that advertise their existence as SLPv2 SAs (SAAdverts) should minimize leakage of information, by minimizing the information that is contained in the multicast advertisements.
- SMI-S servers, functioning as SAs, should register with all discovered DAs, which advertise any of its configured scopes and establish connections with these DAs over unicast.
- When SMI-S servers are also functioning as clients (e.g., cascading), they should follow the security guidance provided in 10.6.

10.8 Directory Agents (DAs)

SMI-S supports existing SLPv2 Directory Agents (without modification). That is, SMI-S makes no assumptions on Directory Agents that are not made by SLPv2. Note that this cannot quite be said for User Agents, which are looking for SMI-S specific services, or Service Agents, which are advertising SMI-S specific services.

10.9 Service Agent Server (SA Server)

10.9.1 General Information

The reserved listening port for SLP is 427, the destination port for all SLP messages. Service Agents (SAs) are required to listen for both unicast and multicast requests. A Directory Agent (DA) shall listen for unicast request and specific multicast DA discovery service requests. SAs and User Agents (UAs) that perform passive DA discovery shall listen for multicast DA Advertisements (DAAdverts).

TCP/IP requires that a single server process per network interface control all incoming messages to a port. That requirement necessitates a mechanism to share the SLP port (427).

Sharing the SLP port (427) is accomplished with a Service Agent Server (SA Server) process that ‘owns’ the port on behalf of all SAs, UAs and optional DA that are listening for SLP messages. The SA Server listens for incoming messages that request advertisement information and either answer each request or forward it to the appropriate SA. The SA Server also performs passive DA discovery and distribute the DA addresses and scopes to the SAs and UAs that it serves.

A SA Server may also function as a DA if the SA Server is implemented so that it answers requests for advertisement information rather than forwarding each request to the appropriate SA. The combined DA/SA Server is acting as an intermediary between a SA that registered an advertisement and a UA requesting information about the advertisement.

10.9.2 SA Server (SAS) Implementation

The RFC 2614 document describes APIs for both the C and Java languages. Both APIs are designed for standardized access to the Service Location Protocol (SLP).

The goals of the C API are:

- Directly reflect the structure of SLP messages in API calls and return types as character buffers and other simple data structures.
- Simplify memory management to reduce API client requirements.
- Provide API coverage of just the SLP protocol operations to reduce complexity.

- Allow incremental and asynchronous access to return values, so small memory implementations are possible.
- Support multithreaded library calls on platforms where thread packages are available.

The Java API goals are:

- Provide complete coverage of all protocol features, including service type templates, through a programmatic interface.
- Encourage modularity so that implementations can omit parts of the protocol that are not needed.
- In conformance with Java's object-oriented nature, reflect the important SLP entities as objects and make the API itself object-oriented.
- Use flexible collection data types consistently in the API to simplify construction of parameters and analysis of results.
- Designed for small code size to help reduce download time in networked computers.

10.9.3 SA Server (SAS) Clients

10.9.3.1 Description

An SAS Client is a Service Agent (SA), User Agent (UA), or Directory Agent (DA) that is associated with a SA Server. The SA Server listens on the SLP port (427) and appropriately handle all incoming messages for each SAS Client. A DA acting as a SAS Client is separately configured on the same host as the SA Server.

10.9.3.2 SAS Client Requests – SA Server Responses

A SA Server responds when appropriate, to incoming unicast and multicast messages from SAS Clients. The SA Server may answer with the appropriate advertisement, if available, or forward the request on to the appropriate SAS Client. If the SA Server is also functioning as a DA, it discards a multicast SrvRqst of "service:directory-agent" that has either a missing scope list or the scope list does not contain a scope the Service Agent Server/DA is configured with.

10.9.4 SA Server Configuration

10.9.4.1 Overview

SA Servers may be configured via an individual SLP configuration file, programmatically, or a combination of the two. DHCP may also be used obtain the scope list for a SA Server. Figure 225: "SA Server Configuration" illustrates the various means of configuring a SA Server.

10.9.4.2 SLP Configuration File

10.9.4.2.1 If a SA Server is also functioning as a DA, the following DA configuration properties shall be set:

Table 1230: Required Configuration Properties for SA as DA

Keyword	Data Type	Value
net.slp.isDA	boolean	true
net.slp.DAAttributes	string	(SA-Server=true)

The DA attribute/value pair of "SA-Server=true" allows a query to be used when a SA Server/DA needs to be identified. In addition, when the SA Server/DA responds to a SrvRqst message with a DAAdvert message, the DA attribute/value pair is included.

10.9.4.2.2 The remaining DA configuration property, net.slp.DAHeartBeat, with a default of 10,800 seconds, may be set as appropriate. If a SA Server is not functioning as a DA, the following SA configuration property shall be set:

Table 1231: Required Configuration Properties for SA

Keyword	Data Type	Value
net.slp.SAAttributes	string	(SA-Server=true)

10.9.4.3 Programmatic Configuration

Both the C and Java language API's provide access to SLP properties contained in the SLP configuration file. The actual SLP configuration file is not accessed or modified via the interfaces. Once the file is loaded into memory at the start of execution, the configuration property accessors work on the in-memory representation.

The C language API provides the `SLPGetProperty`() and `SLPSetProperty`() functions. The `SLPGetProperty`() function allows read access to the SLP configuration properties while the `SLPSetProperty`() function allows modification of the configuration properties.

The SLPSetProperty() function has the following prototype:

```
void SLPSetProperty(const char *pcName, const char *pcValue);
```

The SLPSetProperty() function takes two string parameters: pcName and pcValue. The pcName parameter contains the property name and pcValue contains the property value. The following example uses the SLPSetProperty() function to configure a SA Server that is not functioning as a DA:

```
void setSAAttributes() {
char value[80]; /* A buffer for storing the attribute string. */
value = “SA Server=true”;
SLPSetProperty(“net.slp.SAAttributes”, value);
}
```

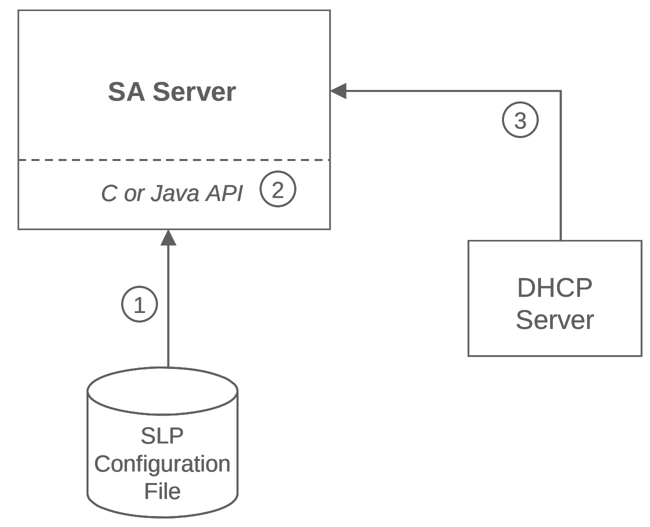
10.9.4.4 DHCP Configuration

If the Service Agent Server is also functioning as a DA, its scope list may be obtained via DHCP. Scopes discovered via DHCP take precedence over the net.slp.useScopes property in the SLP configuration file.

10.9.4.5 Scope

A Service Agent Server is configured with a minimum scope of DEFAULT. If a Service Agent Server is not functioning as a DA, DEFAULT is the only scope configured. If a Service Agent Server is functioning as a DA, it may have additional scopes configured. Use of the DEFAULT scope enables the associated SAS Clients (UAs, SAs and DA) to actively discover the Service Agent Server using a well-known value for scope.

Figure 225: SA Server Configuration



1) The SA Server may obtain specific configuration values via an individual SLP Configuration file.

2) The C or Java API provides programmatic access to the configuration file properties.

3) The SA Server may obtain its scope values from a DHCP Server.

10.9.5 SA Server Discovery

"Discovery" of a SA Server by its SAS Clients is accomplished by successfully establishing the required communication link between the two entities. There is no need for active or passive discovery as described by SLP since both the SA Server and SAS Clients reside on the same host system.

10.9.6 SAS Client Registration

Service Agents (SAs) that are SAS Clients register and deregister with the local SA Server using the SrvReg/SrvDereg messages. The SA Server responds with a Service Acknowledgement (SrvAck) message. The SA Server store a service advertisement until either its lifetime expires or a SrvDereg message is received.

If the SA Server is also functioning as a DA, the DA registration requirement is also met. The SA server also forwards any SA registration to other DAs that have the same scope as the SA.

10.10 'Standard WBEM' Service Type Templates

Note: For each description in the template that states the value shall be the ClassName.PropertyName value, the format/rules for these values are defined in the Interop Model of the CIM Schema and in the "Server Profile" section of this specification. This SLP Template requires a minimum Schema version of 2.7 to support the required values. Some of the optional values require CIM Schema version 2.8.

```
Name of submitter: "DMTF" <technical@dmtf.org>


Language of service template: en
```

```
Security Considerations:

Information about the specific CIM Server implementation or the
Operating System platform may be deemed a security risk in certain
environments. Therefore these attributes are optional but
recommended.

Template Text:

-------------------------template begins here-----------------------
template-type=wbem

template-version=1.0

template-description=
    This template describes the attributes used for advertising
    WBEM Servers.

template-url-syntax=string
#The template-url-syntax MUST be the wbem URI encoding of
#the location of one service access point offered by the WBEM Server
#over TCP transport. This attribute must provide sufficient addressing
#information so that the WBEM Server can be addressed directly using
#the url.

service-hi-name=string O
# This string is used as a name of the CIM service for human
# interfaces. This attribute MUST be the
# CIM_ObjectManager.ElementName property value.

service-hi-description=string O
# This string is used as a description of the CIM service for
# human interfaces.This attribute MUST be the
# CIM_ObjectManager.Description property value.

service-id=string L
# The ID of this WBEM Server. The value MUST be the
# CIM_ObjectManager.Name property value.

CommunicationMechanism=string L
# The communication mechanism (protocol) used by the CIM Object Manager for
# this service-location-tcp defined in this advertisement. This information
# MUST be the CIM_ObjectManagerCommunicationMechanism.CommunicationMechanism
# property value.
# CIM-XML is defined in the CIM Operations over HTTP specification which can
# be found at http://dmtf.org/
```

```
“Unknown”, “Other”, “cim-xml”

OtherCommunicationMechanismDescription = String L O
# The other communication mechanism defined for the CIM Server in the case
# the “Other” value is set in the CommunicationMechanism string.
# This attribute MUST be the
CIM_ObjectManagerCommunicationMechanism.OtherCommunicationMechanism
# property value. This attribute is optional because it is only required if the
# “other” value is set in CommunicationMechansim. The value returned is
# a free-form string.

InteropSchemaNamespace=string L M
# Namespace within the target WBEM Server where the CIM Interop Schema can be
# accessed. Multiple namespaces may be provided. Each namespace provided
# MUST contain the same information.

ProtocolVersion=String O L
# The version of the protocol. It MUST be the
# CIM_ObjectManagerCommunicationMechanism.Version property value.

FunctionalProfilesSupported=string L M
# ProfilesSupported defines the CIM Operation profiles supported by the
# CIM Object Manager. This attribute MUST be the
# CIM_ObjectManagerCommunicationMechansim.FunctionalProfilesSupported
# property value.
“Unknown”, “Other”, “Basic Read”, “Basic Write”,
“Schema Manipulation”, “Instance Manipulation”,
“Association Traversal”, “Query Execution”,
“Qualifier Declaration”, “Indications”

FunctionalProfileDescriptions=string L O M
# Other profile description if the “other” value is set in the ProfilesSupported
# attribute.  This attribute is optional because it is returned only if the “other”
# value is set in the ProfilesSupported attribute. If provided it MUST
# be equal to the
CIM_ObjectManagerCommunicationMechanism.FunctionalProfileDescriptions
# property value.

MultipleOperationsSupported=Boolean
# Defines whether the CIM Object Manager supports batch operations.
# This attribute MUST be the
# CIM_ObjectManagerCommunicationMechanism.MultipleOperationsSupported
# property value.

AuthenticationMechanismsSupported=String L M
# Defines the authentication mechanism supported by the CIM Object Manager.
# This attributed MUST be the
# CIM_ObjectManagerCommunicationMechanism.AuthenticationMechanismsSupported
property value.
“Unknown”, “None”, “Other”, “Basic”, “Digest”

AuthenticationMechansimDescriptions=String L O M
# Defines other Authentication mechanisms supported by the CIM Object Manager
# in the case where the “Other” value is set in any of the
# AuthenticationMechanismSupported attribute values. If provided, this attribute
MUST be the
# CIM_ObjectManagerCommunicationMechanism.AuthenticationMechansimDescriptions
# property value.

Namespace=string L M O
# Namespace(s) supported on the CIM Object Manager.
# This attribute MUST be the
# CIM_Namespace.name property value for each instance of CIM_Namespace
# that exists. This attribute is optional.
# NOTE: This value is literal (L) because
# the namespace names MUST not be translated into other languages.

Classinfo=string M O
# This attributes is optional but if used, the values MUST be the
# CIM_Namespace.classinfo property value.
# The values represent the classinfo (CIM Schema version, etc.) for
# the namespaces defined in the corresponding namespace listed in the
# Namespace attribute. Each entry in this attribute MUST correspond
# to the namespace defined in the same position of the namespace
# attribute. There must be one entry in this attribute for each
# entry in the namespace attribute.


RegisteredProfilesSupported=string L M
# RegisteredProfilesSupported defines the Profiles that
# this WBEM Server has support for. Each entry in this
# attribute MUST be in the form of
# Organization:Profile Name{:Subprofile Name}
#
# examples:
#     DMTF:CIM Server
#     DMTF:CIM Server:Protocol Adapter
#     DMTF:CIM Server:Provider Registration
# The Organization MUST be the
# CIM_RegisteredProfile.RegisteredOrganization property value.
# The Profile Name MUST be the
# CIM_RegisteredProfile.RegisteredName property value.
# The subprofile Name MUST be the
# CIM_RegisteredProfile.RegisteredName property value when it is
# used as a Dependent in the CIM_SubProfileRequiresProfile
# association for the specified Profile Name (used as the antecedent).


--------------------------template ends here------------------------
```

10.11 SLP Bibliography

The following reference materials on SLP are recommended to assist in vendor implementations of SLP processes.

Kempf, J. and P. St. Pierre. _Service Location Protocol for Enterprise Networks_. New York: John Wiley and Sons, Inc., 1999.

Perkins, C. and E. Guttman. “DHCP Options for Service Location Protocol.” IETF RFC 2610, June 1999.

Guttman, E., C. Perkins, and J. Veizades, and M. Day. “Service Location Protocol, Version 2." IETF RFC 2608, June 1999.

Guttman, E., C. Perkins, and J. Kempf. “Service Templates and service: Schemes.” IETF RFC 2609, June 1999.

Kempf, J. and E. Guttman. “An API for Service Location.” IETF Informational RFC 2614, June 1999.

Guttman, E. “The serviceid: URI Scheme for Service Location.” draft-guttman-svrloc-serviceid-01.txt, IETF Informational Draft, Network Working Group, January 4, 1999

Clause 11: Installation and Upgrade

11.1 Introduction

The interoperability of the management communications in a storage network gives customers a choice in vendors of their management solutions, but it also can introduce ease-of-use problems when these different vendors each supply the functions shown in Figure 3: "Example Client Server Distribution in a SAN". In order to supply a complete management solution, many management vendors provide not only WBEM Clients, Providers and other Management Interfaces, but also software components that provide other pieces of the management infrastructure (e.g., Directory Services, WBEM Services, Database Management). Problems are possible when multiple vendors install or remove these components in the same configuration and conflicts can arise. One of the goals of creating management interoperability is to reduce the time and expense end-users apply to the management of their SANs. Thus, SAN management should be easy to install, easy to upgrade, and easy to reconfigure. Mature management products using SMI-S technology should experience seamless and almost completely automated installation, upgrade, and reconfiguration.

This clause deals with issues in installation, upgrade and uninstallation of products using SMI-S technology, and recommends some steps that vendors should take to minimize the problems, leading to better customer satisfaction with the overall management solution.

11.2 Role of the Administrator

Ultimately, a vendor’s installation software cannot make perfect decisions when the conflicts referenced above arise, since there may be valid reasons why a customer has deployed software of similar function from multiple vendors. In the situation where two software components are both installed that perform the same shared function, and only one can reasonably operate without conflicts, the administrator must be able to resolve these conflicts and remove or disable the redundant component(s).

Installation software should, however, make a best effort to detect any conflicts and notify the administrator of possible conflicts during its installation and initialization. A vendor’s installation software should allow the administrator to install and uninstall the various infrastructure components on an individual basis should such a conflict arise. The implications of this are that vendors are motivated to support interoperation with other vendor’s components. The advantage to the vendor is that a customer is more likely to install a component that can demonstrate the most interoperability with other components.

11.3 Goals

11.3.1 Non-Disruptive Installation and De-installation

WBEM Clients & Services, Providers, and Directory Services may be capable of being installed and de-installed without disrupting the operation of other constituents in a SMI-S management environment. As SANs are often deployed in mission critical environments the up-time of the solution is critical and thus, the uptime of the management backbone as a key component of the solution is equally critical. Additionally, the installation and de-installation of SMI-S interface constituents should not compromise the availability of mission critical applications.

11.3.2 Plug-and-Play

The ultimate goal of management interoperability is zero administration of the management system itself. A customer should be able to install new storage hardware and software and have the new component become part of the management system automatically. Use of the Service Discovery process (see Clause 10:, "Service Discovery"), the discovery-related aspects of the SMI-S Role definitions (see Clause 9:, "SMI-S Roles"), and the Server profile (see 8.2.4.1, "Server Profile") are intended to assist in achieving this goal.

During the reconfiguration of the management system, the schema that Clients see should remain consistent (Schema forward compatibility is ensured via CIM standard).

11.4 Device Support

Manufacturers of storage hardware and software typically install their product and the accompanying management support at the same time. The SMI-S Reference model (see 4.3, "Reference Model") defines a number of different models for this management support.

Conflicts are possible between Agents if multiple vendors attempt to install support for the same device. Also, when a device vendor needs to upgrade an Agent or Provider for a device, the installation software needs to determine all of the locations of the previous installations to insure there is not duplicate management paths to the device and thus, insure reliable on-going operation of the device.

11.4.1 Installation

Installation software for devices needs to be able to locate existing CIM Servers that may control the device in order to offer an administrator a choice in management constituents for the device. In addition, the installation software may desire to locate existing Agents and Providers that provide device support in order to reliably upgrade that support. For these reasons, an installation software program may want to act as a SMI-S Client during installation. This will allow it to employ the Service Discovery (see Clause 10:, "Service Discovery") to locate the appropriate functions, and to make the automated decisions that eliminate the need for an administrator to manually configure or adjust certain aspects of the management system.

The RegisteredProfile part of the model described in Server Profile in 8.1.4.1 shows what device support is already installed and installation software should consult this schema before installing new software. If the installation software is changing the device support from one configuration to another, the installation software needs to uninstall or disable the previous software support elements.

11.4.2 Discovery and Initialization of Device Support

The SMI-S Reference Model (see 4.3, "Reference Model") defines two “Proxy Models” in which management support is provided via an Agent or through an Object Manager (with providers). In these models, the device support is expected to provide a means for establishing a reliable connection between the device itself and the Agent or Object Manager. Also, a special Client with administration/ installation capability (as supplied by the vendor) is required to supply the relevant credentials for device access to the Agent or Object Manager designated to manage the device. This special Client may obtain the IP address of the device via automated means (not defined in this standard) or via manual means (e.g., by requiring a system manager to manually input the IP address of the device/ subsystem from documentation supplied by the vendor).

11.4.3 Uninstallation

During the uninstallation of a device, the installation/uninstallation software (if available) should automatically detect existing management support software for the device in order to shutdown and remove it in a consistent manner. This detection process need to be cognizant that SMI-S Clients may be actively using the device and that the device may need to be disabled for new management operations and administrated through an orderly shut-down procedure prior to uninstallation. The implementation of such procedures and any order dependency is outside the scope of this specification, but may need to be considered by implementors.

11.4.4 Update

During the update of device support software, installation software should automatically detect any existing device support software in order to successfully complete the upgrade. This device support may exist on multiple hosts, but that situation is not specified in this version. If the update includes installing a new provider, the installation software needs to use the provider installation/upgrade method that is supported by the existing Object Manager.

When a software update involves a major schema version upgrade (e.g., 2.x to 3.x), the installation software needs to be cognizant of the effect of the schema upgrade on existing clients. For example, it may choose to simultaneously support both versions for some period of time.

11.4.5 Reconfiguration

When device support update requires an update of an agent or provider, the device support installation software should configure the new provider with the same subscriptions that exist in the old agent or provider before removing it, unless those subscriptions are specifically defined as being periodically cleaned up. This can be done via the instances of the subscriptions in the agent or object manager that currently exist.

11.4.6 Failure

Agents can become unavailable for several reasons. This includes the managed device being powered off and transient network failures. If a device’s model becomes unavailable, it is recommended that Clients do not immediately remove that device from its visualization. If the device model reappears in another location, the old visualization should be updated to remove the previous occurrence. Also, the client can keep track of how long the device was down for purposes of availability management, etc. Clients may have to restore indication subscriptions when the agent subsequently becomes available. In the case of the two Proxy Models in the SMI-S Reference Model, the agent (or its host, or the Object Manager) may go down, or its network connection could fail, but the device may still be available and this needs to be considered in designing availability management. In the case of a provider, the provider to device communication channel may also fail, but the device may still be available for access.

11.5 WBEM Service Support & Related Functions

11.5.1 Installation

Customers are increasingly sensitive to the size of the memory footprint for management software. The goal is to minimize the impact on hosts that are not dedicated to running management software by making appropriate choices during installation and giving the administrator control over these issues.

It is recommended that vendors take advantage of an existing Object Manager where one exists, by installing a provider that communicated with that Object Manager for device support. Additional support for such “multi-tenant” Object Managers will be included in a future version of this document.

If an object manager does not exist, or the device support does not work with the existing object manager (e.g., due to interface requirements) it is recommended that the vendor supply a Agent that is lightweight for device support. Another option is to offer to install an Object Manager that the vendor does have provider support for, allowing other vendors to further leverage that installation.

Providers that use an in-band connection to devices have an issue where zoning may alter the management path to the device from a provider or agent. In this case, the device support may need to be installed on multiple hosts in the network and the vendor needs to provide some way to coordinate which provider or agent is responsible for a particular device.

Vendors should install their providers in a unique namespace for isolation and qualification reasons. The installer should employ the Service Discovery process (see Clause 10:, "Service Discovery"), and/or the Server profile (see 8.2.4.1, "Server Profile") to discover the existing namespaces and insure that the one created for the new device is truly unique.

11.5.2 Multiple CIM Servers on a Single Server System

At installation and setup, a user interface should be provided by the CIM Server installation utility that allows an administrator to manually set the TCP port number in a persistent fashion.

To support discovery, the SLP Service Agent (see 10.7, "Service Agents (SAs)") associated with a newly-installed CIM Server should register its TCP port number along with all the other necessary discovery information with the Discovery Service. This requirement applies to both automated port selection as well as manually configured installations. Clients, working through their SLP User Agent (see 10.6, "User Agents (UA)"), then use this information to establish contact with the CIM Server.

11.5.3 Uninstallation/Upgrade

An Object Manager may be upgraded without needing to change the Providers that it supports. Depending on the Object Manager, the Providers may have to be reinstalled and reconfigured following such an upgrade. In this case, an administrator may need to re-run the device support installation software and such software should be able to restore the previous configuration.

11.5.4 Reconfiguration

Device Support Reconfiguration (see 11.4, "Device Support") identifies issues that may also be applicable to Object Managers.

11.5.5 Failure

Temporary failure of an object manager (for example, a host being powered off) can result in bad installation decisions for installation software. In this case, it is advisable that the installation software provide for manual input of the characteristics of additional components of the management system that the installation process needs to consider.

11.6 Client

11.6.1 Uninstallation

When Client software is removed, the uninstallation software should ensure that all client-defined information (settings, policies etc.), and any subscriptions for that client that exist in any agent or object manager, are also removed.

11.6.2 Reconfiguration

Client software can include a Listener that is configured to listen on a specific port. When this port is reconfigured, the client should redirect any Indication Handlers in existing agent and object managers as a result.

11.7 Directory Service

11.7.1 Installation

The installation of more than one Directory Agent (see 10.6, "User Agents (UA)") or Service Agent Server (see 10.7, "Service Agents (SAs)") providing a Directory Service in a management system does not impose a significant burden for management clients and adds to the overall availability. Vendors should recommend to administrators of their products that one or more SA Servers or Directory Agents should be deployed in the management system. This may also be done for network or system management reasons.

11.7.2 Uninstallation/Failure

SLP Clients are defined to handle failure and uninstallation of DAs as per the specification (see Clause 10:, "Service Discovery").

11.8 Issues with Discovery Mechanisms

Experience with existing SMI-S installations has indicated that some sites have policies that can impact the Service Discovery process (see Clause 10:, "Service Discovery"). This subject will be addressed in greater detail in a future revision of this document, but two specific items of guidance are given here, as follows:

a) Where the site policy has caused multicast to be disabled, the DHCP option for SLP defined in RFC 2610 is recommended as an alternate method of locating Service Agent Servers or Directory Agents. Also note that the shipping configuration of many network routers has multicast disabled.

b) Where the site policy has caused support for SLP itself to be disabled, an out of band method of providing a list of IP addresses for CIM Servers is recommended, after which the Server profile (see 8.2.4.1, "Server Profile") should be used to obtain the information about Registered Profiles usually retrieved via SLP.

Annex A: (Informative) Mapping CIM Objects to SNMP MIB Structures

A.1 Purpose of this appendix

In order to encourage adoption of the WBEM initiative, its associated data model (CIM), protocol (xmlCIM), and profiles (described in previous sections of this specification), the Storage Media Library (SML) workgroup defined a means of mapping CIM objects to SNMP MIB objects, or ìfields.î At the time of this writing, SNMP (Simple Network Management Protocol) is the dominant non-proprietary network management protocol used by the storage devices described above. This ìCIM-to-MIBî mapping methodology has been successfully used by members of SNIA-SML to demonstrate ñ at minimal cost in development time -- WBEM-based interoperability in ìplugfests and industry demonstrations such as Storage Networking World. The SML workgroupís ìCIM-to-MIBî mapping methodology is mentioned in this specification in order to:

- Document that a standard path of backward compatibility is obtainable between WBEM and SNMP-based management paradigms,
- Document one successful method of CIM-to-MIB mapping,
- Recommend this method as *the* standard CIM-to-MIB mapping method in order to avoid a proliferation of deviant *de facto* standards, and
- Allow SNIA member companies outside the SML workgroup to benefit from earlier experience and work.

A.2 CIM-to-MIB Mapping Overview

CIM is an object-based modeling schema that supports all common object-oriented principles, including abstract class objects, instance objects, inheritance, single- and multiple-association, aggregation, properties, methods, and qualifiers. In contrast, SNMPís ASN.1-based modeling schema is strictly hierarchical, involving such structures as nested parent and child nodes, and scalar and tabular fields. While unique CIM objects are typically referenced by parent class name (or Creation Class Name) and key properties, SNMP objects are typically referenced by an Object Identifier (OID) that points to their position in the SNMP Management Information Base (MIB) hierarchy or ìtree.î (In the case of tabular fields, additional indexes are appended to a base OID to identify unique instances of information.) The task of any CIM-to-MIB mapping methodology is primarily to create a one-to-one mapping between object-oriented information and tree-based hierarchical information. Naming constraints within the CIM and MIB domains must also be adhered to in a way that prevents ambiguities in uniquely identifying and referencing information, particularly in the SNMP/MIB domain. Therefore, SMLs mapping methodology provides the following:

- A description of mapping CIM data -- classes, instances, properties, associations ñ into an SNMP format involving nodes, fields, and tables,
- A naming convention in the SNMP/MIB domain that allows for unambiguous identification of the original CIM data,
- A data type mapping that allows common CIM data to be represented by existing ASN.1 data types.

A.3 The SML MIB

As the CIM object model continues to change and expand, the SML MIB has also changed and expanded. As a result, it has become impractical to include the full MIB in each revision of this SMI specification.

SMI client application vendors or others interested in obtaining the latest SML MIB, or more information on the CIM-to-MIB mapping methodology in general, should contact the Technical Council Managing Director at tcmd@snia.org.

Annex B: (Normative) Compliance with the SNIA SMI Specification

B.1 Compliance Statement

The declaration of SMI-S compliance of a given CIM Instance within a CIM Server also declares that any CIM Instance associated, directly or indirectly, to the first CIM Instance will also be SMIS compliant if SMIS itself declares compliance rules for either CIM Instance or instances of their superclasses.

B.2 How Compliance Is Declared

- The declaration of SMI-S compliance is made through the use of the server profile and the declaration of supported profiles.
- Direct association between CIM Instances is made through instance of a CIM Association.
- Indirect association between CIM Instance is made through more than one CIM Association.
- SMI-S Compliance is assessed against CIM Instances that are directly or indirectly associated to the CIM Instance declared as part of the declaration of supported registered profiles. These CIM Instances comprise the compliance test set.
- All CIM Instances / CIM Classes included in the compliance test set for whom compliance rules are defined in SMI-S or for superclasses thereof shall be themselves be compliant to the rules defined in SMI-S.
- Compliance tests on a superclass of a given CIM Instance are limited to the attributes and behaviors defined for the superclass.

B.3 The Server Profile and Compliance

Compliance is declared by the implementation of the Server Profile. All profiles require the Server profile. The server profile defines the means by which a SMI-S Client determines the profiles and subprofiles supported and the ComputerSystems associated. (see 8.2.4.1, "Server Profile"for more details.)

B.3.1 Example

A CIM Agent for Vendor X declares compliance to the Array Profile and the Pool Manipulation Capabilities, and Setting Subprofile through the Server Profile. Once the association (via the ElementConformsToProfile association) is made to from the Array Profile declaration to the ComputerSystem that realizes the Array Profile, then compliance tests begin testing compliance. Vendor X decided to extend the StorageVolume class with additional properties. StorageVolume is associated to the ComputerSystem via SystemDevice association. ComputerSystem, StorageVolume, and SystemDevice are defined in SMI-S as required CIM elements (see Table 8.2.8.1.9, “CIM Elements” in the “Array Profile”).

In implementing FCPort, Vendor X decided to not provide ElementName but did provide the rest of the required properties. Vendor X decided to not use to WWN and instead used a vendor specific value for the PermanentAddress (see 6.2.4, "Correlatable and Durable Names") Additionally, Vendor X added FRUStatus to their subclass of FCPort. Vendor X also decided to model the back-end fibre channel, but not use an SMI-S model to do so. These back-end FCPorts are associated to the ComputerSystem via the ConsumedSystemDevice association, a subclass of SystemDevice without properties overridden. These back-end fibre channel ports where modeled using a Vendor X specific class, BackendFCPorts, that is not derived from FCPort. This BackendFCPorts were associated to the ComputerSystem with the ConsumedSystemDevice.PartComponent role.

The compliance test includes FCPort because compliance declaration identified a particular ComputerSystem the entry point into compliant CIM instantiation of the Array Profile. the compliance test includes FCPorts as part of the test set because the SystemDevice association, also defined as part of the profile, includes the FCPort realized in that implementation. The compliance test also includes BackendFCPorts because the ConsumedSystemDevice association to the ComputerSystem for these instances is a SystemDevice association.

The compliance test locates the StorageConfigurationService, StoragePools including a Primordial StoragePool, and StorageCapabilities associated to the ComputerSystem. Vendor X's implementation supports the creation of a StoragePool. The test attempts to create a StoragePool given one of the sizes reported by the Primordial StoragePool.getSupportedSizes() method using the Primordial StoragePool reference and a StorageSetting generated from one of the StorageCapabilities.

The compliance test for Vendor X's Array Profile implementation fails because:

- FCPort.PermanentName property has a noncompliance value. Specifically, the FCPort.PermanentAddress is required to be WWN, 16 unseperated uppercase hex digits;
- ElementName property was not provided (i.e., was null);
- the SystemDevice associations contained references to BackendFCPort in the PartComponent property. CIM defined that the PartComponent is a LogicalDevice. Since BackendFCPort is not a LogicalDevice, then the test failed;
- The “Size not supported” return code was returned from CreateOrModifyStoragePool even though one of the supported sizes was used verbatim.

The compliance test for Vendor X's Array Profile implementation did not fail because:

- StorageVolume was extended;
- SystemDevice was extended.

B.4 Backward Compatibility

Backward compatibility between versions of SMI-S profiles is a requirement with very few exceptions. The goals of backwards compatibility include:

a) New profile implementations that are deployed in a customer environment work with existing SMI-S Clients. This includes:

   1) SMI-S operations, including recipes and CTP, continue to work against the new profile implementation;

   2) SMI-S Clients can support a given profile version and above (later minor version numbers);

b) No guarantee of backwards compatibility is implied between major version numbers (i.e., 1.x to 2.x);

c) If a profile in a newer version of SMI-S cannot maintain backward compatibility, it shall be renamed (and the old profile deprecated). Otherwise the client may assume that the newer profile is backwards compatible and that all operations in the earlier version will continue to work in this newer version.

d) It shall be possible for SMI-S provider and client implementations to support older versions of an incompatible profile.

e) Content marked experimental is not standard in this version of the specification. Future versions of the specification may not be backwards compatible to content marked experimental in this version.

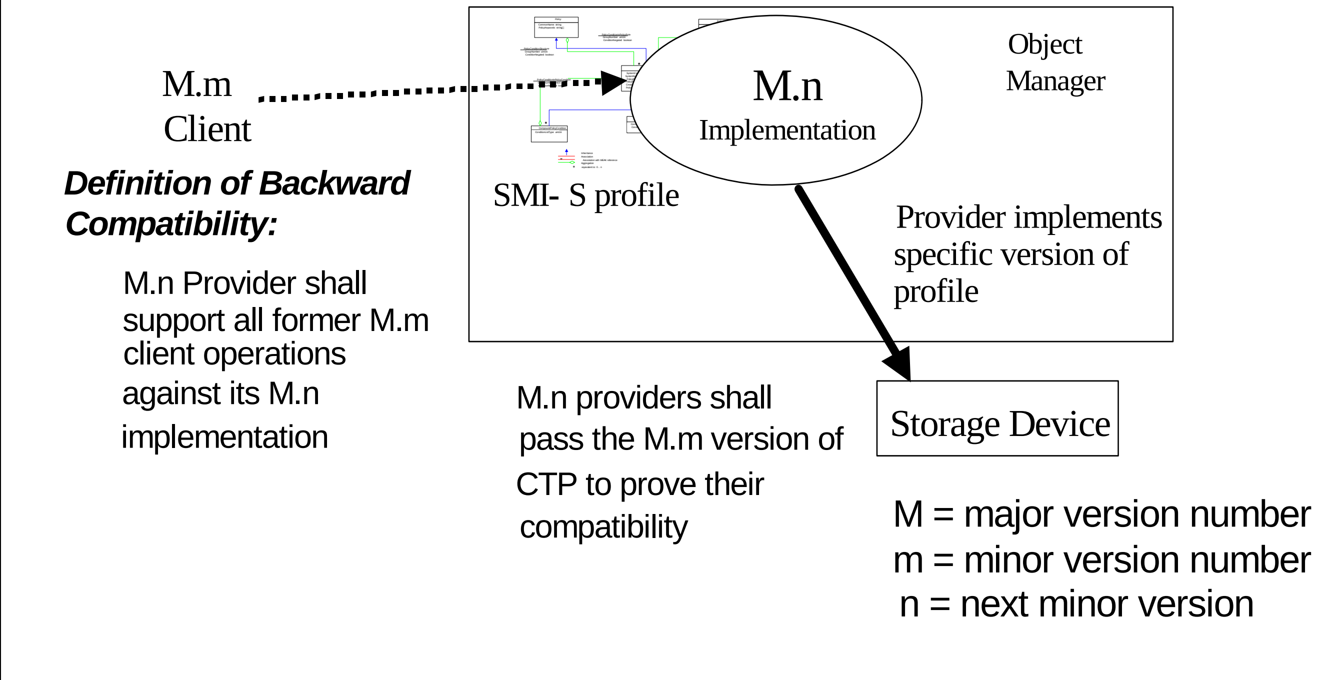
Content marked experimental in this version of the specification may be removed in a future version.

B.4.1 Overview

SMI-S backward compatibility is necessary to ensure that customer environments are minimally disrupted by newer implementations of SMI-S. Deployment of several concurrent implementations of multiple minor versions of SMI-S shall be possible in a customer environment. Compatibility is required from both the Client side and from the provider side. Compatibility also has aspects both in the specification of newer functionality via SMI-S and in the implementation of both providers and clients.

Figure B.1: "Provider Migration" shows the interaction between a Client coded to an older minor version of SMI-S (M.m) acting against a later minor version (M.n) provider implementation

Figure B.1: Provider Migration


M.m
Client
M.n
Implementation
Object
Manager
SMI- S profile
Provider implements specific version of profile
Storage Device
Definition of Backward Compatibility:
M.n Provider shall support all former M.m client operations against its M.n implementation
M.n providers shall pass the M.m version of CTP to prove their compatibility
M = major version number
m = minor version number
n = next minor version

As shown in the diagram, the newer implementation shall support all of the old operations from the previous minor version of SMI-S in order to maintain compatibility. The Client will not be able to take advantage of any newer features that have been added in the later version of the specification, but will still be able to accomplish all of the functions it was coded for in the previous version. This allows minimum disruption to the customer environment.

Clients shall be written to take advantage of the functionality of implementations that are currently shipping and that are or will soon be deployed in customer environments. This client functionality needs to be careful in how it makes use of each SMI-S version's new features. Any client code that uses a specific version's features shall also include a version check against the profile or subprofile version in the RegisteredProfile (Subprofile) instance for that functionality. This version check shall verify that the functionality is at a specific minor version and above (up to the next major release). If a client were only to check for a specific version, it would not be able to use newer implementations of that functionality. A client will, over time, contain multiple such code blocks as newer versions are supported. Each piece of code will be written to the functionality introduced in a specific version and continue to work against that functionality in later minor releases.

B.4.2 Requirements

In order to maintain backwards compatibility with older minor versions of the specification, profile authors have followed specific rules in developing the specification. The requirements that were followed in profile versioning and shall be followed by subsequent implementations include:

Support for required classes: A newer minor version of an SMI-S profile shall support all required classes of the previous minor version of the profile and shall continue to require them.

Deprecation of classes: A newer minor version of an SMI-S profile may deprecate or include deprecated (via the CIM schema) classes introduced in previous minor version(s), but shall continue to require their implementation.

Support for required properties: A newer minor version of an SMI-S profile shall support all required properties of classes in the previous minor version(s) of the profile and shall continue to require them.

Deprecation of properties: A newer minor version of an SMI-S profile may deprecate or include deprecated (via the CIM schema) properties of classes introduced in previous minor version(s), but shall continue to require their implementation.

Support for subprofiles: A newer minor version of an SMI-S profile shall support the functionality of all subprofiles of the previous minor version(s) of the profile and shall continue to require them if they were required in the previous version. A newer minor version of an SMI-S profile may require a subprofile that was optional in the previous minor version, but shall not make optional a subprofile that was required in a previous minor version. If a newer minor version of an SMI-S profile does not have subprofiles by the same name as previous minor version(s), it shall still require implementation of the Registered (Sub)Profile with the previous version information such that the client will be able to find and use the subsumed functionality.

Profile renaming: A newer minor version of an SMI-S profile that cannot remain backwards compatible shall either become a major revision of the profile or shall be renamed to a different profile name such that a client will not find newer, incompatible, versions of that functionality.

B.4.3 Implementation Considerations

Even in the case of a newer minor version of an SMI-S profile that was unable to retain backward compatibility, an implementation may support clients with a separate implementation of the previous minor version's functionality. Implementations shall not implement these earlier versions in such a way that a client of the previous minor version would become confused or break when accessing this functionality. This may happen if the previous version's functionality is implemented in the same namespace as the later version, but a careful evaluation needs to be done by the implementer to determine this. Particular attention should be paid to the recipes from the earlier version, but since recipes are not exhaustive, a fuller evaluation is necessary.