

BLUEFIN SPECIFICATION

Revision 1.0.0

Abstract

This specification documents a secure and reliable interface that allows storage management systems to identify, classify, monitor, and control physical and logical resources in a Storage Area Network.

Partner Development Process (PDP)

1060 El Camino Real, Suite E
Redwood City, CA 94062-1645
Phone (650) 556-9380
Fax (650) 556-9385
www.PartnerDevelopment.org
pdpadmin@PartnerDevelopment.org

PDP Membership

(As of May 2, 2002)

BMC Software, Inc.	Hitachi, Ltd.
Brocade Communication Systems, Inc.	IBM
Compaq Computer Corporation	JNI Corporation
Computer Associates International, Inc.	Prisa Networks
Dell Computer Corporation	Q-Logic Corporation
EMC Corporation	Storage Technology Corporation
Emulex Corporation	Sun Microsystems, Inc.
Gadzoox Networks, Inc.	VERITAS Software Corporation
Hewlett-Packard Company	

Dedication

Dedicated to the memory of Ross Jeynes. His enthusiasm and dedication will be remembered by us all.

DOCUMENT HISTORY

<u>Date</u>	<u>Version</u>	<u>Description</u>
02 May 2002	Revision 1.0.0	Final PDP Specification (Bluefin)

Intended Audience

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

Document Status

This document represents confidential work in progress and should not be distributed or copied without authorization by the Partner Development Process. It is intended to provide a baseline for the construction of the first widely embraced industry standard for SAN management and for eventual submission to the Storage Networking Industry Association or similar standards organizations.

Disclaimer

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

Copyright

Copyright © 2001-2002 Partner Development Process. All rights reserved. All other trademarks or registered trademarks are the property of their respective owners.

Typographical Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [Network Working Group, 1997].

Table of Contents

TABLE OF CONTENTS..... III

TABLE OF TABLES.....XV

TABLE OF FIGURES XVII

CLAUSE 0: INTRODUCTION**21**

 0.1 PREAMBLE 21

 0.2 BUSINESS RATIONAL 21

 0.3 INTERFACE DEFINITION 21

 0.4 TECHNOLOGY TRENDS 24

 0.5 MANAGEMENT ENVIRONMENT 25

 0.6 ARCHITECTURAL OBJECTIVES 26

 0.7 DISCLAIMER 27

CLAUSE 1: BLUEFIN OVERVIEW.....**29**

 1.1 BASE CAPABILITIES 29

 1.1.1 OBJECT ORIENTED 29

 1.1.2 MESSAGING BASED 32

 1.2 CAPABILITIES OF THIS VERSION 32

 1.3 OPERATIONAL ENVIRONMENT 33

 1.4 USING THIS SPECIFICATION 34

 1.5 LANGUAGE BINDINGS 34

CLAUSE 2: TRANSPORT AND REFERENCE MODEL**35**

 2.1 INTRODUCTION 35

 2.1.1 LANGUAGE REQUIREMENTS 35

 2.1.2 COMMUNICATIONS REQUIREMENTS 35

 2.1.3 XML MESSAGE SYNTAX AND SEMANTICS 35

 2.2 TRANSPORT STACK 36

 2.3 REFERENCE MODEL 37

 2.3.1 OVERVIEW 37

 2.3.2 ROLES FOR INTERFACE CONSTITUENTS 37

 2.3.2.1 Client 37

 2.3.2.2 Agent 38

2.3.2.3	Object Manager	38
2.3.2.4	Lock Manager	38
2.3.2.5	Directory Server	38
2.3.3	PEER-TO-PEER ACCESS (CASCADED CLIENTS)	38
3	CLAUSe 3: OBJECT MODEL	39
3.1	MODEL OVERVIEW (KEY RESOURCES)	39
3.1.1	INTRODUCTION TO CIM UML NOTATION	39
3.2	TECHNIQUES	40
3.2.1	CIM FUNDAMENTALS	40
3.2.2	MODELING PROFILES	43
3.2.3	DURABLE NAMES	44
3.2.4	EVENTS – CIM INDICATIONS	46
3.2.4.1	Background	46
3.2.4.2	Using indications	46
3.2.4.3	Indication hierarchy	48
3.2.4.4	Agent/Provider Considerations	49
3.2.4.5	Client Considerations	50
3.2.4.6	Requirements	51
3.2.4.7	Implementation Considerations	51
3.2.5	DEVICE CREDENTIALS	51
3.3	PROFILES	51
3.3.1	PROFILE CONTENT	51
3.3.2	FABRIC	52
3.3.2.1	Fabric Topology	52
3.3.2.2	Switches	56
3.3.2.3	Zoning	59
3.3.2.4	Routers	62
3.3.2.5	Extender	66
3.3.3	HOSTS	69
3.3.3.1	Host Bus Adapters	69
3.3.3.2	Host Discovered Resources	72

3.3.3.3	Management Appliance	79
3.3.4	STORAGE SYSTEMS	82
3.3.4.1	Disk Arrays.....	82
3.3.4.2	Tape Library	106
3.4	CROSS PROFILE CONSIDERATIONS	121
3.4.1	OVERVIEW	121
3.4.1.1	HBA model.....	122
3.4.1.2	Switch Model	122
3.4.1.3	Array Model.....	123
3.4.2	FABRIC TOPOLOGY (HBA, SWITCH, ARRAY)	124
3.4.3	DURABLE IDENTIFIERS	124
3.4.4	STORAGE CONNECTIONS (HBA, ARRAY).....	125
CLAUSE 4:	SECURITY.....	127
4.1	INTRODUCTION	127
4.2	BACKGROUND	127
4.3	MODELING DEVICE CREDENTIALS	128
4.4	REQUIREMENTS.....	128
4.5	AGENT CONSIDERATIONS	129
CLAUSE 5:	SERVICE DISCOVERY.....	131
5.1	DEFINITIONS	131
5.2	OVERVIEW	132
5.3	SLP MESSAGES.....	132
5.3.1	MESSAGE HEADER	132
5.3.2	PROTOCOL EXTENSION BLOCK.....	133
5.3.3	REQUIRED MESSAGES	134
5.3.3.1	Service Request (SrvRqst)	134
5.3.3.2	Service Reply (SrvRply)	134
5.3.3.3	Service Registration (SrvReg)	135
5.3.3.4	Service Acknowledgment (SrvAck)	136
5.3.3.5	Directory Agent Advertisement (DAAdvert)	136
5.3.3.6	Service Agent Advertisement (SAAdvert)	137

5.3.4	OPTIONAL MESSAGES	137
5.3.4.1	Service Deregistration (SrvDereg)	137
5.3.4.2	Service Type Request (SrvTypeRqst).....	138
5.3.4.3	Service Type Reply (SrvTypeRply).....	138
5.3.4.4	Attribute Request (AttrRqst).....	138
5.3.4.5	Attribute Reply (AttrRply)	138
5.4	SCOPES	139
5.4.1	ADMINISTRATIVE SCOPE DISCOVERY	139
5.4.2	DYNAMIC SCOPE DISCOVERY	139
5.5	SERVICES DEFINITION.....	139
5.5.1	SERVICE: URL.....	139
5.5.1.1	Service Types	140
5.5.1.2	Service Access Information.....	140
5.5.1.3	Generic URL Schemes	141
5.5.2	SERVICE TYPE TEMPLATES	141
5.5.2.1	Service Type Template Syntax.....	142
5.5.2.2	Template Description Attributes	142
5.5.2.3	Service Attributes	143
5.6	USER AGENTS (UA).....	145
5.6.1	CONFIGURATION	145
5.6.2	DISCOVERY OF DIRECTORY AGENTS AND SERVICE AGENTS	145
5.6.3	SCOPE.....	146
5.6.4	SERVICE REQUESTS	146
5.6.5	MINIMAL IMPLEMENTATION	147
5.7	SERVICE AGENTS (SAs).....	151
5.7.1	CONFIGURATION	151
5.7.2	DISCOVERY OF DIRECTORY AGENTS	151
5.7.3	SCOPE.....	152
5.7.4	MINIMAL IMPLEMENTATION	152
5.8	DIRECTORY AGENTS (DAs)	153
5.8.1	DIRECTORY AGENT (DA) STATELESS BOOT TIMESTAMP	153

5.8.2	SCOPE.....	153
5.8.3	NETWORK PROTOCOL SPECIFICS	153
5.9	SERVICE AGENT SERVER (SA SERVER)	154
5.9.1	SA SERVER (SAS) IMPLEMENTATION	154
5.9.2	SA SERVER (SAS) CLIENTS	155
5.9.2.1	SAS Client Requests – SA Server Responses.....	155
5.9.3	SA SERVER CONFIGURATION.....	155
5.9.3.1	Overview	155
5.9.3.2	SLP Configuration File.....	155
5.9.3.3	Programmatic Configuration.....	155
5.9.3.4	DHCP Configuration.....	156
5.9.3.5	Scope	156
5.9.4	SA SERVER DISCOVERY	156
5.9.5	SAS CLIENT REGISTRATION	157
5.10	‘BLUEFIN’ SERVICE TYPE TEMPLATES.....	158
5.10.1	‘BLUEFIN’ ABSTRACT SERVICE TEMPLATE.....	158
5.10.2	BLUEFIN’ CONCRETE SERVICE TEMPLATE.....	158
CLAU	SE 6: LOCK MANAGEMENT	161
6.1	INTRODUCTION.....	161
6.2	TERMS	161
6.3	OBJECTIVES.....	161
6.3.1	PROTECTED OPERATIONS.....	161
6.3.2	UNPROTECTED OPERATIONS.....	162
6.3.3	GRANULARITY OF LOCKING.....	162
6.3.4	WHAT IS NOT COVERED	162
6.4	LOCK TYPES	162
6.5	LOCK MANAGER REFERENCE MODEL	163
6.5.1	LOCK MANAGEMENT SERVER OPERATIONS.....	163
6.5.2	LOCK MANAGEMENT AGENT OPERATIONS	165
6.6	DISCOVERY	166
6.7	DEADLOCK MANAGEMENT	166

6.8	LOCK LEASING AND LEASE EXTENSION	166
6.9	LOCK IDENTIFICATION TOKEN CONSIDERATIONS	168
6.10	LOCK MANAGEMENT IMPLEMENTATIONS	169
6.10.1	LOCK UNAWARE CLIENTS	169
6.10.2	LOCK UNAWARE AGENTS/OBJECT MANAGERS	169
6.11	LOCK MANAGEMENT CLIENT – RULES AND RECOMMENDATIONS	169
6.12	LOCK MANAGEMENT SERVER – RULES AND RECOMMENDATIONS	170
6.12.1	STANDARD FEATURES	170
6.12.2	LOCK MANAGER OPTIONAL PROPRIETARY ENHANCEMENTS	170
6.13	PROTOCOL EXTENSIONS – METHODS	171
CLAUSE 7:	BLUEFIN ROLES	173
7.1	INTRODUCTION	173
7.2	CLIENT	174
7.2.1	SLP FUNCTIONS	174
7.2.2	CIM-XML PROTOCOL FUNCTIONS	174
7.2.3	SECURITY CONSIDERATIONS	174
7.2.4	LOCK MANAGEMENT FUNCTIONS	174
7.3	AGENT	174
7.3.1	SLP FUNCTIONS	175
7.3.2	CIM-XML PROTOCOL FUNCTIONS	176
7.3.2.1	Security Considerations	176
7.3.2.2	Required Intrinsic Methods	176
7.3.2.3	Required Model Support	176
7.3.3	LOCK MANAGEMENT FUNCTIONS	177
7.4	OBJECT MANAGER	177
7.4.1	SLP FUNCTIONS	178
7.4.2	CIM-XML PROTOCOL FUNCTIONS	178
7.4.2.1	Security Considerations	178
7.4.2.2	Required Intrinsic Methods	178
7.4.2.3	Required Model Support	179
7.4.3	LOCK MANAGEMENT FUNCTIONS	179

7.4.4	PROVIDER.....	179
7.4.4.1	Required Model Support.....	179
7.5	LOCK MANAGER.....	179
7.5.1	SLP FUNCTIONS.....	179
7.5.2	LOCK MANAGEMENT FUNCTIONS.....	180
7.6	DIRECTORY SERVER.....	180
7.6.1	SLP FUNCTIONS.....	180
7.7	COMBINED ROLES ON A SINGLE SYSTEM.....	180
7.7.1	OBJECT MANAGER AS AN AGENT AGGREGATOR.....	181
7.7.1.1	SLP Functions.....	181
7.7.1.2	CIM-XML Protocol Functions.....	181
7.7.1.3	Security Considerations.....	181
7.7.1.4	Lock Manager Functions.....	181
8	INSTALLATION AND UPGRADE.....	183
8.1	INTRODUCTION.....	183
8.2	ROLE OF THE ADMINISTRATOR.....	183
8.3	GOALS.....	183
8.3.1	NON-DISRUPTIVE INSTALLATION AND DE-INSTALLATION.....	183
8.3.2	PLUG-AND-PLAY.....	184
8.4	INSTALLING DEVICE SUPPORT.....	184
8.4.1	INSTALLATION.....	184
8.4.2	DISCOVERY AND INITIALIZATION OF DEVICE SUPPORT.....	185
8.4.3	REMOVAL/UPDATE.....	185
8.4.4	RECONFIGURATION.....	186
8.4.5	FAILURE.....	186
8.5	OBJECT MANAGER.....	186
8.5.1	INSTALLATION.....	186
8.5.2	REMOVAL/UPGRADE.....	187
8.5.3	RECONFIGURATION.....	187
8.5.4	FAILURE.....	187
8.6	CLIENT.....	187

8.6.1 REMOVAL187

8.6.2 RECONFIGURATION187

8.6.3 FAILURE187

8.7 LOCK MANAGER187

8.7.1 INSTALLATION187

8.7.2 REMOVAL187

8.7.3 RECONFIGURATION188

8.8 DIRECTORY SERVER188

8.8.1 INSTALLATION188

8.8.2 REMOVAL/FAILURE188

8.9 MANAGEMENT DOMAINS188

8.9.1 INITIAL CONFIGURATION188

8.9.2 RECONFIGURATION188

APPENDIX A: GLOSSARY..... 189

INTRODUCTION.....189

NEW ADDITIONS/MODIFICATIONS:189

A.....190

B.....191

C.....191

D193

E.....195

F.....195

G196

H196

I.....197

J197

K198

L.....198

M.....199

N200

O201

P 202

Q 203

R 203

S 204

T 207

U 208

V 209

W 210

X 211

Y 211

Z 211

APPENDIX B: BIBLIOGRAPHY..... 213

APPENDIX C: DETAILED CLASS DERIVATIONS..... 214

C.1 ACTIVECONNECTION 214

C.2 ADMINDOMAIN 214

C.3 ALERTINDICATION..... 215

C.4 ALLOCATEDFROMSTORAGEPOOL..... 215

C.5 ASSOCIATEDSTORAGECONFIGURATIONJOB 215

C.6 BASEDON 216

C.7 CHANGERDEVICE 216

C.8 CHASSIS..... 217

C.9 COMPONENT 217

C.10 COMPONENTCS 217

C.11 COMPUTERSYSTEM..... 218

C.12 COMPUTERSYSTEMPACKAGE..... 218

C.13 CONCRETEIDENTITY 219

C.14 CONFIGURATION 219

C.15 CONFIGURATIONCAPACITY 220

C.16 CONTROLLEDBy 220

C.17 CONTROLLER 221

C.18 DEPENDENCY..... 221

C.19	DEPENDENCYCONTEXT	222
C.20	DEVICESAPIIMPLEMENTATION	222
C.21	DEVICESERVICESLOCATION	223
C.22	DEVICESOFTWARE	223
C.23	DISKDRIVE	224
C.24	ELEMENTCAPABILITIES	224
C.25	ELEMENTCAPACITY	225
C.26	ELEMENTCONFIGURATION	225
C.27	ELEMENTSETTING	225
C.28	ELEMENTSTATISTICS	225
C.29	EXECUTINGSTORAGECONFIGURATIONJOB.....	226
C.30	EXTENTREDUNDANCYCOMPONENT	226
C.31	EXTRACAPACITYGROUP.....	226
C.32	FCPORT	227
C.33	FCPORTSTATISTICS.....	228
C.34	FRU	229
C.35	FORWARDINGSERVICE	230
C.36	FORWARDSAMONG.....	231
C.37	HOSTEDACCESSPOINT.....	231
C.38	HOSTEDCOLLECTION	231
C.39	HOSTEDSERVICE.....	232
C.40	HOSTEDSTORAGEPOOL.....	232
C.41	INDICATIONFILTER	232
C.42	INDICATIONHANDLER.....	233
C.43	INDICATIONHANDLERCIM-XML.....	233
C.44	INDICATIONSUBSCRIPTION	234
C.45	INSTALLEDSOFTWAREELEMENT.....	234
C.46	INSTCREATION.....	235
C.47	INSTDELETION.....	235
C.48	INSTMODIFICATION	235
C.49	INTERLIBRARYPORT	236

C.50	LIBRARYEXCHANGE.....	236
C.51	LIBRARYPACKAGE.....	237
C.52	LIMITEDACCESSPORT.....	237
C.53	LOGICALDEVICE	237
C.54	LOGICALMODULE	239
C.55	LOGICALNETWORK	239
C.56	LOGICALPORTGROUP	239
C.57	MEMBEROFCOLLECTION	240
C.58	MODULEPORT.....	240
C.59	OBJECTMANAGER.....	241
C.60	PACKGEDCOMPONENT.....	241
C.61	PHYSICALCONNECTOR.....	242
C.62	PHYSICALMEDIA.....	242
C.63	PHYSICALMEDIAINLOCATION	243
C.64	PHYSICALPACKAGE.....	244
C.65	PHYSICALTAPE	244
C.66	PORTIMPLEMENTSENDPOINT	245
C.67	PRODUCT	245
C.68	PRODUCTPHYSICALELEMENTS.....	246
C.69	PROTOCOLENDPOINT.....	246
C.70	PROVIDESSERVICETOELEMENT	247
C.71	REALIZES	247
C.72	REDUNDANCYCOMPONENT.....	247
C.73	REDUNDANCYGROUP	248
C.74	REMOTESERVICEACCESSPOINT.....	248
C.75	SCSICONTROLLER.....	249
C.76	SCSIINTERFACE	250
C.77	SCSILUN.....	251
C.78	SERVICE	252
C.79	SERVICEACCESSBYSAP	252
C.80	SETTING.....	253

C.81	SETTINGCONTEXT.....	255
C.82	SHAREDSECRETSERVICE	256
C.83	SHAREDSECRET	256
C.84	SOFTWAREELEMENT.....	257
C.85	SPAREGROUP	257
C.86	STORAGEACCESSSERVICE	258
C.87	STORAGECAPABILITIES.....	264
C.88	STORAGECONFIGURATIONJOB.....	266
C.89	STORAGECONFIGURATIONSERVICE	267
C.90	STORAGEEXTENT	273
C.91	STORAGEMEDIALOCATION	275
C.92	STORAGELIBRARY	276
C.93	STORAGEPOOL.....	277
C.94	STORAGEPOOLCOMPONENT	279
C.95	STORAGEREDUNDANCYGROUP.....	279
C.96	STORAGESETTING	280
C.97	STORAGESETTINGWITHHINTS	283
C.98	STORAGEVOLUME.....	287
C.99	SYSTEMDEVICE.....	288
C.100	TAPEDRIVE	289
C.101	UNITACCESS.....	289
C.102	ZONE.....	289
C.103	ZONEALIAS	290
C.104	ZONECAPABILITIES.....	290
C.105	ZONEMEMBER	291
C.106	ZONESERVICE	291
C.107	ZONESET.....	292
APPENDIX D: FUTURES.....		293
D.1	HBA LUN MASKING AND PERSISTENT BINDING	293
D.2	MANAGED HUB SECTION.....	293
D.3	IPSEC	293

D.4	MULTI-PATH MODELING	293
D.5	PROVIDER MODELING	293
D.6	REQUIREMENTS HIGHLIGHTING.....	293
D.7	NON-FIBRE FABRICS	293
D.8	COMPLIANCE NOTIFICATION	294
D.9	CASCADED AGENTS	294
D.10	DURABLE ID FORMATS.....	303

Table of Tables

Table 1: Profile Components	52
Table 2: Switch Required Classes	59
Table 3: Zoning Required Classes	62
Table 4: Router Required Classes	65
Table 5: Required Classes for HBA.....	72
Table 6: SCSI Device Type Mapping.....	77
Table 7: Required Classes for Management Appliance	82
Table 8: LogicalDevice Durable Names	88
Table 9: Required Classes for Disk Arrays.....	106
Table 10: Required Classes for Tape.....	121
Table 11: Cross Profile Durable Identifiers.....	125
Table 12: Message Types	132
Table 13: SLP v2 Status Codes	133
Table 14: ActiveConnection Association Derivation	214
Table 15: AdminDomain Derivation	215
Table 16: AllocatedFromStoragePool Derivation.....	215
Table 17: AssociatedStorageConfigurationJob Association Derivation.....	216
Table 18: BasedOn Derivation	216
Table 19: ChangerDevice Derivation	217
Table 20: Chassis Derivation.....	217
Table 21: Component	217
Table 22: ComponentCS Aggregation Derivation	218
Table 23: ComputerSystem Derivation	218
Table 24: ComputerSystemPackage Derivation.....	219
Table 25: ConcreteIdentity Derivation	219
Table 26: Configuration Class Derivation	219
Table 27: ConfigurationCapacity Derivation	220
Table 28: ControlledBy Derivation	221
Table 29: Controller Derivation	221
Table 30: Dependency Derivation	222
Table 31: DependencyContext Derivation	222
Table 32: DeviceServicesLocation Derivation	223
Table 33: DeviceSoftware Derivation	223
Table 34: DiskDrive Derivation	224
Table 35: ElementCapabilities Derivation	224

Table 36: ElementCapacity Derivation.....	225
Table 37: ElementConfiguration Association Derivation.....	225
Table 38: ElementSetting Association Derivation.....	225
Table 39: DeviceStatistics Derivation.....	226
Table 40: ExecutingStorageConfigurationJob Derivation.....	226
Table 41: ExtentRedundancyComponent Derivation.....	226
Table 42: ExtraCapacityGroup Derivation.....	227
Table 43: FCPort Derivation.....	228
Table 44: FCPortStatistics Derivation.....	229
Table 45: FRU Derivation.....	230
Table 46: ForwardingService Derivation.....	231
Table 47: ForwardAmong Association Derivation.....	231
Table 48: HostedAccessPoint Derivation.....	231
Table 49: HostedCollection Inheritance.....	231
Table 50: HostedService Derivation.....	232
Table 51: HostedStoragePool Derivation.....	232
Table 52: IndicationFilter Derivation.....	233
Table 53: IndicationHandler Derivation.....	233
Table 54: IndicationHandlerCIM-XML Derivation.....	234
Table 55: IndicationSubscription Association Derivation.....	234
Table 56: InstalledSoftwareElement Derivation.....	235
Table 57: InstCreation Derivation.....	235
Table 58: InstDeletion Derivation.....	235
Table 59: InstModification Derivation.....	236
Table 60: InterLibraryPort Derivation.....	236
Table 61: LibraryExchange Derivation.....	236
Table 62: LibraryPackage Derivation.....	237
Table 63: LimitedAccessPort Derivation.....	237
Table 64: LogicalDevice Derivation.....	238
Table 65: LogicalModule Derivation.....	239
Table 66: LogicalNetwork Derivation.....	239
Table 67: LogicalPortGroup Derivation.....	240
Table 68: MemberOfCollection Inheritance.....	240
Table 69: ModulePort Derivation.....	240
Table 70: ObjectManager Derivation.....	241
Table 71: PhysicalConnector Derivation.....	242
Table 72: PhysicalMedia Derivation.....	243
Table 73: PhysicalPackage Derivation.....	244
Table 74: <i>PortImplementsEndpoint</i> Association Derivation.....	245
Table 75: Product Derivation.....	245
Table 76: ProductPhysicalElements Derivation.....	246
Table 77: ProtocolEndpoint Derivation.....	246
Table 78: ProvidesServiceToElements Derivation.....	247
Table 79: Realizes Derivation.....	247
Table 80: RedundancyComponent Derivation.....	247
Table 81: RedundancyGroup Derivation.....	248
Table 82: RemoteServiceAccessPoint.....	249
Table 83: SCSIController Derivation, General Case.....	250
Table 84: SCSIController Derivation, Alternate Case.....	250
Table 85: SCSIInterface Derivation, General Case.....	251
Table 86: SCSELUN Derivation.....	251
Table 87: Service Derivation.....	252
Table 88: ServiceAccessBySAP Derivation.....	252
Table 89: Setting Derivation.....	255

Table 90: SettingContext Aggregation Derivation.....	256
Table 91: SharedSecretService Derivation.....	256
Table 92: SharedSecret Derivation.....	257
Table 93: SoftwareElement Derivation.....	257
Table 94: SpareGroup Derivation.....	258
Table 95: StorageAccessService Derivation.....	264
Table 96: StorageAccessService Alternate Derivation.....	264
Table 97: StorageCapabilities Derivation.....	266
Table 98: StorageConfigurationJob Class Derivation.....	267
Table 99: StorageConfigurationService Derivation.....	273
Table 100: StorageExtent Derivation.....	275
Table 101: StorageMediaLocation Derivation.....	276
Table 102: StorageLibrary Derivation.....	277
Table 103: StoragePool Class Derivation.....	279
Table 104: StoragePoolComponent Derivation.....	279
Table 105: StorageRedundancyGroup Derivation.....	280
Table 106: StorageSetting Class Derivation.....	283
Table 107: StorageSettingWithHints Derivation.....	287
Table 108: StorageVolume Derivation.....	288
Table 109: SystemDevice Derivation.....	289
Table 110: TapeDrive Derivation.....	289
Table 111: UnitAccess Association Derivation.....	289
Table 112: Zone Derivation.....	290
Table 113: ZoneAlias Derivation.....	290
Table 114: ZoneCapabilities Derivation.....	291
Table 115: ZoneMember Derivation.....	291
Table 116: ZoneService Derivation.....	292
Table 117: ZoneSet Derivation.....	292

Table of Figures

Figure 1: Interface Function.....	22
Figure 2: Large SAN Topology.....	25
Figure 3: Example Client Server Distribution in a SAN.....	26
Figure 4: Bluefin Modeling Conventions.....	29
Figure 5: Object Model/Server Relationship.....	30
Figure 6: Canonical Inheritance.....	31
Figure 7: Sample CIM-XML Message.....	32
Figure 8: Operational Environment.....	33
Figure 9:Transport Stack.....	36
Figure 10: Reference Model.....	37
Figure 11: Cluster Model.....	42
Figure 12: Common Elements.....	43
Figure 13: WBEMService Hierarchy.....	44
Figure 14 - Indications Filters Schema.....	47
Figure 15 - Indications Schema.....	48
Figure 16: Fabric Schema.....	54
Figure 17: Fabric Instance Diagram.....	55
Figure 18: Fabric Required Classes.....	56
Figure 19: Switch Schema Diagram.....	57
Figure 20: Switch Instance Diagram.....	57
Figure 21: Zoning Schema.....	60

Figure 22: Zoning Instance Diagram	60
Figure 23: Router Schema Diagram	63
Figure 24: Router Instance Diagram	64
Figure 25: Extender Schema Diagram	66
Figure 26: Extender Instance Diagram	67
Figure 27: Required Classes for Extender	69
Figure 28: HBA Schema Diagram	70
Figure 29: HBA Instance Diagram	71
Figure 30: HBA Binding Instance Diagram	71
Figure 31 Host Discovered Resources Schema Diagram	74
Figure 32 Host Discovered Objects Instance Diagram	75
Figure 33 Host Discovered Objects Instance Diagram	75
Figure 34: Management Appliance Schema Diagram	79
Figure 35: Management Appliance Instance Diagram	80
Figure 36: Disk Array Core Schema Diagram	83
Figure 37: Disk Array Service Schema Diagram	83
Figure 38: Disk Array Instance Diagram	84
Figure 39: JBOD Array Model	85
Figure 40: Single/Dual Processor Models	86
Figure 41: Simple Disk Model	87
Figure 42: Raid Group Model	89
Figure 43: Virtualization Across Multiple Systems	90
Figure 44: Batch LUN Masking Objects	92
Figure 45: Storage Configuration	94
Figure 46: Storage Pool Example	96
Figure 47: Storage Configuration	97
Figure 48: Physical Disk Model	98
Figure 49: Array Internal Connections	99
Figure 50: Spare Disk	99
Figure 51: Storage Redundancy Model	100
Figure 52: JBOD Model	101
Figure 53: Asymmetric Virtualization Appliance	102
Figure 54: Symmetric Virtualization Appliance	102
Figure 55: Symmetric Virtualization Appliance	108
Figure 56: Storage Media Library Device View	110
Figure 57: Storage Media Library Schema: Physical View	113
Figure 58: Storage Media Library Schema: Software/Service View	114
Figure 59: Tape Library Instance Diagram	115
Figure 60: System Diagram	122
Figure 61: Host Bus Adapter Model	122
Figure 62: Switch Model	123
Figure 63: Array Instance	124
Figure 64 - Device Credentials	128
Figure 65: Directory Agent (DA) Discovery – Active and Passive	146
Figure 66: Service Agent Discovery using a Directory Agent	148
Figure 67: Service Agent Discovery without a Directory Agent	149
Figure 68: Service Agent Discovery Using a Directory Agent and Object Manager	150
Figure 69: Service Agent Registration with a Directory Server	152
Figure 70: SA Server Configuration	156
Figure 71: Lock Management Reference Model	163
Figure 72: Lock Request Success Sequence Diagram	164
Figure 73: Unsuccessful Lock Request Sequence Diagram	165
Figure 74: Lock Lease Renewal Success	167
Figure 75: Lock Lease Renewal Failure	168

Figure 76: Complete Reference Model 173

Figure 77: Interop Schema Object Model 177

Figure 78: Configuration Administration..... 185

Clause 0: Introduction

0.1 Preamble

Storage Area Networks (SANs) are emerging as a prominent layer of IT infrastructure in enterprise class computing environments. Applications and functions driving the emergence of SAN technology include:

- 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.
- Sharing of vast storage resources between multiple systems.

To accelerate the emergence of SANs in the market, the industry requires a standard management interface that will allow different classes of hardware and software products supplied by multiple vendors to reliably and seamlessly interoperate for the purpose of monitoring and controlling resources. It is the goal of this interface to provide for the functionally rich, reliable, and secure monitoring/control of mission critical global resources in complex and potentially broadly distributed multi-vendor SAN topologies. As such, this interface overcomes the deficiencies associated with legacy management interfaces principally developed for the networking industry or the "pre-SAN" storage industry.

0.2 Business Rational

The business goal is to provide an "open" and extensible interface that will allow subsystems and devices within the global context of a SAN 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) shall, via a standard SAN management interface, be able to more flexibly select between multiple vendors when building the hierarchy of software systems required to manage a large SAN independent of the underlying hardware systems. Additionally, SAN integrators shall be able to more flexibly select between alternate hardware vendors when constructing SAN configurations. As such, this interface is targeted at creating: broad multi-vendor management interoperability and thus, increasing customer satisfaction. Increased customer satisfaction will:

- More rapidly expand the acceptance of SANs.
- Accelerate customer acquisition of SAN technology.
- Expand the total market.

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

0.3 Interface Definition

This management interface allows storage management systems (either user written or formal products) to reliably identify, classify, monitor, and control physical and logical resources in a SAN. 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.

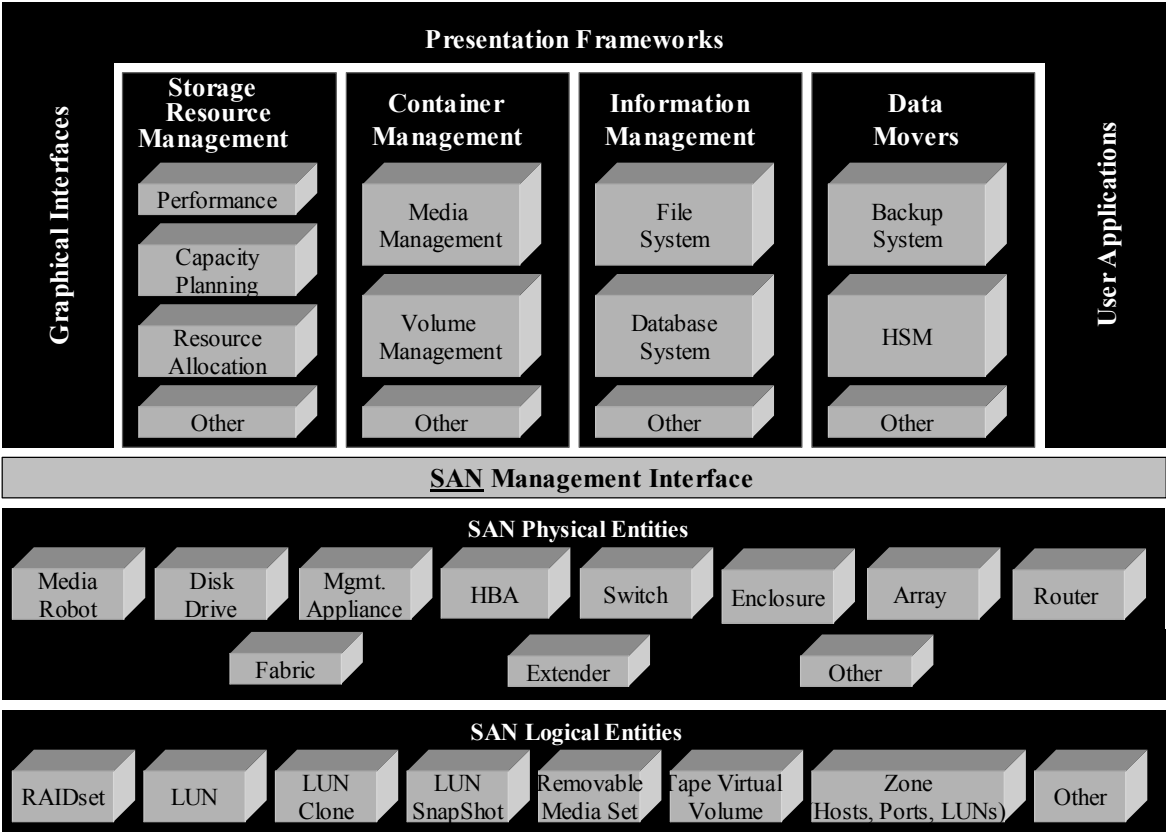


Figure 1: Interface Function

The diagram illustrates that functions of the interface may be distributed across multiple SAN devices (i.e., Switches or Array Controllers) and/or software systems (i.e., Discovery Systems). While the functionality of the interface is distributed within a SAN, to insure that monitoring and control operations by clients are consistent and reliable, the state of a given resource should not be 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 satisfied by a single monitoring entity to insure information consistency. Should the SRM application and Backup engine obtain different available bandwidth information for a given Fibre Channel path from multiple unsynchronized sources they may function in conflict and degrade the efficiency of the environment.

Satisfying this requirement is the responsibility of parties configuring SAN management clients in conjunction with the primitives defined in the specification.

It should also be noted that within this architecture (as depicted by the illustration above) entities like an appliance based volume manager are potentially both a client and a server to the interface.

Example: a host based volume manager may desire 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 must 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) must 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.

Relative to Figure 1, examples of long-term functional goals for clients using this interface include:

1. The need for a graphical management console to visualize the resources in a SAN as well as the topology of those resources.
2. The need for a management console to identify a resource that has experienced an error/fault condition that has degraded/disabled its operation.
3. The need for a management console to construct a zone of allocation around a select group of host and storage resources.
4. The desire for a volume manager to inspect the nonvolatile storage resources available to it such that it may construct a storage pool of a consistent level of performance and availability.
5. The desire that a server-free backup engine be able to identify the 3rd party copy engines (and associated media libraries/robots) available, and allocate an engine/library/robot to a given backup task.
6. The need for a file system to extend its capacity through dynamically utilizing additional non-volatile storage volumes. Note: each volume to be utilized must meet strict availability and performance requirements and thus, the file system must inspect the properties of each volume prior to allocation.
7. The desire for a Storage Resource Management (SRM) application like a SAN performance monitor to identify topology and line utilization such that performance bottlenecks may be exposed.
8. The requirement that a capacity planning system identify each storage pool in the SAN and then interact with the manager of each pool to assess utilization statistics.
9. The need 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.
10. The requirement that fault isolation and analysis systems asynchronously receive events relative to the health and performance of the devices and subsystems in the SAN.

Example non-goals for this interface include:

- Select a logical communications port over which to send/receive data.
- Read/Write data to a volume.
- Identify and recover from data communications errors and failures.
- Synchronization message between two cluster nodes.
- Log a new communications device into a network.

0.4 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:

1. **Improved Connectivity:** Whether available In-band (i.e., over Fibre Channel) or available out-of-band (i.e., over a LAN/MAN/WAN), or available over a mix of both, virtually all devices in a SAN have (or soon will have), access to a common communications transport suitable for carrying management information content (namely TCP/IP).
2. **Increased Device Capability:** All SAN devices (even simple ones like a switch) have sufficient capability to communicate via a common, general-purpose network transport (again, TCP/IP) or, using proxy services through another resource (e.g. general purpose computer system), to gain access to a common communications transport.

Example 1: 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 2: 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.

3. **XML Standardization:** XML is providing management protocols with an extensible, platform independent, human readable, content describable communication language for the first time. 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.
4. **Increased SAN Complexity:** SANs being configured with diverse classes of components and widely distributed topologies. Management clients and servers in the environment being widely distributed on systems, appliances, and devices throughout large SAN topologies while maintaining real-time distributed state for logical entities. Figure 2 below provides an example of a single SAN of multiple classes of components spanning three physical locations (i.e., Sites A, B and C). In this figure a communications switch is denoted by label “SW”, a host denoted by “H”, and a storage array denoted by “A”.

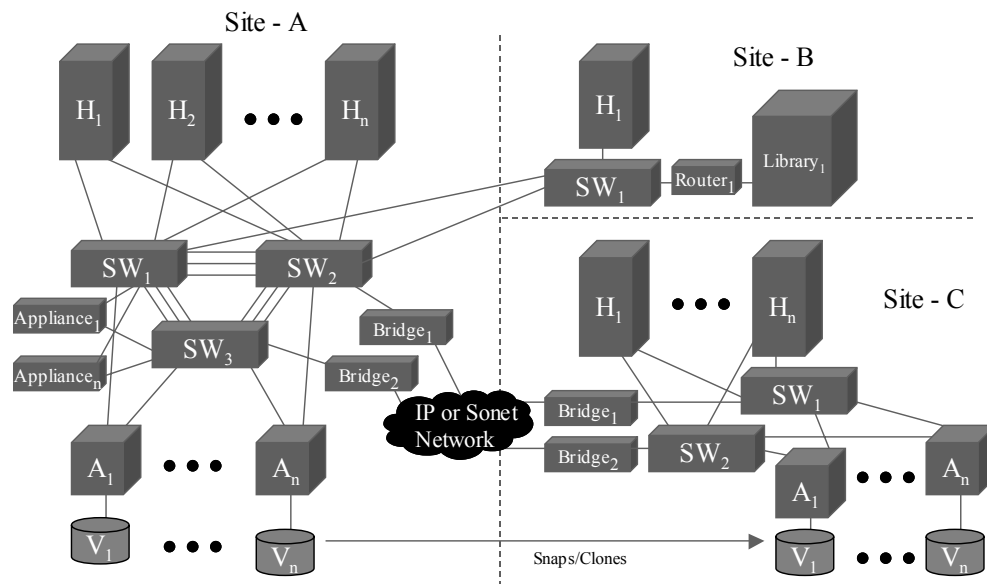


Figure 2: Large SAN Topology

0.5 Management Environment

Clients and Servers of this interface will be widely distributed on systems, appliances, and devices throughout large SAN topologies. The configuration in Figure 3 provides an example client/server distribution using in-band TCP/IP communications, out of band TCP/IP communication, or employing proxy services to bridge legacy and/or proprietary communication interfaces.

In Figure 3, the device “Old Array Controller” is incapable of either in-band or out-of-band communications with clients and servers in the management environment. 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.

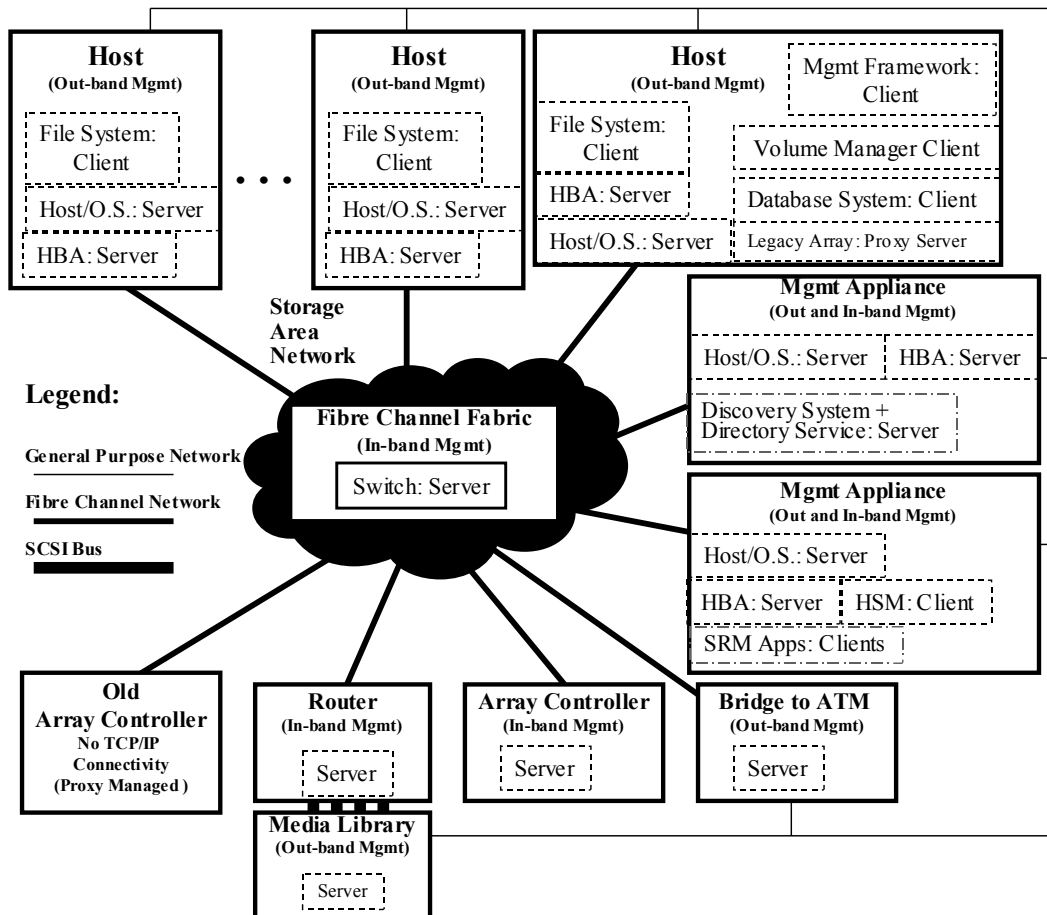


Figure 3: Example Client Server Distribution in a SAN

0.6 Architectural Objectives

The following reflect architectural objectives of the interface. Some of these capabilities may not be present in early releases of the interface but are inherent in its architecture to insure extensibility and thus, eventually broad adoption.

1. **Consistency:** State within an object and between objects shall be consistent independent of the number clients simultaneously exerting control, the distribution of objects in the environment, or the management action being performed.
2. **Isolation:** A client that must execute an atomic set of management actions against one or more objects shall do so in isolation of other clients who may desire to simultaneously execute management actions against those same objects.
3. **Durability:** Atomicity, consistency, and isolation shall be preserved independent of the failure of any entity or communications path in the management environment.
4. **Consistent Name Space:** Managed objects in the SAN must adhere to a consistent naming convention independent of state or reliability of any object, device, or subsystem in the SAN.
5. **Distributed Security:** Monitoring and control operations shall be secure. In specific, the architecture shall support:

- Client authentication.
 - Privacy (encryption) of the content of the messages in this protocol.
 - Client authorization (by object class).
6. Physical Interconnect Independence: The interface shall function independent of any particular SAN physical interconnect, supplier, or topology.
 7. 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.
 8. Scalability: The size, physical distribution, or heterogeneity of the SAN shall not degrade the quality or function of the management this interface.
 9. Vendor Unique Extension: The interface shall allow vendors to implement proprietary functionality to distinguish their products and services in the market independent of the release of a new version of the interface.
 10. Volatility of State: This interface shall not assume that objects are preserved in non-volatile repositories. Clients and servers may or may not preserve object state across failures.
 11. Replication: This interface provides no support for the automatic replication of object state within the management environment.
 12. Functional Layering Independence: The design of this interface is independent of any functional layering a vendor may choose to employ in constructing the storage management systems (hardware and software) necessary to manage a SAN.
 13. Asynchronous or Synchronous execution. Management actions may execute either asynchronously or synchronously. In synchronous management actions, a client shall not perform other work until a response is received indicating success or failure of the management action. In asynchronous management actions a client may perform other work in the presence of a service executing the management action. Asynchronous management actions will be signal as complete or failed with the delivery of an event to the client.
 14. Events: Provide for the reliable asynchronous delivery of events to one or more registered clients.
 15. Cancelable Management Actions. Long running synchronous or asynchronous directives shall be capable of being cancelled by the client. Cancellation results in the termination of work by the server and resource consumed being returned.
 16. Durable Reference: Object classes that persist across power cycles and must be monitored and controlled independent of SAN reconfiguration (i.e., logical volumes) shall be identified via “Durable Names” to insure consistent reference by clients.
 17. Dynamic installation and reconfiguration: New clients and servers shall be capable of being added to or removed from a Bluefin 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 Bluefin environment.

0.7 Disclaimer

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

Clause 1: Bluefin Overview

1.1 Base Capabilities

To achieve the architectural objectives and support the key technological trends in Clause 0, Bluefin is an object-oriented, XML messaging based interface designed to support the specific requirements of Storage Area Networks. To quickly become ubiquitous, Bluefin seeks to the greatest extent possible to leverage existing enterprise management standards like:

- 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 standards emerging from the Storage Networking Industry Association (SNIA) on volume and array management.

1.1.1 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 define most of the capabilities of the Bluefin. The illustration below provides a simple example of UML using CIM classes for reference. Implementers of this specification are encouraged to consult one of the many publicly available texts on UML or the DMTF web site (www.DTMF.org) to develop an understanding of UML. A brief tutorial on UML is provided in the introduction material on the Clause on Object Model in this specification.

Each Bluefin server in a SAN provides one or more object classes (and related instances) to clients for monitoring and control per Figure 5.

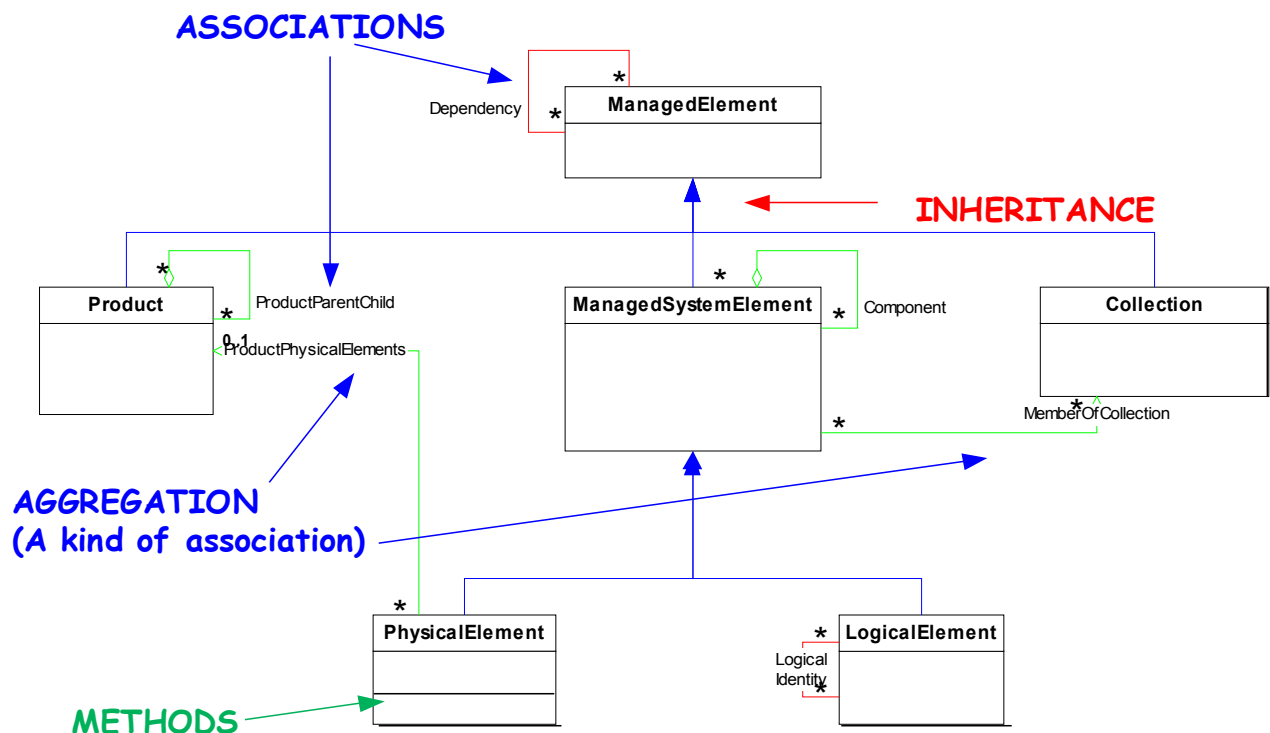


Figure 4: Bluefin Modeling Conventions

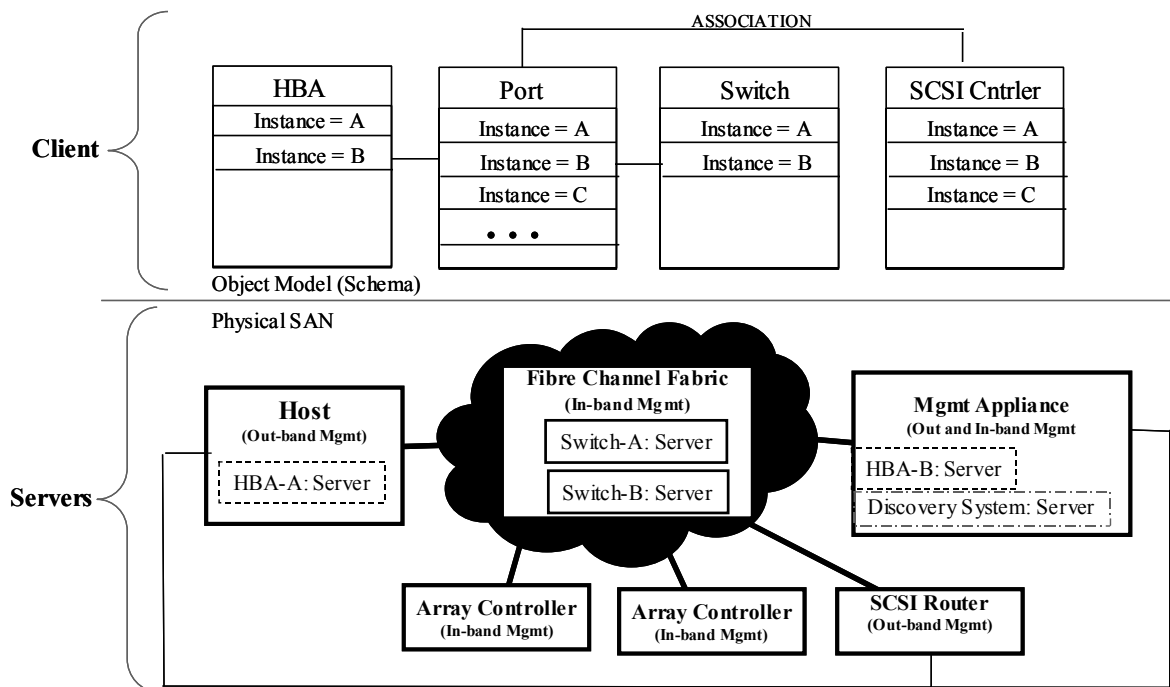


Figure 5: Object Model/Server Relationship

In Figure 5, a Bluefin client obtains the classes of objects as well as the instances that populate those objects from distributed servers in the SAN. In this example, the Switch server identifies its object classes (Switch and Port) and also provides instances of those object classes. The HBA servers populate classes and related instances, and the Array Controllers and Router populates SCSIController classes/instances. After a Bluefin client has populated its schema with object classes and instances it may proceed with monitoring and controlling the resources of a SAN.

Having an object oriented interface Bluefin clients are capable of discovering, monitoring, and controlling a SAN, independent of the precise definition of the object model that defines that SAN. In specific, the code written to discover object classes, enumerate instances of those classes, traverse associations between classes, as well as read/ set properties does not require modification as the object model for that SAN evolves. Additionally, the underlying message/transfer syntax used to communicate between Bluefin clients and servers is also object model independent.

The object model in this specification is expressed in UML diagrams, easy-to-use tables and machine-readable CIM compliant Managed Object Format (MOF) format (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 Object Managers (Called CIM Object Managers or CIMOMs) available through various open-source communities. 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. Thus, programmers implementing this interface should reference MOF representations of the object model when faced with implementation decisions.

The constituents (clients and servers) of a Bluefin management environment do not require identical copies of the Bluefin object model to operate reliably or interoperate. Each client and server may uniquely leverage vendor unique extensions as well as experience different levels of functionality associated with the rolling upgrade of capabilities. However, for clients and servers to interoperate, each server must provide the complete tree of object classes that it sub-classes from. All Bluefin object classes must ultimately root from a single canonical object.

Figure 6 illustrates this requirement.

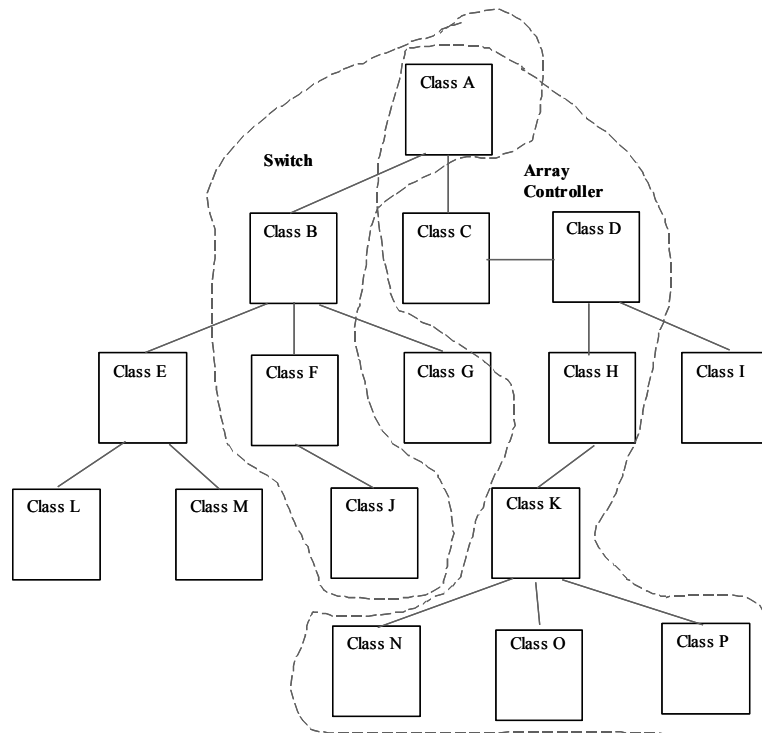


Figure 6: Canonical Inheritance

Figure 6 illustrates that even though a Fibre Channel Switch may only report instances and allow associated method execution for Object Classes F and J, it shall, when asked by a client to enumerate its Object Classes report the entire hierarchy of classes in its tree. Similarly a server that instantiates an array controller must report the complete set of object classes that links it to the base canonical object (object A) of the Bluefin model. It is this single canonical root that allows any Bluefin 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 but not limit the expression of unique features that may differentiate manufacturers in the market. For this reason, the object model herein 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 in select cases (extrinsic methods) may require the modification of client code above and beyond that required to support the core.

1.1.2 Messaging Based

A messaging based versus traditional procedure call interface “style” was selected so that platform and language independence could be achieved across the breadth of devices, clients, and manufacturers that will implement the interface. This messaging based environment also eases the task of transporting management actions over different communications transports and protocols as the computer industry evolves. While a messaging based interface provides these advantages, the implementation of Bluefin clients and servers will require the marshalling and un-marshalling of messages into procedure call semantics such that programmatic environments may operate against the object model exported through this messaging interface.

An example Bluefin CIM-XML message is provided below for familiarity.

```
<?XML Version="1.0"?>
<!DOCTYPE CIM SYSTEM http://www.dmtf.org/cim-v2.dtd>
<CIM VERSION="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>
        <VALUE>Degraded</VALUE>
        <VALUE>Unknown</VALUE>
      </QUALIFIER>
    </PROPERTY>
  </CLASS>
</CIM>

<?XML Version="1.0"?>
<!DOCTYPE CIM SYSTEM http://www.dmtf.org/cim-v2.dtd>
```

Figure 7: Sample CIM-XML Message

1.2 Capabilities Of This Version

Functional capabilities of the interface as described in this version of the specification include:

1. Allow a client to identify key resources in a SAN (e.g. HBA, Array Controller, Switch)
2. Allow a client to identify interconnects between key resources in a SAN.
3. Allow a client to receive asynchronous notification that the configuration of a SAN has changed.
4. Allow a client to identify the health of key resources in a SAN.
5. Allow a client to identify the available performance of interconnects in a SAN.
6. Allow a client to receive asynchronous notification that the health of a SAN resource has degraded.
7. Allow a client to receive asynchronous notification that the performance of a SAN interconnect has degraded.

8. Allow a client to identify the zones being enforced in a SAN.
9. Allow a client to create/delete and enable/disable zones in a SAN.
10. Allow a client to identify the LUN masks/maps in a SAN.
11. Allow a client to create/delete and enable/disable LUN masks/maps in a SAN.

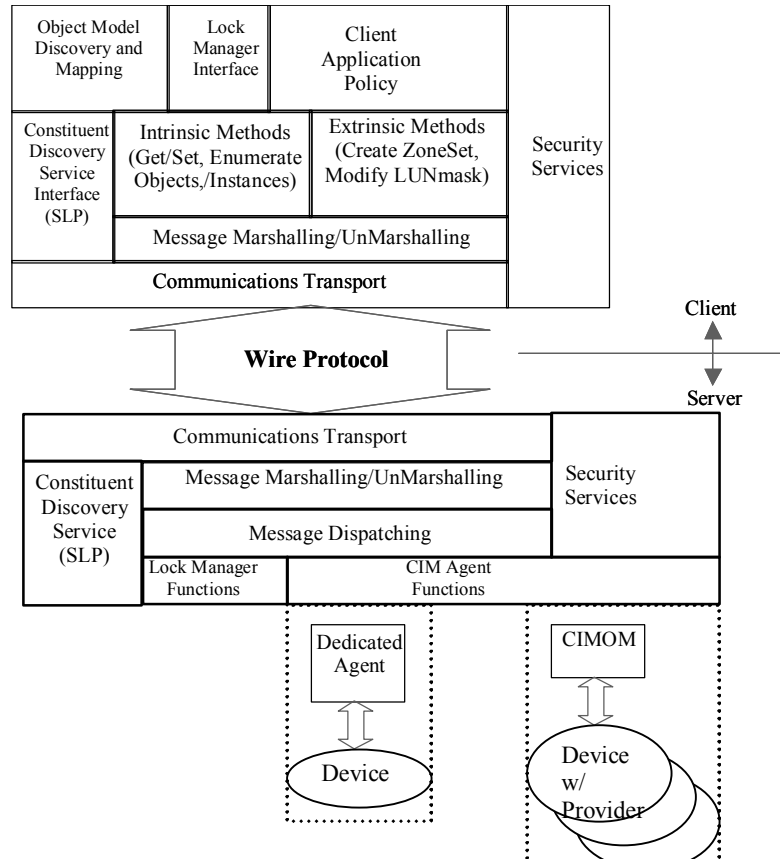


Figure 8: Operational Environment

1.3 Operational Environment

Figure 8 illustrates activities that either clients or servers may be required 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 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.

1.4 Using This Specification

This specification is insufficient as a single resource for the developers of Bluefin clients and servers. Developers are encourage 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 Protocols (SLP).

A developer implementing Bluefin clients/servers should read this specification in sequence noting that the section (Object Model) is intended principally as a reference relative to the particular device type that is being provided or managed in a Bluefin environment.

Developers engaging in the construction of a Bluefin environment that does not need to provide isolation among clients need not read the Clause on Locking.

Developers engaging in the construction of a Bluefin environment that does not require the automatic discovery of servers does not need to read the Clauses on Discovery, or Roles

1.5 Language Bindings

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

Several open-source sources are available for programmers who wish to streamline the task of parsing Bluefin messages into traditional procedure call semantics and using these semantics to store object instances. Consult the OpenGroup (<http://www.opengroup.org>) for current language bindings available to implement the Bluefin interface.

Clause 2: Transport and Reference Model

2.1 Introduction

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 will use the common transport (i.e., a reference model).

2.1.1 Language Requirements

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

- 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 for the language to express management information and related operations, as it meets the requirements above.

2.1.2 Communications Requirements

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

- 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 it meets the requirements above.

2.1.3 XML Message Syntax and Semantics

In order to be successful, the expression of XML management information (messages) across this interface must follow consistent rules for Semantics and Syntax. These rules should be of sufficient quality, extensibility, and completeness that they become widely adopted 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, C, 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 *XML-CIM Encoding* as developed and maintained by the DMTF. The WBEM source 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 XML-CIM message syntax was chosen because it meets the requirements of the storage industry as enumerated above. This specification extends the capabilities of XML-CIM in the areas of discovery and locking to facilitate ease of management and add reliability in the presence of non-cooperating clients competing for shared resources. Those extensions are explained later in this document.

2.2 Transport Stack

The complete transport stack for this interface is illustrated below in Figure 9. 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 in time. Thus, it should be noted that the transport stack has been layered such that (if required) other protocols may be substituted or added as technology evolves. For example, should SOAP or IIOP become prominent the content in the stack below can be adjusted causing minimal changes to existing product implementations in the market.

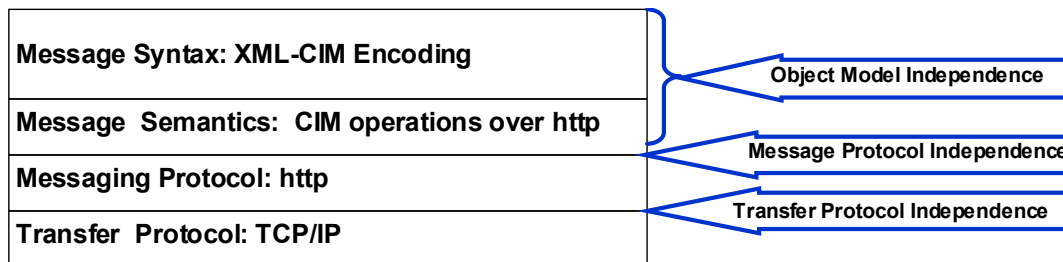


Figure 9: Transport Stack

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

1. 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 “intrinsic methods”. *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 “extrinsic methods”.
2. The second specification, *XML-CIM Encoding* details the precise W3C compliant syntax and grammar for encoding *CIM operations over HTTP* into XML.

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

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

2.3 Reference Model

2.3.1 Overview

As shown below in Figure 10: Reference Model, the Reference Model shows all possible constituents of the management environment in the presence of the transport stack for this interface.

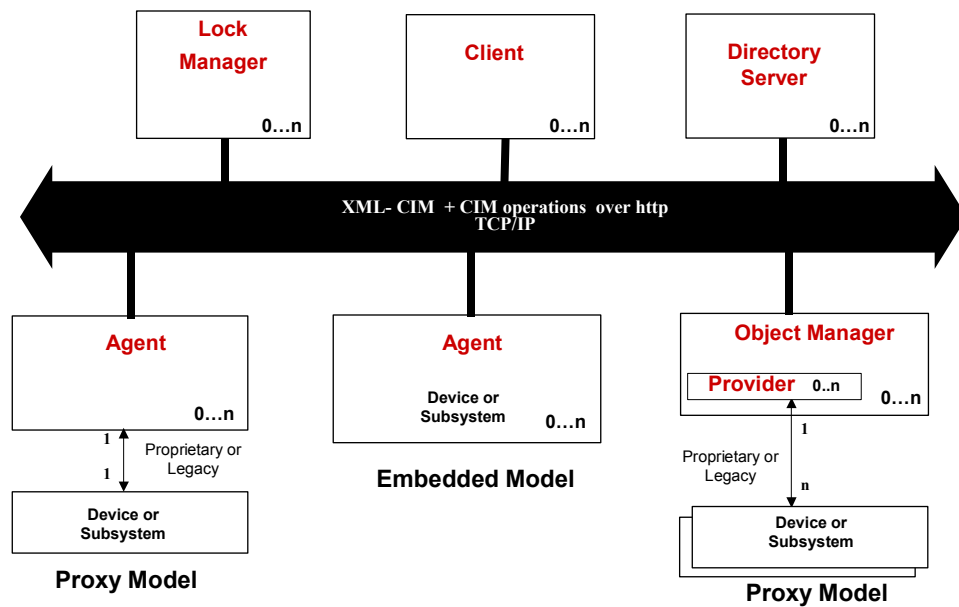


Figure 10: Reference Model

Figure 9: Transport Stack illustrates that the transport for this interface uses CIM operations over HTTP with XML-CIM Encoding and http/TCP/IP to execute intrinsic and extrinsic methods against the schema for this interface.

2.3.2 Roles for Interface Constituents

2.3.2.1 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.

2.3.2.2 Agent

An agent implements a subset of the object manager and as such controls only one device or subsystem and is typically incapable of providing support for complex intrinsic methods like schema traversal. An agent may be embedded in a device (like a Fibre Channel Switch) or provide a proxy 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 of a customer and eliminates the requirement for a stand-alone host (running the proxy agent) to support the device.

2.3.2.3 Object Manager

An object manager serves management information from multiple 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.

Provider

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

2.3.2.4 Lock Manager

A lock manager provides a common service for use by agents and object managers to coordinate resources between multiple non-cooperating clients such that Isolation and Consistency for the information in the schema is maintained.

2.3.2.5 Directory Server

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

2.3.3 Peer-to-Peer Access (Cascaded Clients)

This specification discusses constituents in the Bluefin environment in the context of Clients and Servers (Agents and Object Managers). However, these distinctions are only used to facilitate the easy introduction of the Bluefin reference model. In future versions, this architecture allows constituents (like virtualizers) in a Bluefin management environment to function as both client and server. This capability is referred to in this specification as cascaded clients. Special provisions will be made in this architecture to support cascaded clients in future Versions. Until those provisions are added to the specification, the Bluefin Clients and Servers will exist in a “flat” reference model. See *Appendix D: Futures* for a discussion of extensions to be applied to this specification.

Clause 3: Object Model

3.1 Model Overview (Key Resources)

The Bluefin object model is based on the Common Information Model (CIM), developed by the DRM working group of the SNIA. The Version 1 Bluefin Object Model is based on the 2.7 revision of CIM. For a more complete discussion of the full functionality of CIM and its modeling approach, see http://www.dmtf.org/standards/standard_cim.php.

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

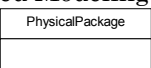
- CIM Tutorial
(<http://www.dmtf.org/education/cimtutorial/index.php>)
- CIM UML Diagrams and MOFs
(http://www.dmtf.org/standards/standard_cim.php)
- CIM System / Device Working Group Modeling Storage
(http://www.dmtf.org/var/release/Whitepapers/CIM_Device23_storage_wp.PDF)

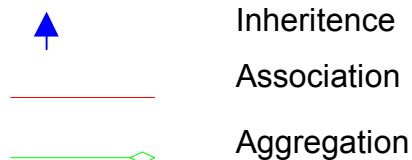
Managed Object File (MOF) is an ASCII file that expresses a formal definition a CIM schema. A MOF can be used as input into an MOF editor, parser or compiler for use in an application.

The Bluefin 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 discovery and instrumentation logic for a profile. In many implementations, providers operate in context of a *CIMOM* that is the datastore and infrastructure for a collection of providers. A CIM *client* interacts with providers running under one or more CIMOMs

3.1.1 Introduction to CIM UML Notation

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

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 may be added for *methods*.




Three types of lines connect classes.

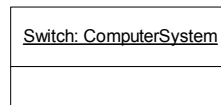
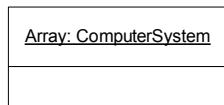
The CIM documents generally follow the convention of using **blue** lines for **inheritance**, **red** lines for **associations** and **green** lines for **aggregation**. The color-coding makes large diagrams much easier to read but is not a part of the UML standard.

The ends associations may have numbers (cardinality) indicating the valid count of object instances. Cardinality may be expressed 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,



represent an array and a switch – two instances of ComputerSystem objects.

3.2 Techniques

3.2.1 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 Bluefin object model.

Associations as Classes

CIM models associations and aggregations as classes that contain properties. The two endpoints are the *Antecedent* and *Dependent* properties. The association may also contain domain-related properties. For example, **ControlledBy** (see Clause C.16) associates a controller and a device. There is a many-to-many cardinality between controllers and devices (and 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 **AccessState** property of **ControlledBy** (see Clause C.16).

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 will create 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** (see Clause C.31) and **SCSIController** (see Clause C.75) objects associated by **ConcreteIdentity** (see Clause C.13).

Redundancy Groups

CIM models redundancy with an object representing the group of redundant objects. The **RedundancyGroup** subclass objects serve as a handle for operations on the entire group. The group can then be used in associations to the collection as an abstract entity. For example, a spare disk is associated with a **RedundancyGroup**.

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 **CIM_Slot** in the **CIM_Physical** MOF.

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", "Warmup",
        "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", "Timeout",
"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.

ComputerSystem (see Clause C.11): Most SAN products are modeled as **ComputerSystem**. The term “cluster” is used for systems with multiple loosely coupled processors; the individual processors known as “component” **ComputerSystems**. A cluster is modeled with a **ComputerSystem**; **ConcretelIdentity** associates the cluster **ComputerSystem** and a **RedundancyGroup** that aggregates the component **ComputerSystems**. **ComputerSystem**'s "dedicated" property describes the functions provided by a system (e.g., host, storage system, switch).

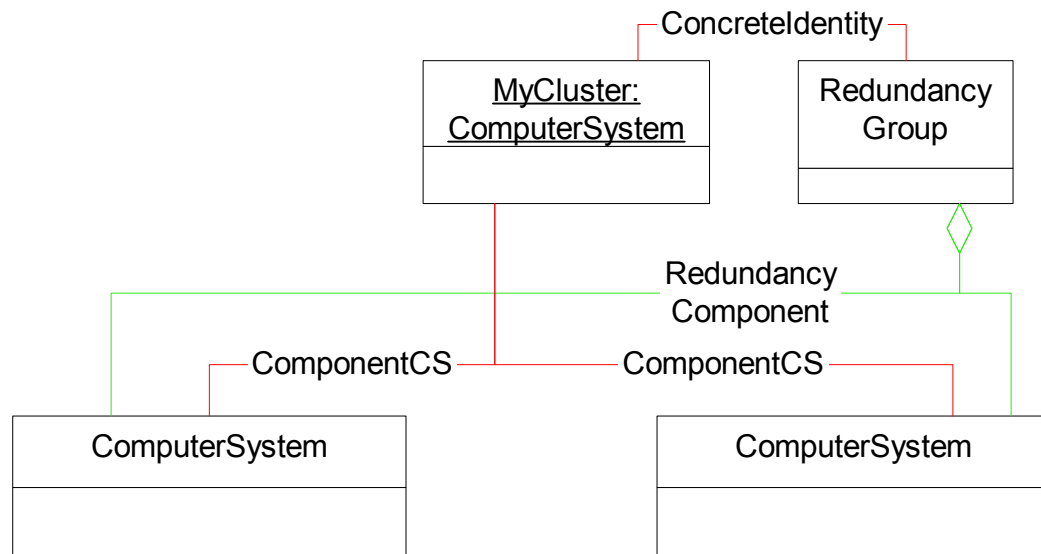


Figure 11: Cluster Model

PhysicalPackage represents the physical storage product. **PhysicalPackage** can be sub-classed to **ChangerDevice** (see Clause C.7), but **PhysicalPackage** accommodates products deployed in multiple chassis.

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

SoftwareElement (see Clause C.84) models firmware and optional software packages. **InstalledSoftwareElement** associates **SoftwareElement** and **ComputerSystem**. **DeviceSoftware** associates **SoftwareElement** 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 DiskDrive and TapeDrive) and also intermediate nodes like Controller and FCPort. Each LogicalDevice subclass must be associated to a ComputerSystem with a SystemDevice aggregation. Dues to the large number of LogicalDevice subclasses, SystemDevice aggregations are often omitted in instance diagrams in this specification.

The following diagram combines these common elements; this combination is used in several of the profiles.

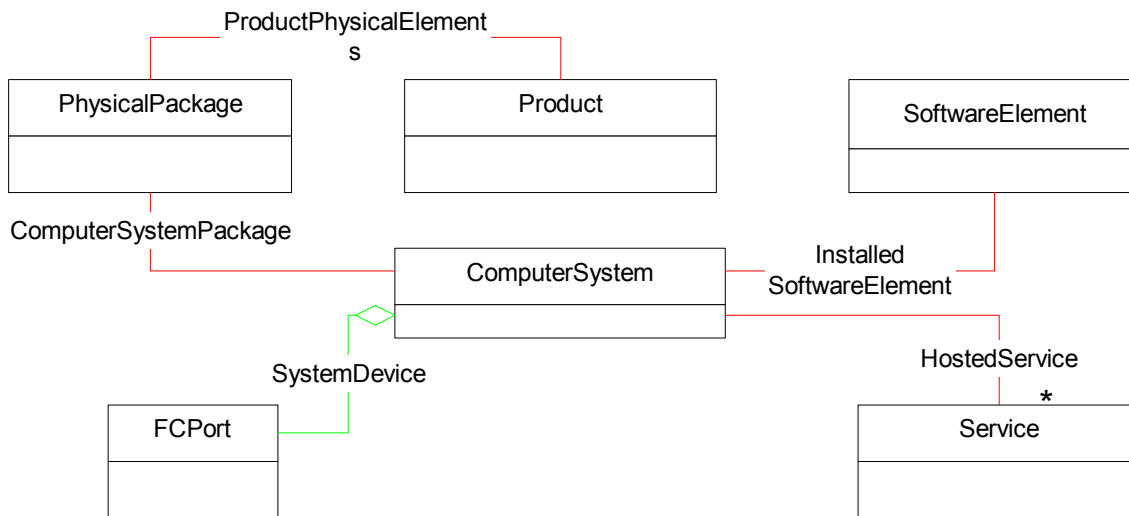


Figure 12: Common Elements

This specification covers many common storage models and management interfaces, but some implementations may include other objects and associations. In some cases, these may be modeled by CIM schema 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.

3.2.2 Modeling Profiles

In addition to modeling SAN components, Bluefin agents/providers must 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 CIMOM/agent and automatically determine the appropriate SLP Service Template information (see *Clause 5:Service Discovery*)

ProviderCapabilitiesMajorCategory defines the standards group defining the profile. Setting this to “Storage” indicates that one of the Bluefin/SNIA profiles applies. **MinorCategory** is an array of strings that define the specific Bluefin/SNIA profile (e.g., Switch, Tape Library, ...). A client can enumerate the Providers in each CIMOM to see which providers (and objects) claim Bluefin compliance.

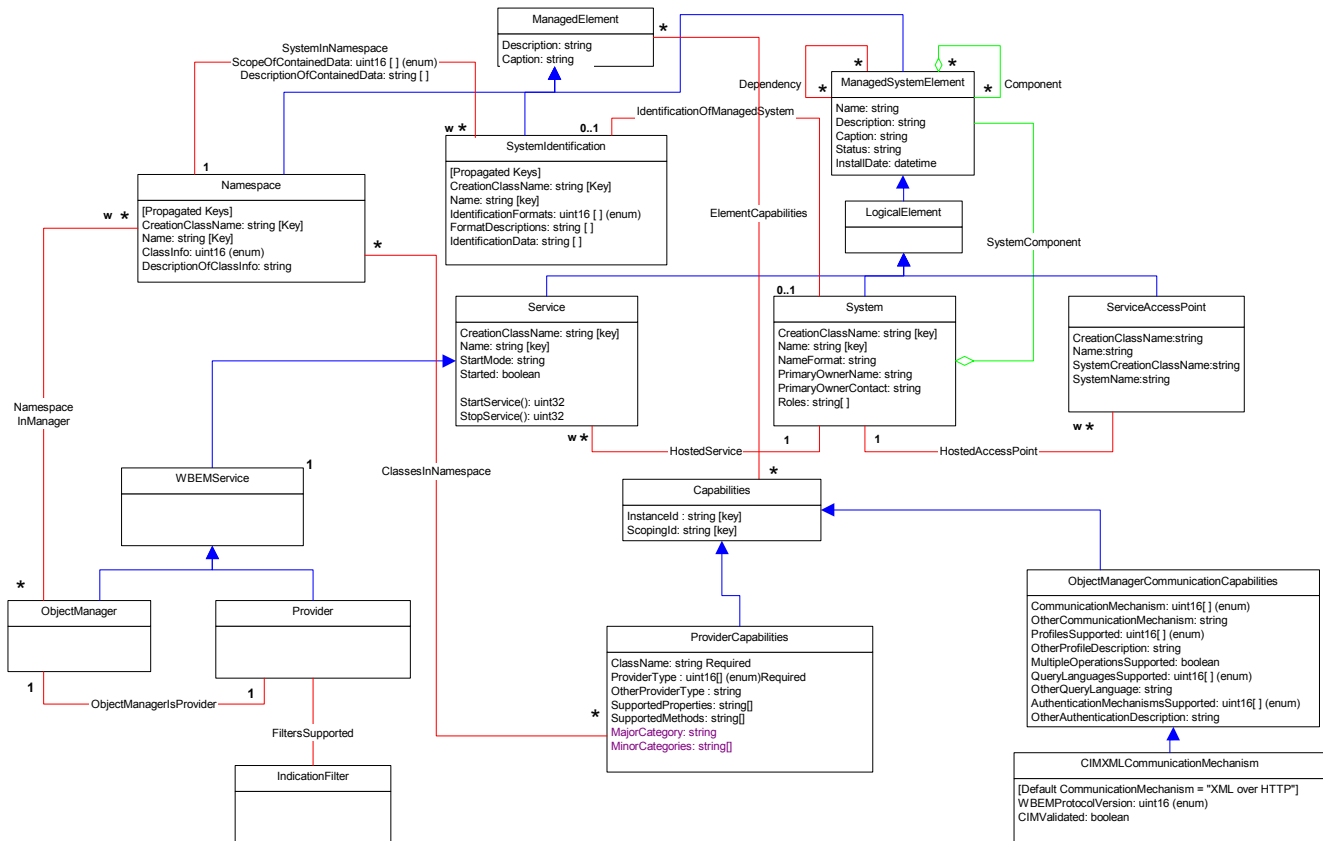


Figure 13: WBEMService Hierarchy

We are requesting two new properties in **ProviderCapabilities**. **MajorCategory** defines the standards group defining the profile. Setting this to “Storage” indicates that one of the Bluefin/SNIA profiles applies. **MinorCategory** is an array of strings that define the specific SNIA profile (e.g., Switch, Tape Library). A client can enumerate the **Providers** in each CIMOM to see which providers (and objects) claim Bluefin compliance.

Bluefin needs to reevaluate this model when it is incorporated in CIM and complete the Bluefin profile for its use.

3.2.3 Durable Names

Management applications need to read and write information about managed objects in multiple CIM namespaces. 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 needs a way to understand when objects in different namespaces represent the same managed resource. A unique common identifier, referred to as a durable name, is designated as a required property for any objects representing managed resources that might be “seen” from multiple points of view. These durable names can be used by a management application for object coordination.

Durable names thus provide a means of reliably “stitching together” information from multiple sources about the same managed resource in a SAN. They also provide a means of stitching together information obtained at different points in time, such as when a managed resource is returned to a SAN after having been removed for some period of time.

A necessary technique associated with durable names involves the use of the **NameFormat** property. CIM key value combinations are unique across all instances of a class within a single namespace, 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 durable names whenever different types of identifiers are possible.

When different types of identifiers are possible, objects requiring durable names must support a **NameFormat** property that selects one of a set of prescribed strings that define valid identifier types for the class. Appropriate format values are defined for each class where different identifier types can be an issue. Each valid identifier type for a class is included as a separate property of an object. If an implementation instantiating such an object does not support certain identifier types, then those properties are left blank. For each class, a preferred order is established for setting the **NameFormat** property to one of the non-blank valid identifier types, resulting in a consistent approach for forming a durable name for the object.

Durable names are required for the following objects in the model:

- **StorageVolume**
- **FCPort**
- *Fabric (see AdminDomain)*
- **ComputerSystem** objects with the following roles
 - Host
 - Management Appliance
 - CIM Server
 - Switch
 - Router
 - Bridge
 - Extender
 - Block Server
 - **StorageLibrary Server**
 - Enclosure Server

Note that CIM keys and durable names are not tightly coupled. For some classes, they may be the same thing, but this is not required as long as all durable names are unique and management applications can determine when objects in different namespaces are providing information about the same managed resource in a SAN. In the cases where CIM keys and durable names are not the same thing, multiple CIM operations may be required to satisfy asset management use cases.

The main types of information used for durable names include SCSI mode page information, Fibre Channel World Wide Names, Fully Qualified Domain Names, and IP Address information. The details for each class requiring durable names are provided in the Profiles section of this document. An overview of the information used to form durable names for objects is as follows:

- **StorageVolume**: Multiple valid identifier types exist, and **NameFormat** is used to indicate which is used. Durable names are based upon SCSI mode page information.

- **FCPort**: Durable names are based upon Fibre Channel Port World Wide Name information.
- **Fabric**: Durable names are based upon Fibre Channel World Wide Name information. Note that this durable name can change under some circumstances, such as when the Fibre Channel fabric is partitioned or when the principle switch in a fabric fails.
- **PhysicalPackage** or subclass can be used to satisfy asset management use cases, by making Vendor/Model/Serial Number information available.
- **ComputerSystem** roles “Switch”, “Router”, “Bridge”, “Extender”, and “Enclosure”: Durable names are based upon Fibre Channel World Wide Name information.
- **ComputerSystem** roles “Block Server” and “StorageLibrary”: Multiple valid identifier types exist, and **NameFormat** is used to indicate which is used. Durable names are based upon Fibre Channel World Wide Names or IP Address information. Note that when Fibre Channel World Wide Names are used, the durable name may be a list of Fibre Channel World Wide Names.

Durable names are not supported for **SCSIController** objects. This is because in Fibre Channel there exists a one-to-one relationship between **SCSIController** objects and corresponding **FCPort** objects. Since **FCPort** objects have durable names, **SCSIController** object instances can be unambiguously identified using the association to the corresponding **FCPort** object instance.

3.2.4 Events – CIM Indications

3.2.4.1 Background

Indications are the mechanism used to accomplish the following functional capabilities in Bluefin (from the list of Bluefin capabilities, clause 1.2):

1. Allow a client to receive asynchronous notification that the configuration of a SAN has changed.
2. Allow a client to receive asynchronous notification that the health of a SAN resource has degraded.
3. Allow a client to receive asynchronous notification that the performance of a SAN interconnect has degraded.

CIM Indications are described in a DMTF white paper, which can be obtained from the DMTF web site, <http://www.dmtf.org/education/whitepapers.php> (follow the link labeled “CIM V2.5 Event Model”). This paper, “Common Information Model Indications”, is at <http://www.dmtf.org/var/release/Whitepapers/DSP0107.pdf>.

Indications are also used in place of non-blocking methods for long-running operations. In most cases, the operation requested in a method completes quickly, the return status from the method indicates the status of the operation. When a long-running operation (such as RAID volume creation) is requested, the method return code indicates whether the operation started successfully; an indication is sent when the operation is complete. Information on the indication is included in the profile whenever long-running operations are implemented.

3.2.4.2 Using indications

Clients request indications to be sent to them by subscribing to the indications. Subscriptions are stored in CIMOM as CIM object instances. A *Subscription* is expressed by the creation of an **IndicationSubscription** association instance that references a **IndicationFilter** (a filter) instance, and an **IndicationHandler** (a handler) instance. A Filter contains the query that selects an indication class or classes.

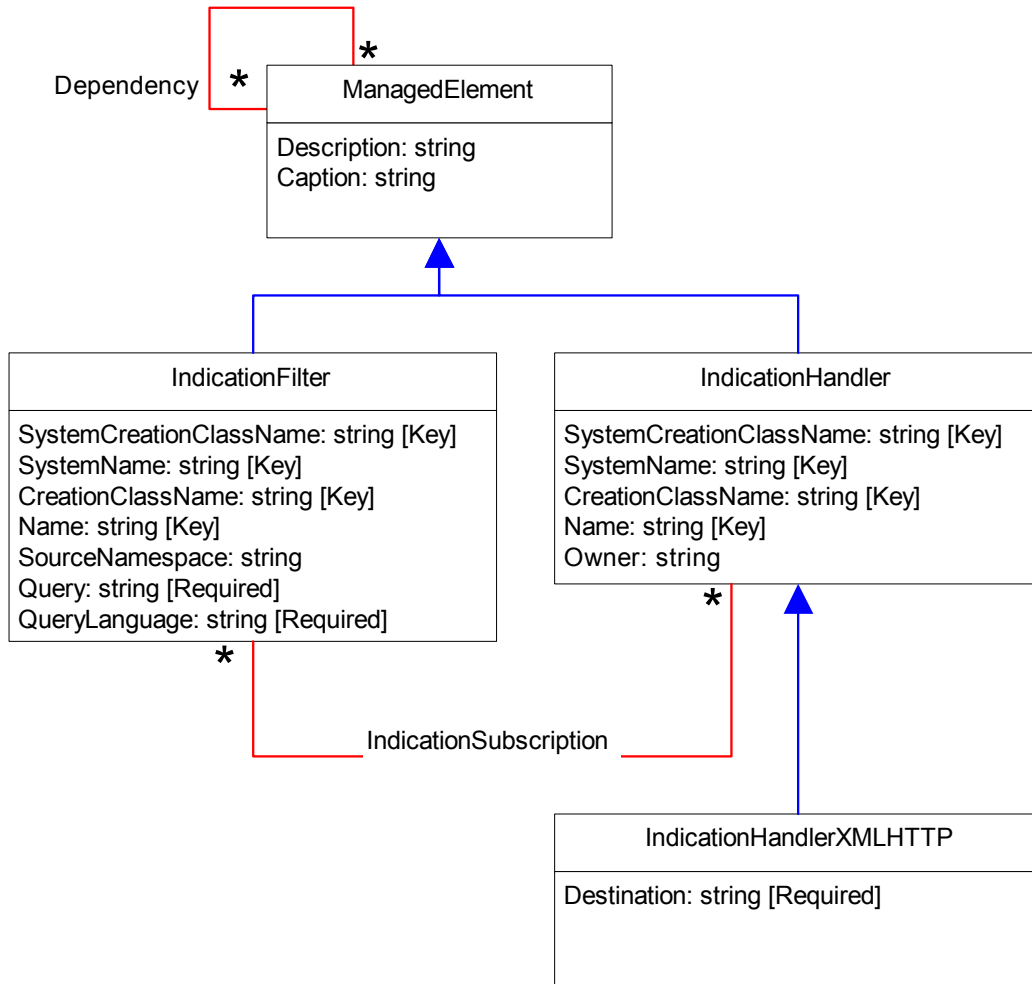


Figure 14 - Indications Filters Schema

Filters can be created by indication consumers (e.g. Bluefin Clients) or indication providers (e.g. Bluefin Agents). The client would create these 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 CIM_AlertIndication"
```

```
"SELECT * FROM CIM_InstModification WHERE SourceInstance ISA CIM_ComputerSystem"
```

AlertIndication and **InstModification** are types of indications (see the following section). 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).

IndicationHandler specifies the means of delivering indications to the client. The subclass **IndicationHandlerXMLHTTP** 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. However, to the extent possible, it is desirable to put all relevant information in the indication.

3.2.4.3 Indication hierarchy

Indications are grouped in three broad categories, **ClassIndication**, **InstIndication**, and **ProcessIndication** (see Figure 15 - Indications Schema).

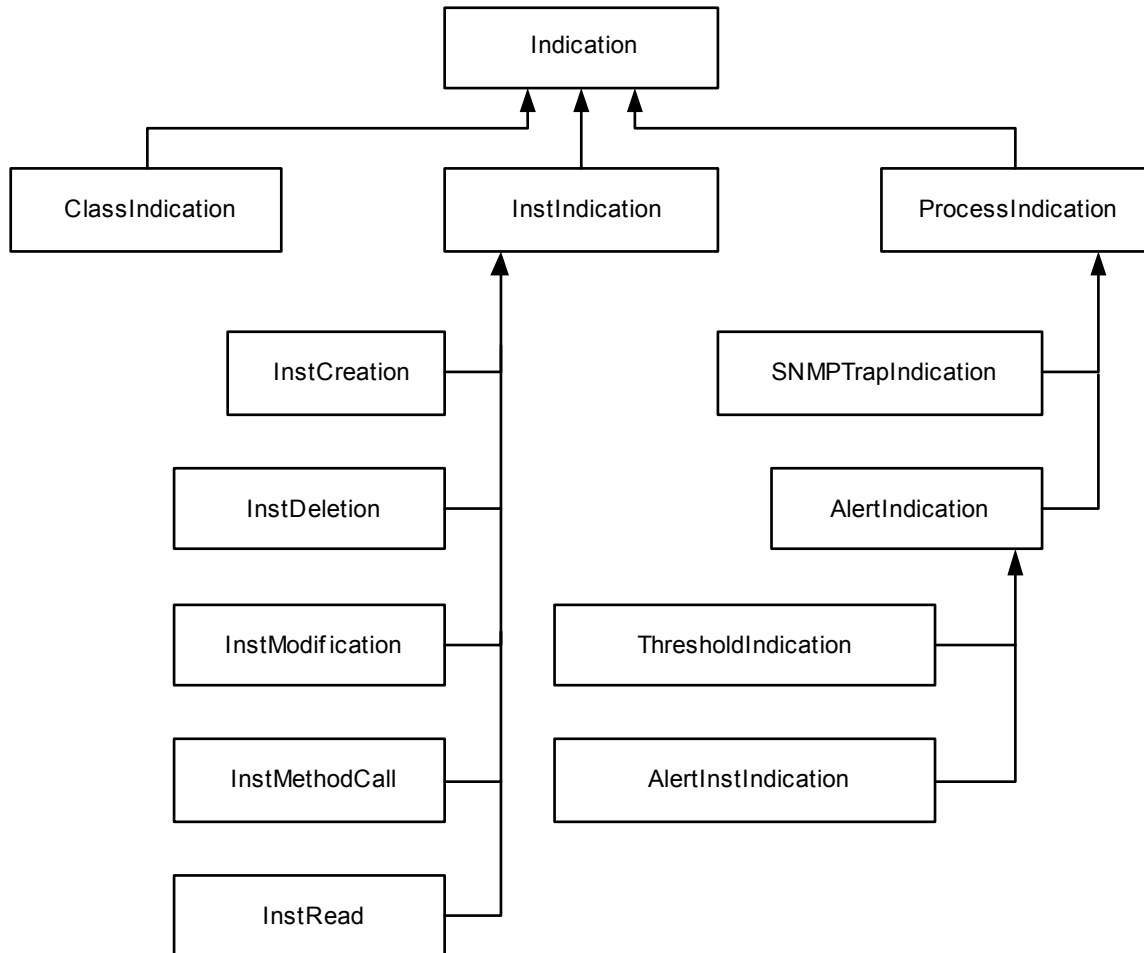


Figure 15 - Indications Schema

A **ClassIndication** is delivered in response to the creation, deletion, or modification of a class, i.e. when there are changes to the schema. Bluefin clients generally will not need to subscribe to **ClassIndications**. **InstIndication** is delivered in response to the creation, deletion, modification, etc. of an instance. For example, when a new volume is created or a zone is deleted. **ProcessIndication** allows for indications that are not associated with changes to specific instances. An event can be modeled with one of three indication subclasses.

- An **InstIndication** contains an embedded copy of the object that generated the indication. In the case of an **InstModification**, there will be two copies, one with the old value and the other with the new value. These embedded copies may be full copies of the object, or they may contain only the properties that have changed.
- An **AlertIndication** is a simpler indication that contains just strings and enumerated types. One of its properties is the path to the object generating the alert. Other properties include alert type, severity, and description. An **AlertIndication** can be used to indicate changes in the health condition or other state of a SAN.

A **SNMPTrapIndication** is designed to encapsulate the information from an SNMP trap in an indication. Without a standardization process, **SNMPTrapIndications** will not be interoperable and are discouraged in Bluefin agents.

In general, it is best to use **InstIndication** for all events that result in the creation, deletion, or modification of instances in the Bluefin Agent.

3.2.4.4 Agent/Provider Considerations

As mentioned above, a Bluefin profile can be deployed as a proxy provider running in a general-purpose CIMOM or as a Bluefin agent – a combination lightweight CIMOM and provider – used when CIM is embedded on a device.

Considerations that apply to either deployment:

- A general purpose CIMOM (and perhaps an embedded agent) allows a client to create indications filters. The provider may 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** should return unsupported for an indications filter create request.
- Agents must persist subscription information across reboots; for CIM, the subscription information is **IndicationFilter** and **IndicationHandler** classes.
- An **InstIndication** subclass can only embed a single instance. A hardware configuration change may involve many instances of objects and associations. We recommend that agents detect and merge groups of related hardware events and then send a single indication for an object identifying the system using the **SystemCreationClassName** property. The client is expected to rediscover the indicating system to determine the impact of the change.

3.2.4.4.1 Bluefin Embedded Agent Considerations

A Bluefin Agent can minimize footprint by initializing “canned” **IndicationFilter** objects and returning “unsupported” for all requests to create filter instances. A Bluefin client can determine what indications the agent supports by enumerating these objects. A minimal embedded agent can simply support a subset of these **IndicationFilter** query strings:

1. `"SELECT * FROM CIM_InstIndication"`
2. `"SELECT * FROM CIM_AlertIndication"`

The presence of an **IndicationFilter** object with query string 1 indicates that the agent supports **InstIndication**, and similarly for the others.

The embedded agent should supply more detailed queries as described in the profile sections that follow.

A standard implementation of indications requires the agent to accept client requests to create indication handlers. Other aspects of Bluefin profiles do not require the agent to handle instance creation requests (the CIM operations “Basic Read” functional group). The embedded agent implementer has two options:

- Use the Instance Manipulation functional group rather than Basic Read. The agent may treat non-indications instance creation requests as unsupported. At a minimum, the agent must allow instance creation of **IndicationHandlerXMLHTTP** and **IndicationSubscription** instances.
- If the agent wishes to provide NO instance creation, then the agent needs to provide a backdoor for indications subscribers. For example, the agent can require customers to edit a text file describing indications subscriptions.

If the agent opts for no indications support, it must assure that no `IndicationFilter` instances exist in the Bluefin Agent and to return “unsupported” to requests to create instance of `IndicationHandler` instances.

3.2.4.5 Client Considerations

The client needs to determine whether each target CIMOM is an embedded Bluefin agent or a general-purpose CIMOM with a Bluefin provider. The client should try to create an instance of an `IndicationHandlerXMLHTTP`. If the embedded agent does not allow the client to subscribe via CIM, it returns unsupported. The client can then enumerate `IndicationHandlerXMLHTTP` instances to determine whether they are subscribed via some non-CIM facility.

If the client can create an `IndicationHandlerXMLHTTP` instance, it should then try to create an `IndicationFilter` instance; a return of unsupported indicates the CIMOM is an embedded agent and supplies its own filters. In this case, the client enumerates the existing filters and creates `IndicationSubscription` associations to their `IndicationHandler`.

The client can minimize the number of filters by using the indications schema hierarchy. For example, subscribing to `InstIndication` is the same as subscribing to `InstCreation`, `InstDeletion`, and `InstModification`.

Client needs to consider subscriptions that generate excessive events. Subscriptions to a general-purpose CIMOM (as determined by the tests described above) should be specific to the provider – for example “select * from HDS_InstIndication” rather than “select * from CIM_InstIndication”.

When a client receives an `InstCreation` subclass, it needs to rediscover the indicating system to determine the associations and other classes impacted by the configuration change. Providers may opt to consolidate complex configuration changes into a single indication.

The general algorithm for client subscription is:

```

Look for an existing IndicationHandlerXMLHTTps

If one exists targeting your indication listener,
    Then you are already subscribed from agent persistence, exit
Else Create an IndicationHandlerXMLHTTP instance

If the response is “unsupported”,
    Then quit (this provider does not support indications)

Enumerate IndicationFilters

If the desired filter instances do not exist,
    Then try to create them.

If filter instance creation requests fail,
    Then backoff to an existing filter.

Create instances of IndicationSubscription associating the desired filters and your
handler.
```

The client should look for status changes represented as either `AlertIndication` or as `InstModification` with a status change. With `InstModification`, the current and previous statuses can be compared; for example:

```

“select * from Compaq_InstModification
  where PreviousInstance.Status <> SourceInstance.Status”
```

The DMTF events white paper has other examples of filter queries.

The client can use indications to get information about the general health of the SAN. For example, the class `FCPortStatistics` includes among its properties various error counts. A query like this:

```

Select * FROM CIM_FCPortStatistics
  WHERE PreviousInstance.ErrorFrames < SourceInstance.ErrorFrames
```

will generate an indication whenever the `ErrorFrames` count increases.

If the client is unable to create this query (i.e. if the agent doesn't support this filter), then the client can periodically read the `FCPortStatistics` of all the `FCPorts` in the model. This method, however, is much more expensive in terms of communications bandwidth and load on both the client and the server.

3.2.4.6 Requirements

Bluefin Clients must use the subclass `IndicationHandlerXMLHTTP` when creating subscriptions.

If indications are supported, then the DMTF query language **MUST** be supported (this is a DMTF requirement).

3.2.4.7 Implementation Considerations

The encoding of indications is specified in "WBEM Query Language Draft". The specification is still in draft and requires DMTF membership to access. See "WBEM Query Language Draft", Version 2.4, DMTF, June 14, 2000,

<http://www.dmtf.org/members/review/wip/DMTF-query/DSP0104.htm>

The specification for embedding objects is also still in draft and requires DMTF membership. See "EmbeddedObject Qualifier", CR526, DMTF, October 26, 2000

<http://www.dmtf.org/members/tdc/wg-events/.admin/sitemgr.cgi/members/tdc/wg-events/archive/cr526.html>

3.2.5 Device Credentials

The device credentials are modeled using the CIM classes `SharedSecretService` and `SharedSecret`. The `ComputerSystem` class represents the device, and the `SharedSecret` object contains the credentials in its properties.

A Bluefin client or application can pass the device credentials to the agent or object manager by instantiating the `SharedSecret` object, using the CIM intrinsic method `NewInstance()`. The Bluefin agent or provider uses the information from this object to talk to the device.

More information is in the Clause 4.3, "Modeling Device Credentials".

3.3 Profiles

3.3.1 Profile Content

The Bluefin Object model is described in a sequence of profiles, each of which addresses a general class of SAN entities (e.g., Host, Fabric, Array). Each profile is defined in subsections which are described below.

Profile Element	Goal
Description	A textual introduction to the SAN entity being profiled. It provides a high-level foundation for the more detailed descriptions to follow.
Schema Diagram	A diagram of the subset of the Bluefin Object Model that is most concerned with the SAN entity being described. Schema diagrams include all of a profile's classes

Profile Element	Goal
	and associations; the class hierarchy is included and each class is depicted one time in the schema diagram.
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.
Client Considerations	This section summarizes the implementation concerns that will be encountered by products and services that rely on the SAN entity being described.
Agent Considerations	This section summarized the implementation concerns that must be accounted for by agent implementations (either embedded or proxy) that provide information from one or more of the SAN entities to Bluefin clients.
Indications	This section details any indications that have been defined in conjunction with this SAN entity.
Required Classes	<p>This section provides a table listing the classes upon which this profile relies, information on whether the class is required for the particular profile, and profile-specific notes. Each class reference includes a cross-reference to the detailed definition of the class.</p> <p>The classes are listed in a table with a column defining whether the class is required for this particular profile. This column can have three values</p> <ul style="list-style-type: none"> Y “Yes”, the class is required for this profile N “No”, this class is optional. I “Implementation”, if an implementation of this profile includes a particular behavior, than this class is required. For example, a disk array may or may not have a StorageConfigurationService interface for creating volumes; if it does, than this service and some related classes are required.

Table 1: Profile Components

3.3.2 Fabric

3.3.2.1 Fabric Topology

3.3.2.1.1 Description

3.3.2.1.1.1 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 Component. **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/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 thus left up to the implementation (though it must be unique within the discovery of known SANs that populate the same CIM Namespace).

For Fabrics, the identifier is the Fabric WWN which is based on the primary switch.

3.3.2.1.1.2 Fabrics and Topology

A Fabric in CIM today minimally contains a **LogicalNetwork** or the Systems it contains, or more likely both. For the purposes of this discussion, it is assumed one will model both.

LogicalNetwork represents the foundation necessary for routing (and the reason it is defined in the Network model). A **LogicalNetwork** groups a set of **Protocol Endpoints** together, which are able to communicate with each other directly. The **ProtocolEndpoint** is associated to the **LogicalNetwork** by **MemberOfCollection**. A link is represented by the association **ActiveConnection**, which associates two **ProtocolEndpoints**, which is 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 Port 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 apply to FibreChannel also.

One can ultimately represent multiple **LogicalNetwork** (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 **LogicalNetworks**, a **LogicalNetwork** 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 paper) requires a **LogicalNetwork**.

3.3.2.1.1.3 Systems and Ports

As discussed in the previous section, a Port is associated to a device to represent the link layer. A Port is associated to the **ProtocolEndpoint** by **DeviceSAPImplementation** and "joins" the System and Device model to the Network model. 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 "", 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**.

3.3.2.1.2 Schema Diagram

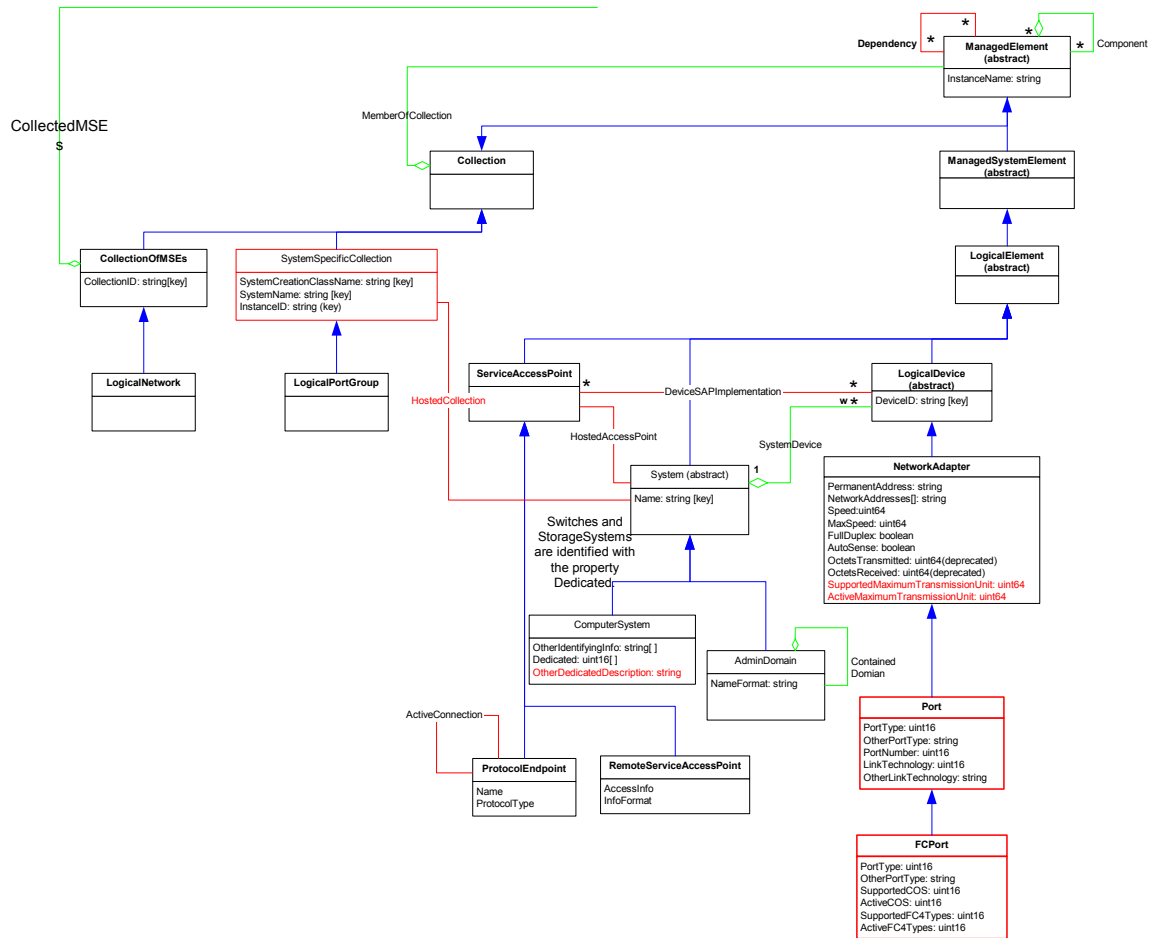


Figure 16: Fabric Schema

3.3.2.1.3 Instance Diagram

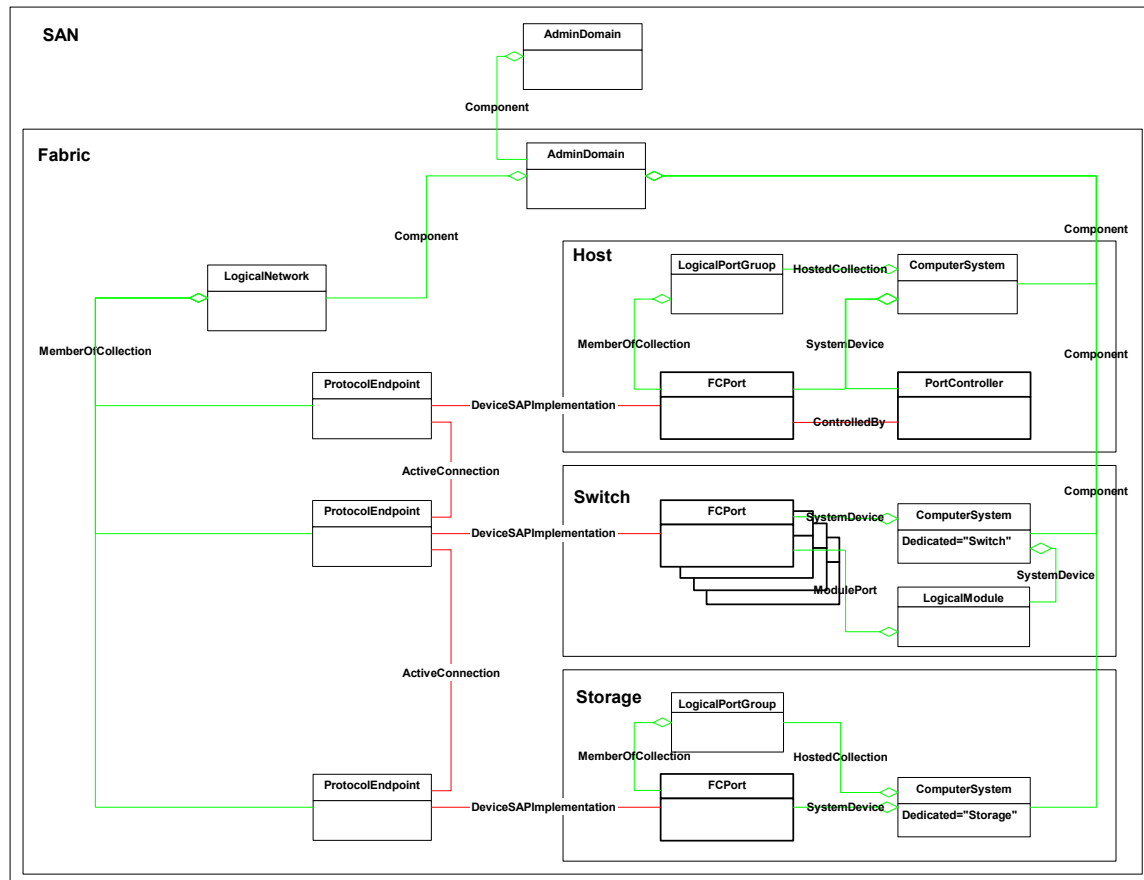


Figure 17: Fabric Instance Diagram

3.3.2.1.4 Client Considerations

The client needs to consider that the fabric identifier is not durable. Please see Clause 3.2.3. In addition, the client needs to consider that agents may not maintain persistent fabric objects (e.g. ports, connections, etc.).

3.3.2.1.5 Agent Considerations

Life cycle indications supported by the CIMOM should not be done in the first release. Basically the provider/agent should return "Not supported" when the CIMOM polls for events at frequent intervals.

The namespace for each vendor provider/agent should be different. The agent/provider should create instances under a namespace that is unique in the CIM Server. This is to allow multiple providers/agents from different vendors to co-exist in a CIM server.

The agent will need 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 not longer be discovered are implementation specific. If a single vendor agent/provider provides information about multiple switches from the same vendor, then the instances created to represent those switches should be created under the same namespace.

If the agent is unable to determine the type of platform discovered (defined in FC-GS), then the agent must set the ComputerSystem.Dedicated property to "Unknown".

3.3.2.1.6 Indications

See Clause 3.3.2.3.6 for information on fabric indications.

3.3.2.1.7 Required Classes

Class	Req	Notes
ActiveConnection (p. 214)	Y	
AdminDomain (p. 214)	Y	for Fabric
AdminDomain (p. 214)	N	for SAN
Component (p. 217)	N	for zoning orphaned devices
Component (p. 217)	Y	LogicalNetworks and ComputerSystems into Fabrics
Component (p. 217)	N	Fabrics into SANs
ComputerSystem(p. 217)	Y	
Dependency (p. 221)	Y	
ElementStatistics (p. 223)	Y	
FCPort (p. 224)	Y	
FCPortStatistics (p. 228)	Y	
HostedAccessPoint (p. 231)	N	
HostedCollection (p. 231)	Y	LogicalPortGroup to ComputerSystem
HostedCollection (p. 231)	Y	of LogicalNetwork
LogicalNetwork (p. 239)	Y	
LogicalPortGroup (p. 239)	Y	
MemberOfCollection (p. 240)	Y	Into LogicalPortGroup
MemberOfCollection (p. 240)	Y	LogicalNetwork to AdminDomain
ProtocolEndpoint (p. 246)	Y	
RemoteServiceAccessPoint (p. 248)	N	
SystemDevice (p. 288)	Y	

Figure 18: Fabric Required Classes

3.3.2.2 Switches

3.3.2.2.1 Description

The switch profile models the physical and logical aspects of a Fibre Channel fabric interconnect element. The **ComputerSystem** class constitutes the core of the switch model. It is identified as a switch using the property **Dedicated** set to "switch".

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.

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.

3.3.2.2.2 Schema Diagram

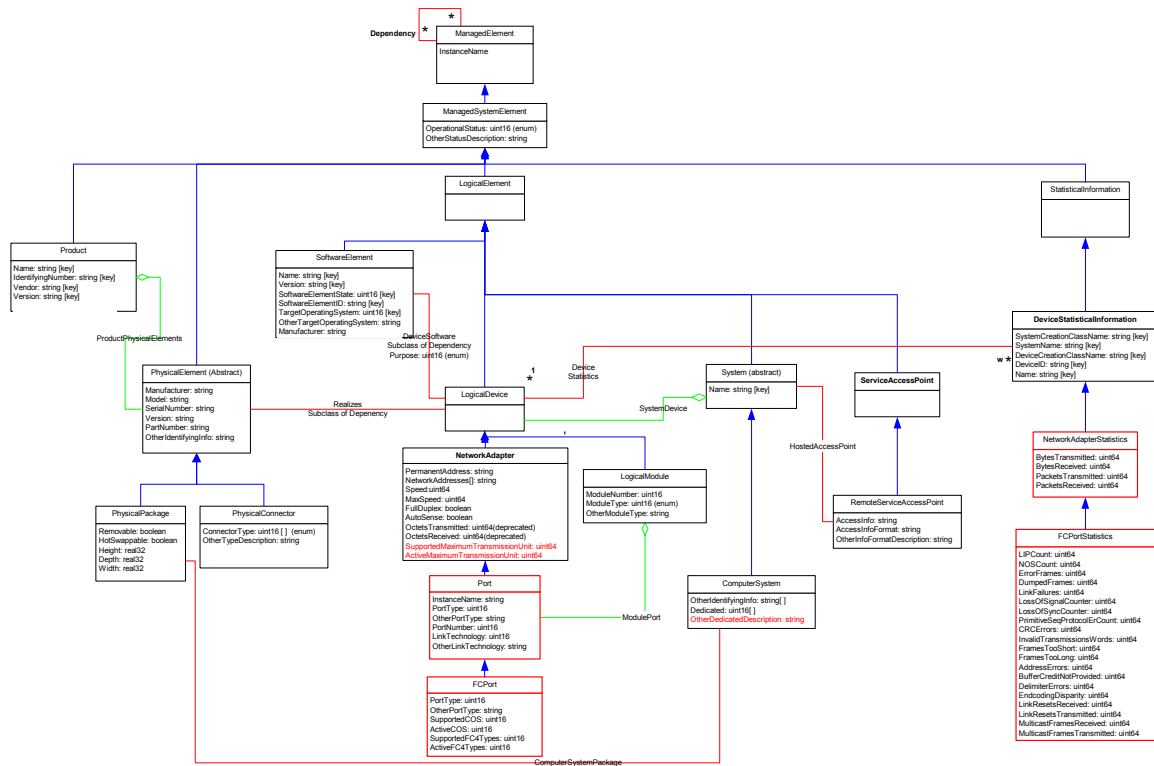


Figure 19: Switch Schema Diagram

3.3.2.2.3 Instance Diagram

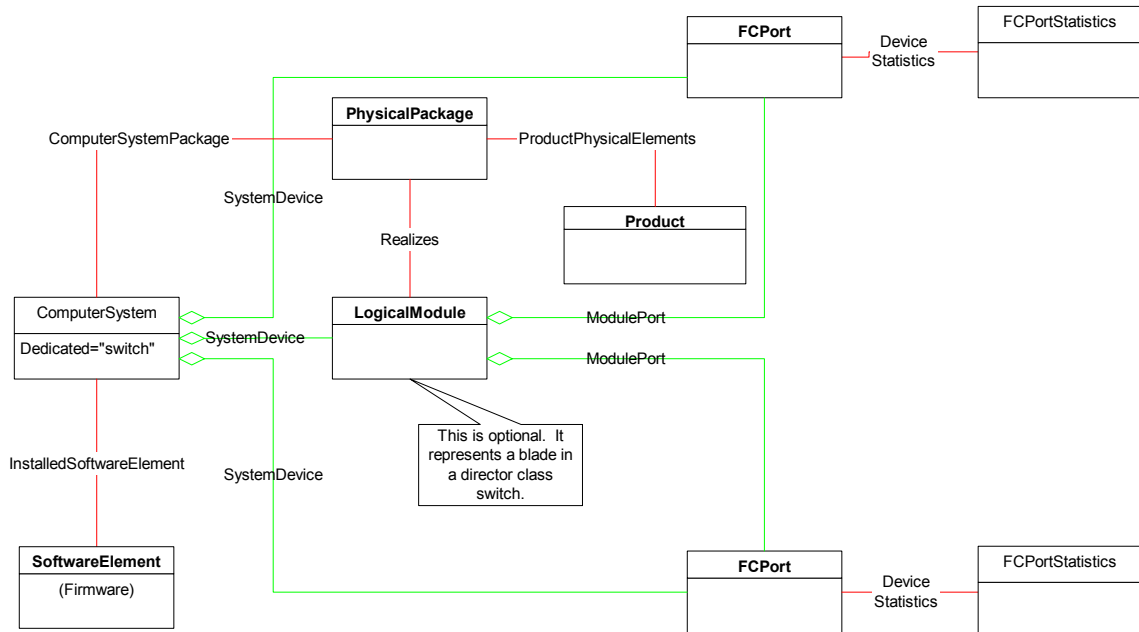


Figure 20: Switch Instance Diagram

3.3.2.2.4 Client Considerations

An agent may represent more than one switch within a fabric. Enumerating `ComputerSystem` where `Dedicated="switch"` will return the list of switches that the agent is able to discover and possibly manage.

3.3.2.2.5 Agent Considerations

The information about the device that is supposed to be managed by the provider running on host (proxy agent) is implementation specific.

The vendor switch profile should contain the information about the provider (proxy agent) that is responsible for retrieving the switch information. There is no standard registration mechanism defined for the provider to register with a CIMOM.

There could be one provider developed to provide information about the fabric as well as the switch.

3.3.2.2.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA
      CIM_ComputerSystem
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA
      CIM_ComputerSystem
```

- Modification

```
SELECT * FROM InstModification WHERE SourceInstance ISA CIM_FCPort AND
      SourceInstance.OperationalStatus[0] <>
      PreviousInstance.OperationalStatus[0]
```

```
SELECT * FROM InstModification WHERE SourceInstance ISA
      CIM_ComputerSystem AND SourceInstance.OperationalStatus[0] <>
      PreviousInstance.OperationalStatus[0]
```

- Optional

- Instance Indications

- None

3.3.2.2.7 Required Classes

Class	Req	Notes
ComputerSystemPackage (p. 218)	Y	
Dependency (p. 221)	I	
DeviceSoftware (p. 223)	N	
ElementStatistics (p. 223)	Y	
FCPort (p. 224)	Y	
FCPortStatistics (p. 228)	Y	
LogicalModule (p. 239)	N	
ModulePort (p. 240)	N	
PhysicalConnector(p. 241)	N	
PhysicalPackage (p. 243)	Y	
Product (p. 245)	Y	
ProductPhysicalElements (p. 246)	Y	
Realizes(p. 247)	Y	
RemoteServiceAccessPoint (p. 248)	I	
SoftwareElement (p. 253)	N	
SystemDevice (p. 288)	Y	

Table 2: Switch Required Classes

3.3.2.3 Zoning

3.3.2.3.1 Description

The Zoning model is based on ANSI FC-GS-4. This model represents the management model of defining and "activating" a zone versus the simple operational model found in other transports which shows connectivity.

The model defines the containment of the zone objects as the **AdminDomain** representing the Fibre Channel Fabric. The **AdminDomain** hosts the service **ZoneService** which has the operational functions to control the zone (e.g. Activate **ZoneSet**, etc.).

Within the **AdminDomain** representing the fabric are **ZoneSet** associated via **HostedCollection**. The **ZoneSet** contains **Zones** associated by **MemberOfCollection**. Devices in the **Zone** are associated into the **Zone** via the association **ZoneMember**. Note that the association **ZoneMember** has additional properties which indicate how the device was "zoned" (e.g. WWN, FCID, etc.).

Zones and **ZoneSets** contain the property **Active** which indicates whether the **Zone** and **ZoneSet** are part of the "active" definition defining the Fibre Channel fabric zoning. **Zone** and **ZoneSet** that are active cannot be modified.

3.3.2.3.2 Schema Diagram

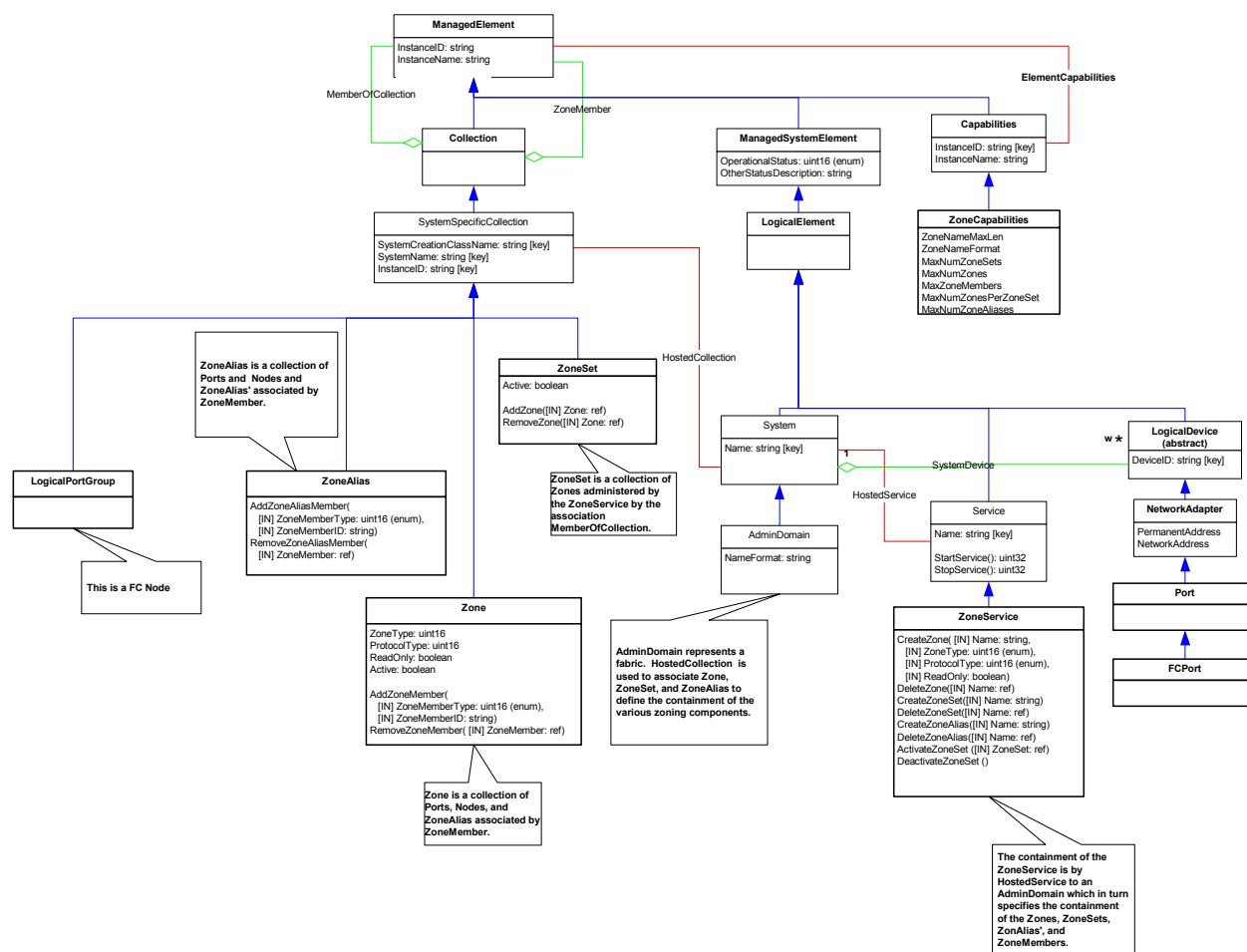


Figure 21: Zoning Schema

3.3.2.3.3 Instance Diagram

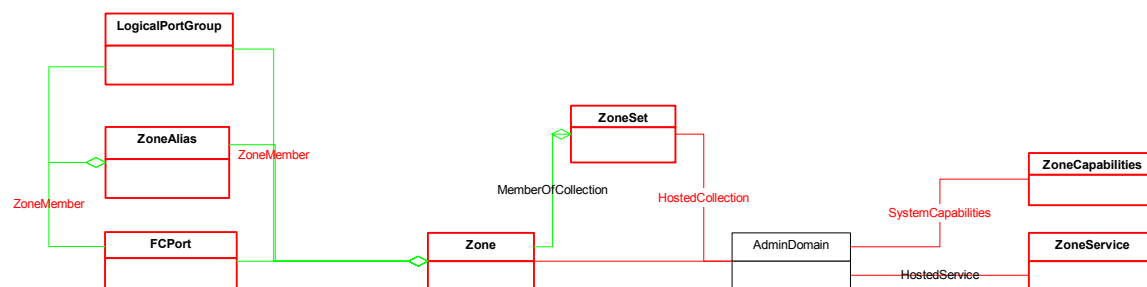


Figure 22: Zoning Instance Diagram

3.3.2.3.4 Client Considerations

The default zoning behavior of a fabric without an active `ZoneSet` is undefined.

3.3.2.3.5 Agent Considerations

The zoning component should allow the creation of FCPorts that are non-existent. This allows the client to create **ZoneMembers** that can be zoned in the future. Also there is the case where a FCPort goes offline and is not discoverable but will later appear when it goes online. The agent will provide a minimal set of properties necessary to identify this "undiscovered" FCPort but generally cannot populate the majority of the properties. The FCPort is owned by the fabric represented by **AdminDomain** and when "discovered" is then associated to the **ComputerSystem** that really contains it. This is often called the case of Parentally- Challenged devices.

3.3.2.3.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

- None

- Modification

```
SELECT * FROM InstModification WHERE SourceInstance ISA CIM_ZoneSet AND
      SourceInstance.Active <> PreviousInstance.Active
```

- Alert

```
Select * From AlertIndication
```

- Optional

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_Zone
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_Zone
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ZoneSet
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ZoneSet
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ZoneAlias
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ZoneAlias
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_ZoneMember
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_ZoneMember
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA
      CIM_MemberOfCollection AND SourceInstance.Member == CIM_Zone
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA
      CIM_MemberOfCollection AND SourceInstance.Member == CIM_Zone
```

3.3.2.3.7 Required Classes

Class	Req	Notes
AdminDomain (p. 214)	Y	
ElementCapabilities (p. 224)		
FCPort (p. 224)	Y	
HostedCollection (p. 231)		
HostedService (p. 232)		
LogicalPortGroup (p. 239)	N	
MemberOfCollection (p. 240)	Y	
SystemDevice (p. 288)	N	for parentally challenged FCPorts
Zone (p. 289)	Y	
ZoneAlias (p. 290)	N	
ZoneCapabilities (p. 290)	Y	
ZoneMember (p. 291)	Y	
ZoneService (p. 291)	Y	
ZoneSet (p. 292)	Y	

Table 3: Zoning Required Classes

3.3.2.4 Routers

3.3.2.4.1 Description

A Router is a device that translates between different types of SCSI buses. The instance diagram shows a system with a parallel SCSI buss and Fibre Channel buss. Devices on the parallel bus are served to the Fibre Channel bus without changing the characteristics of the device.

3.3.2.4.2 Schema Diagram

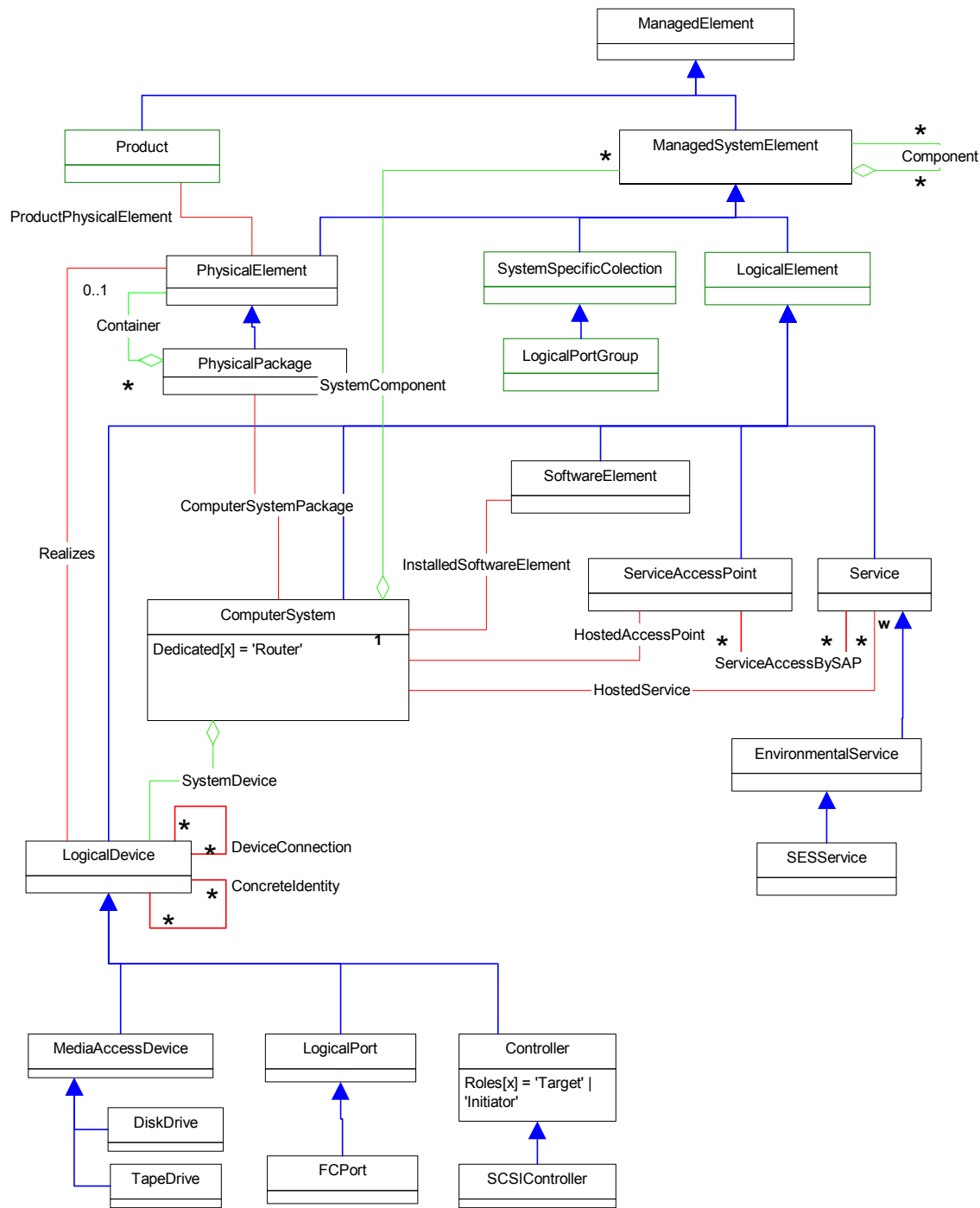


Figure 23: Router Schema Diagram

3.3.2.4.3 Instance Diagram

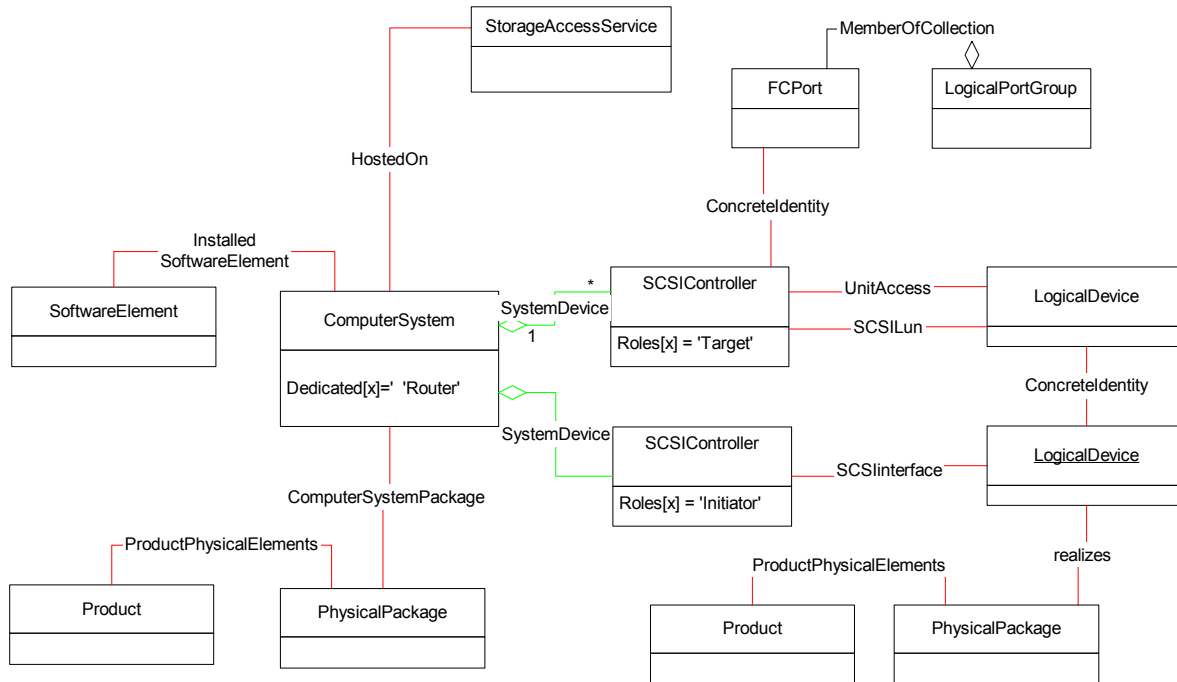


Figure 24: Router Instance Diagram

3.3.2.4.4 Client Considerations

3.3.2.4.4.1 Basic Design

The router model consists of 6 major groups of classes (Core, Physical, Software, SCSI buses, source / exported devices).

The **ComputerSystem** class is the core of the model. It is identified as a router by the dedicated attribute being set to “Router”. The **PhysicalPackage** class and **Product** class represent the physical aspects of the router and served devices. These classes contain attributes that can be used to identify the hardware. This information includes serial number, model number, and vendor name.

The **SoftwareElement** class represents the product’s firmware or vendor specific utilities that are running on the router. This class should be sub-classed for each utility.

The **SCSIController** class and optionally the **FCPort** class represent the SCSI buses that are part of the router. The **SCSIController** class near the bottom of the instance diagram is the parallel SCSI side of the router. Note that it doesn’t have an association to a **FCPort** class. It has **SCSIInterface** associations to the devices on the bus. The SCSI addresses of the devices are stored in the association.

The **SCSIController** class near to top of the instance diagram has a **ConcreteIdentity** association to a **FCPort** class. This indicates the **FCPort** is a Fibre channel SCSI port. This FC bus connects to the SAN. This bus uses **SCSILUN** and **UnitAccess** associations to hold the address mapping and masking. The **SCSIController** manages these associations.

A **LogicalDevice** class represents the device on the back end bus. This class has a **Realizes** association to a **PhysicalPackage** class to identify the hardware. The class uses a **ConcreteIdentity** association to a second instance of **LogicalDevice**. This class represents the device as seen by the front end port.

3.3.2.4.5 Agent Consideration

No agent considerations are defined at this time.

3.3.2.4.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort
```

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA  
CIM_ComputerSystem
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA  
CIM_ComputerSystem
```

- Modification

```
SELECT * FROM InstModification WHERE SourceInstance ISA CIM_FCPort AND  
SourceInstance.OperationalStatus[0] <>  
PreviousInstance.OperationalStatus[0]
```

```
SELECT * FROM InstModification WHERE SourceInstance ISA  
CIM_ComputerSystem AND SourceInstance.OperationalStatus[0] <>  
PreviousInstance.OperationalStatus[0]
```

- Optional

- Instance Indications

- None

3.3.2.4.7 Required Classes

Class	Req	Notes
ComputerSystem (p. 217)	Y	
	I	
ComputerSystemPackage (p. 218)		
FCPort (p. 224)	Y	
HostedCollection (P. 231)	Y	LogicalPortGroup to ComputerSystem
HostedService (p. 232)	Y	
InstalledSoftwareElement (p. 232)	Y	
LogicalDevice (p. 237)	Y	
LogicalPortGroup (p. 239)	Y	
MemberOfCollection (p. 240)	Y	into LogicalPortGroup
StorageAccessService (p. 258)	Y	
PhysicalPackage (p. 243)	N	
Product (p. 245)	N	
UnitAccess (p. 289)	N	
SCSIController (p. 249)	Y	
SCSIInterface (p. 250)	Y	
SCSILUN (p. 251)	Y	
SoftwareElement (p. 253)	N	
SystemDevice (p. 288)	Y	

Table 4: Router Required Classes

3.3.2.5 Extender

3.3.2.5.1 Description

A **FC Extender** is a device that translates Fibre channel communication to be transferred over difference media. Fibre channel extenders are used in pairs. This model shows an extender that uses ATM to extend a Fibre channel bus.

3.3.2.5.2 Schema Diagram

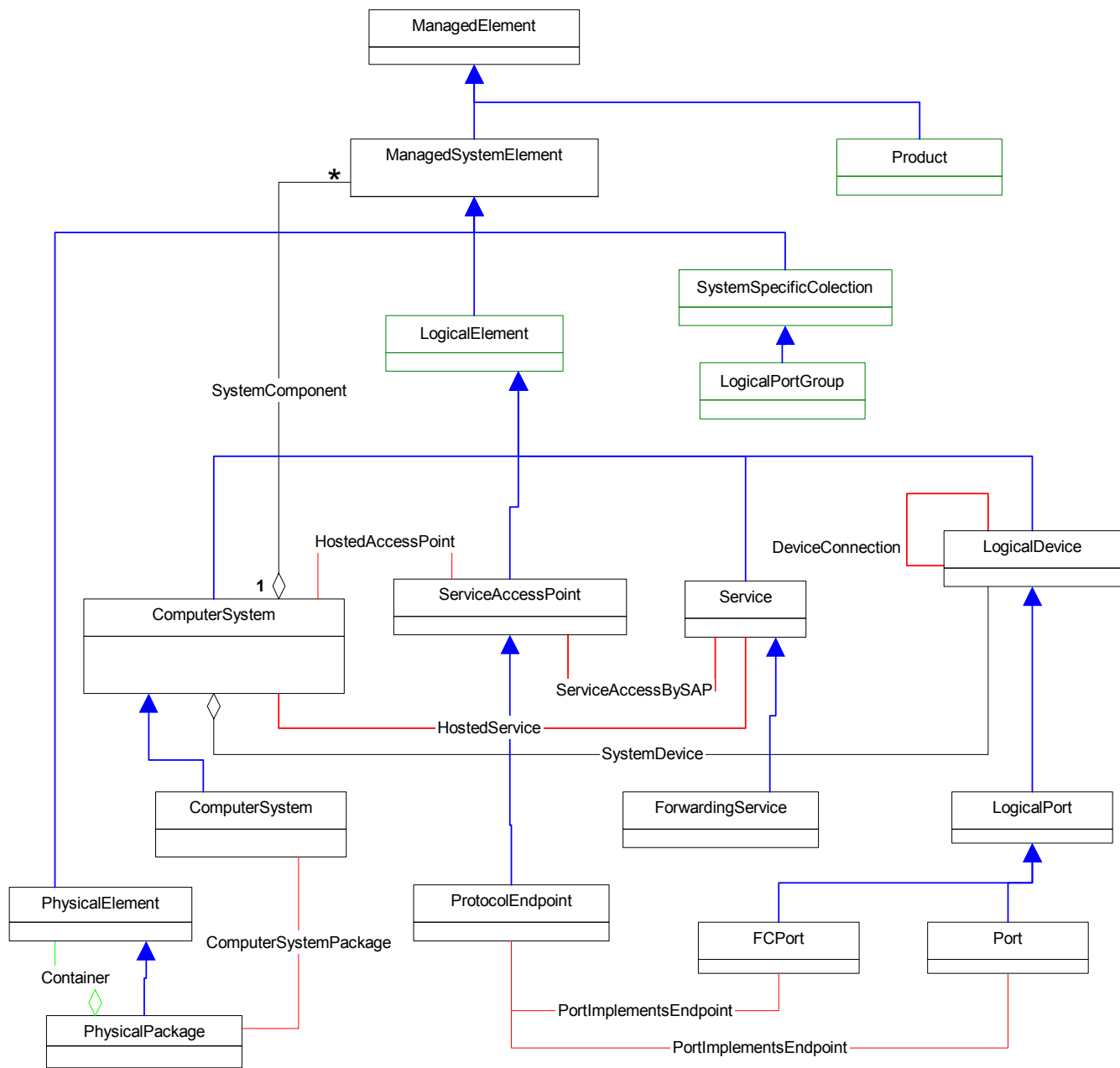


Figure 25: Extender Schema Diagram

3.3.2.5.3 Instance Diagram

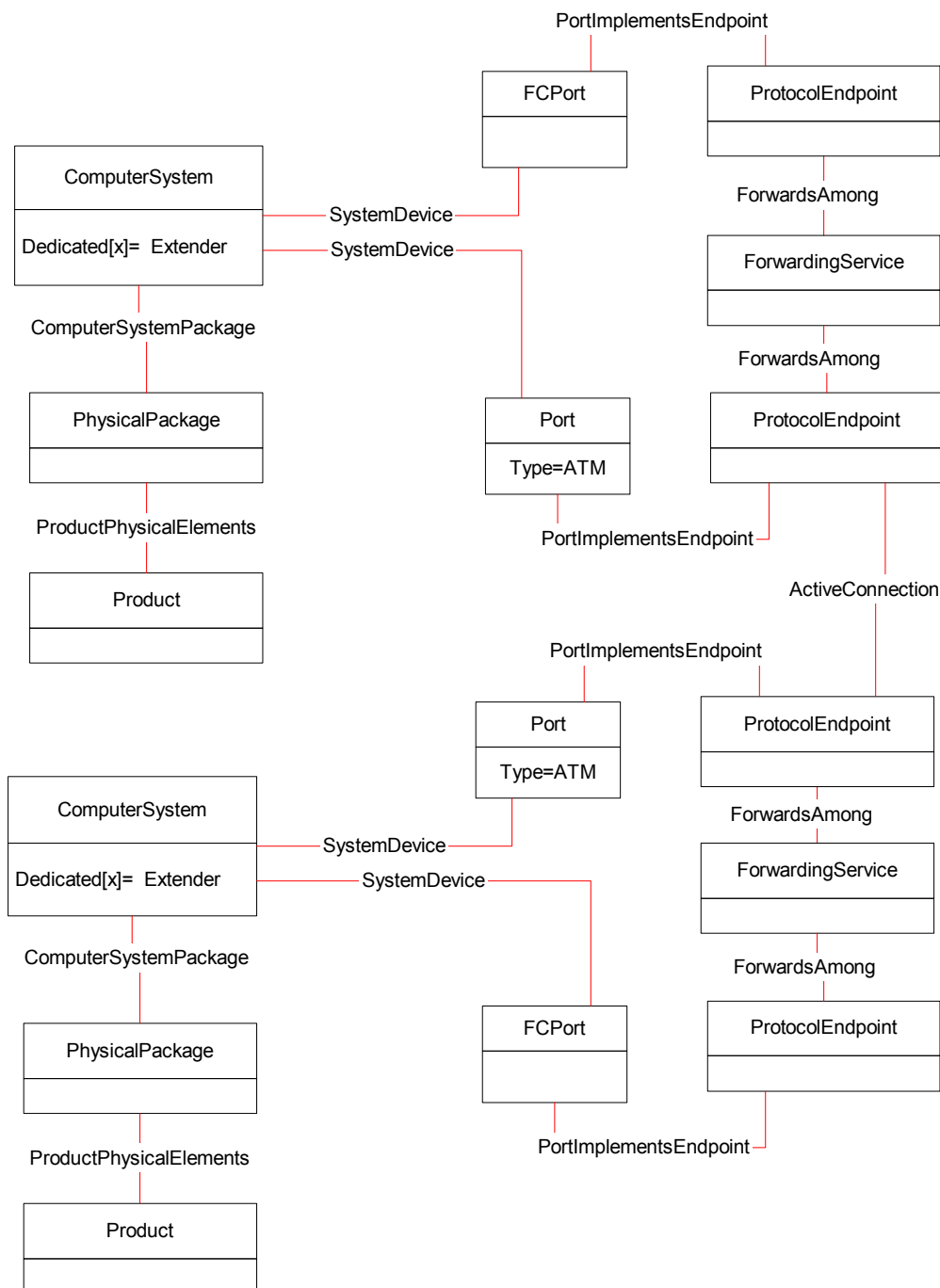


Figure 26: Extender Instance Diagram

3.3.2.5.4 Client Considerations

3.3.2.5.4.1 Basic Design

The Extender model consists of 3 major groups of classes (Core, Port, and Physical).

The **ComputerSystem** class is the core of the model. It is identified as an Extender by the dedicated attribute being set to Extender. The system has one service, the **ForwardingService** that represents the function of the extender and links the ports together.

The model contains two port classes **FCPort** and **Port** (type=ATM). The **FCPort** class represents the connection of the extender to the 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 **Port** class (type=ATM) represents the ATM link between extenders. This port may also have different types for different extension bus types.

The **PhysicalPackage** class and **Product** class represent the physical aspects of the extender. These classes contain attributes that can be used to identify the hardware system. This information includes serial number, model number, and vendor name.

3.3.2.5.5 Agent Considerations

None.

3.3.2.5.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_FCPort

SELECT * FROM CIM_InstDeletion
        WHERE SourceInstance ISA CIM_FCPort

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_ComputerSystem

SELECT * FROM CIM_InstDeletion
        WHERE SourceInstance ISA CIM_ComputerSystem

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_ActiveConnection

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_ActiveConnection

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_Port
        AND SourceInstance.LinkTechnology == "ATM"

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_Port
        AND SourceInstance.LinkTechnology == "ATM"
```

- Modification

```
SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_FCPort
        AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]

SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_ComputerSystem
        AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]
```

```

SELECT * FROM InstModification
WHERE SourceInstance ISA CIM_Port
AND SourceInstance.OperationalStatus[0] <>
PreviousInstance.OperationalStatus[0]
AND SourceInstance.LinkTechnology == "ATM"

```

- Optional
 - None

3.3.2.5.7 Required Classes

Class	Req	Notes
ComputerSystem (p. 217)	Y	
ComputerSystemPackage (p. 218)	N	
FCPort (p. 224)	Y	
ForwardingService (p. 230)	Y	
ForwardsAmong (p. 231)	Y	
HostedCollection (P. 231)	Y	LogicalPortGroup to ComputerSystem
HostedService (p. 232)	Y	
InstalledSoftwareElement (p. 232)	N	
LogicalPortGroup (p. 239)	Y	
MemberOfCollection (p. 240)	Y	into LogicalPortGroup
PhysicalPackage (p. 243)	N	
PortImplementsEndpoint (p. 245)	Y	
Product (p. 245)	N	
ProtocolEndpoint (p. 246)	Y	
SoftwareElement (p. 253)	N	
SystemDevice (p. 288)	Y	

Figure 27: Required Classes for Extender

3.3.3 Hosts

3.3.3.1 Host Bus Adapters

3.3.3.1.1 Description

A Fibre Channel adapter used in a host system is called a Host Bus Adapter (HBA). An HBA is a physical device which contains one or more Fibre Channel ports. A single system contains one or more HBAs.

3.3.3.1.1.1

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 (typically Card) through the Realizes association. If the HBA has logical operations that apply to the HBA and not to an individual port, then the PortController can be instantiated. The PortController is associated to the ComputerSystem through the SystemDevice association and associated to the ports through the ControlledBy association.

3.3.3.1.2 Schema Diagram

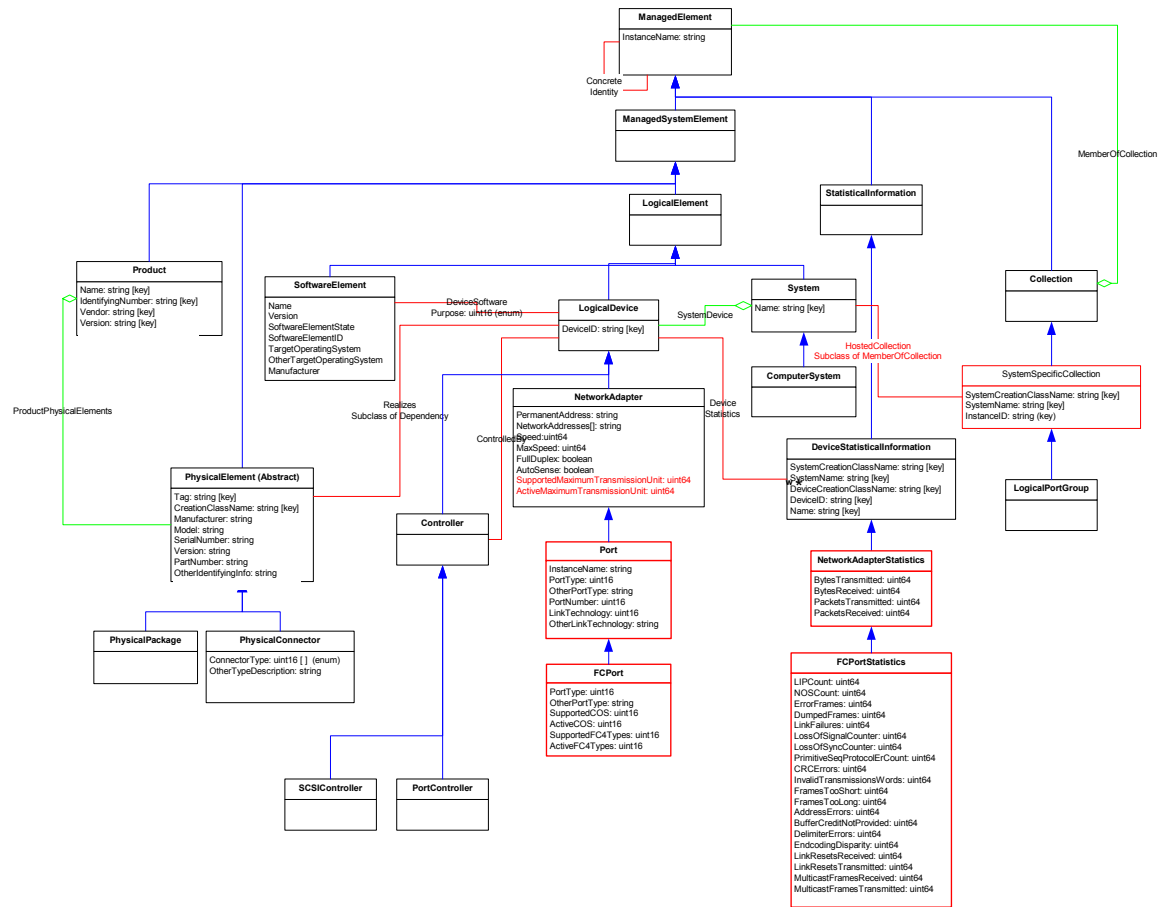


Figure 28: HBA Schema Diagram

3.3.3.1.3 Instance Diagram

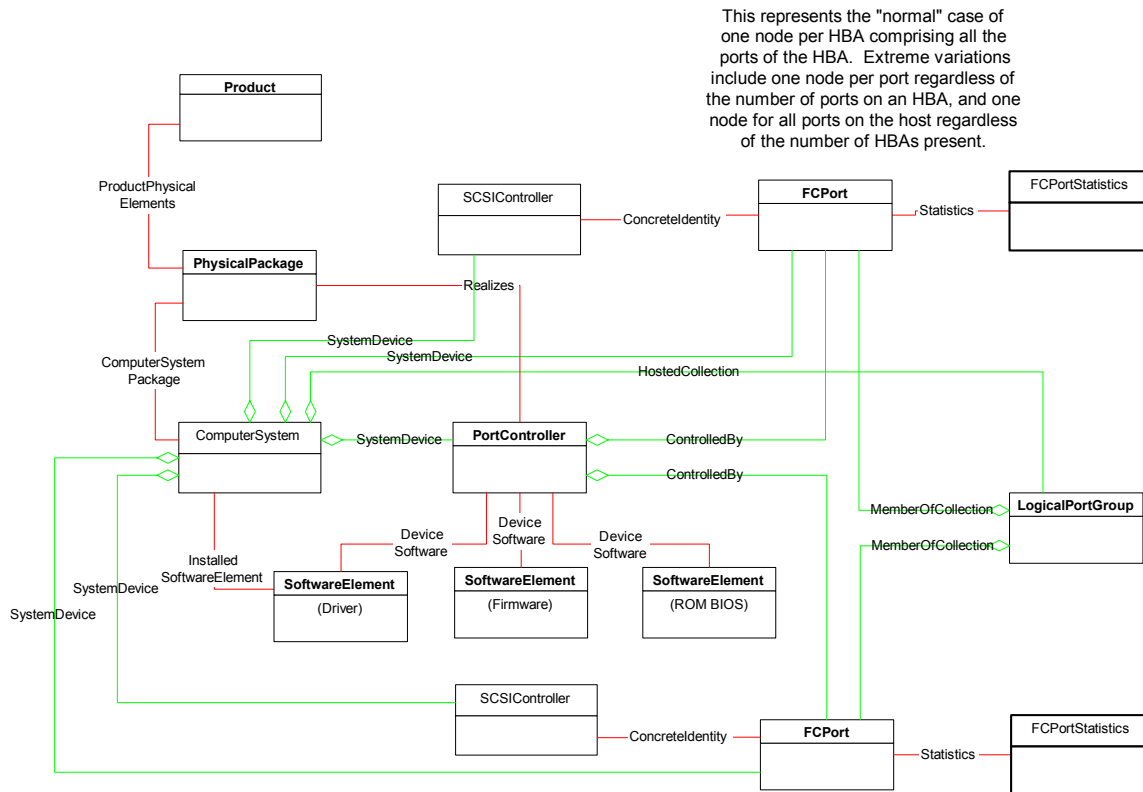


Figure 29: HBA Instance Diagram

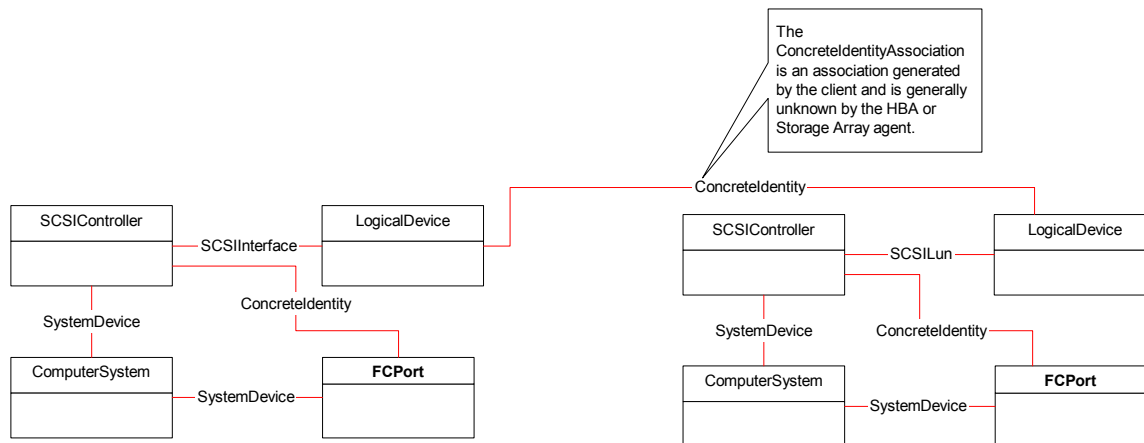


Figure 30: HBA Binding Instance Diagram

3.3.3.1.4 Client Considerations

The client does need to consider that there could be multiple Bluefin agents providing instances unrelated to what maybe provided on the Host system, and may be unrelated to other Bluefin agents on the host.

3.3.3.1.5 Agent Considerations

None.

3.3.3.1.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort
```

- Modification

```
SELECT * FROM InstModification
WHERE SourceInstance ISA CIM_FCPort
AND SourceInstance.OperationalStatus[0] <>
PreviousInstance.OperationalStatus[0]
```

- Optional

- None

3.3.3.1.7 Required Classes

Class	Req	Notes
ComputerSystem (p. 217)	Y	A Host with FCPorts
ConcretelIdentity (p. 219)	I	FCPort to SCSIController if any
DeviceSoftware (p. 223)	I	Software or firmware or BootROM if any to the HBA or port it supports
FCPort (p. 224)	Y	
FCPortStatistics (p. 228)	N	FC-specific port statistics
HostedCollection (p. 231)	Y	LogicalPortGroup (Node) to ComputerSystem
InstalledSoftwareElement (p. 232)	I	HBA software or firmware or BootROM if any to the host system on which it is installed if any
LogicalPortGroup (p. 239)	I	An FC Node
MemberOfCollection (p. 240)	I	FCPort to LogicalPortGroup (Node)
PhysicalConnector(see p. 241)	N	An FC Connector on an HBA card
PhysicalPackage (see p. 243)	Y	An HBA card
PortController	I	Aggregates HBA FCPorts if multiport HBA
Product (see p. 245)	Y	The physical objects that compose the HBA
ProductPhysicalElements (see p. 246)	Y	Packages and Connectors to Products
Realizes(p. 247)	Y	FCPort or PortController to its PhysicalPackage FCPort to its PhysicalConnector
SCSIController (p. 249)	I	An HBA FCPort if it is FCP-capable
SoftwareElement (see p. 253)	I	HBA-relevant software or firmware or BootROM
SystemDevice (p. 288)	Y	FCPort to ComputerSystem SCSIController if any to ComputerSystem PortController if any to ComputerSystem

Table 5: Required Classes for HBA

3.3.3.2 Host Discovered Resources

3.3.3.2.1 Description

Among the primary functions of a Fibre Channel Host Bus Adapter (HBA) and its supporting software is discovery of SAN resources and presentation of those resources to the Host Operating System. A description of the results of these functions is useful for some aspects of SAN management:

- Determination of discrepancies between resources discovered by HBAs and the resources provided by other SAN elements is valuable for diagnostics

- The information discovered by HBAs can provide information about SAN resources not themselves supported by Bluefin agents
- In SANs that lack an agent for the Fabric profile, e.g., Private Arbitrated Loops and FC Direct Attach, a client can construct a view of the fabric by integrating the discovered resources from any available hosts
- Discovered resource information includes the identification of SAN resources as they are presented to the Host OS

The Host Discovered Resource agent uses the SNIA HBA API to create a generic model of the logical SAN and attached storage. HBA API is included as an appendix to the FC-MI specification – see www.t11.org. This agent models elements also exposed by HBA, storage, and switch agents. A client can use durable names to equate objects from different agents.

This profile is restricted to FCP (SCSI over FibreChannel) discovery. A similar approach can be used for other protocols (such as IP over FC), but this is not described in this profile. Note that no physical objects are represented by this profile. Since the objects are here are discovered remotely through an HBA, only logical aspects are available. In general, the objects exposed by this agent duplicate those exposed by canonical HBA, storage, or switch agents which provide the physical model.

The Host SAN Resources are independently instantiated for each HBA FCPort on a host. They include its FC Node, discovered (remote) FCP ports, and SCSI Targets.

A **LogicalPortGroup** represents an FC Node; **MemberOfCollection** associates each FCPort to the **LogicalPortGroup**.

The discovering FCPort and each discovered FCPort are associated by **DeviceSAPImplementation** to a **ProtocolEndpoint** representing its FCP support (**ProtocolType**=other, **OtherProtocolType**=”SCSIOverFC”). An instance of **LogicalNetwork** is created to aggregate the FCP **ProtocolEndpoint** for the discovering FCPort and all its discovered FCPorts.

SCSI Targets are modeled by FCPorts with a **ConcreteIdentity** to a **SCSIController** which in turn has at least one **SCSILUN** association to a **LogicalDevice**. The **SCSIController** / FCPort combination represents a SCSI Port with Target capability. The **LogicalDevice** in such associations represent SCSI Target Logical Units.

SCSI Initiators (HBA ports) are also modeled with a **SCSIController** / FCPort combination. Initiator **SCSIController** have **SCSIInterface** associations to logical units (**LogicalDevice** subclasses) that are mapped to the HBA host.

Target Mappings are a pairing of an OS SCSI ID and an FCPID for a Logical Unit that represents a Logical Unit as presented to a Host Operating System. They are modeled by a **LogicalDevice** with both a **SCSILUN** and a **SCSIInterface** association. The OS SCSI ID is represented as attributes of the **SCSIInterface** association. The paired FCPID is derived from the attributes of the **SCSILUN** association and the FCPort to which it (indirectly) associates.

CIM requires that all **LogicalDevices** (including **SCSIController** and **FCPort**) be weak to a **System** via a **SystemDevice** aggregation. It does not in general have means to discover the containing systems for discovered FCPorts, so for each **LogicalNetwork**, this profile provides an **AdminDomain** to aggregate discovered objects that must be weak to a **System**.

3.3.3.2.2 Schema Diagram

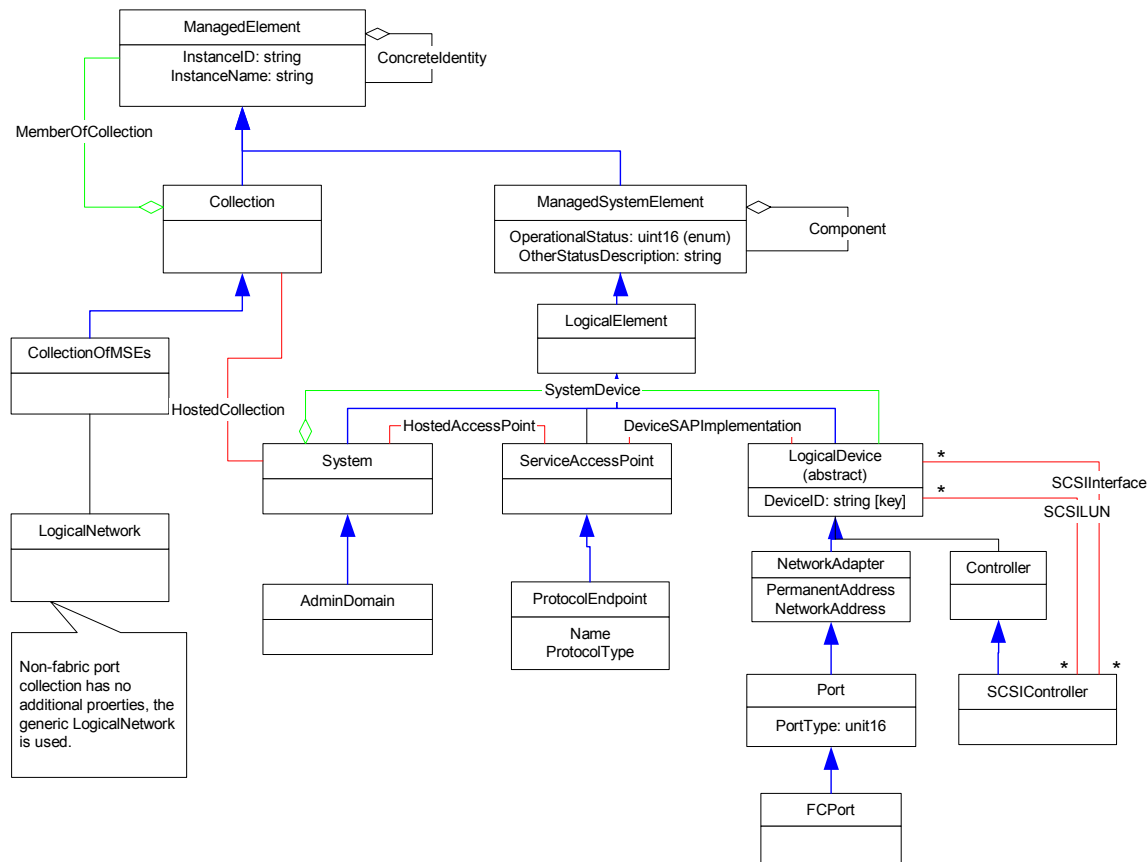


Figure 31 Host Discovered Resources Schema Diagram

3.3.3.2.3 Instance Diagram

The first instance diagram depicts two logical networks – each contains an HBA and one of two FCPorts in a multi-port array. Three volumes are depicted; note that volume 2 has no SCSIInterface associations – indicating that it is not mapped to the OS hosting this agent. Due to the complexity of this example, some associations are missing and some are unlabelled.

Note that all the depicted objects are instantiated by the Discovered Resources agent. The dashed rectangles represent groups of objects that duplicate objects from other agents. For example, an HBA agent also exposes all the objects in the “HBA1 Objects” rectangle. A client can use durable names to “stitch together” these duplicates.

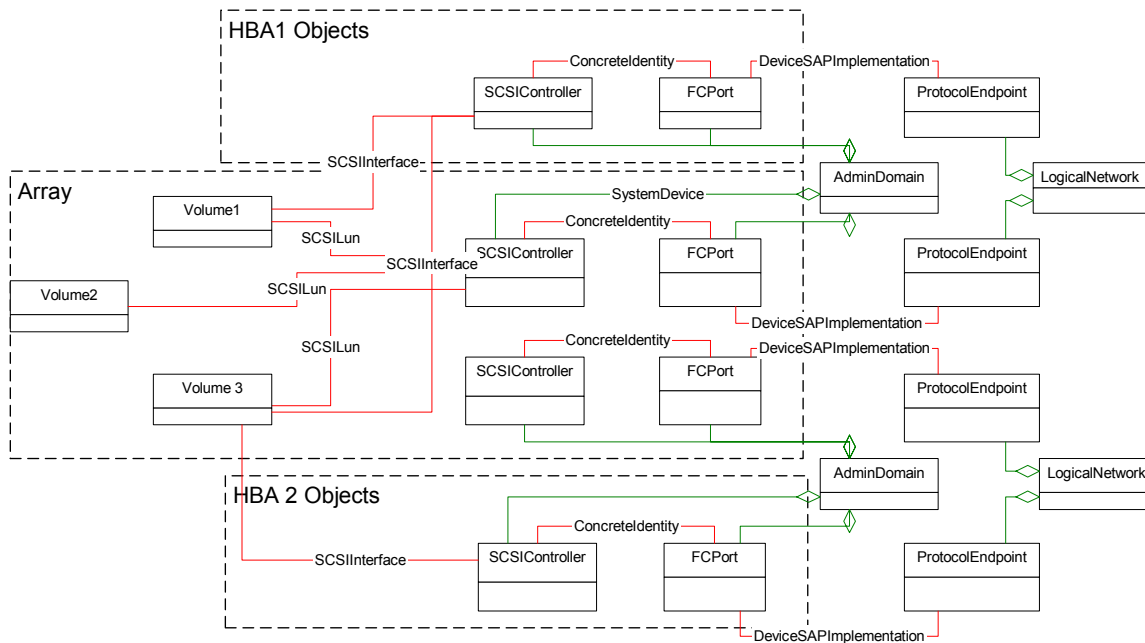


Figure 32 Host Discovered Objects Instance Diagram

The second diagram consists of just a single HBA port, single array port and single volume. All associations are included and labeled.

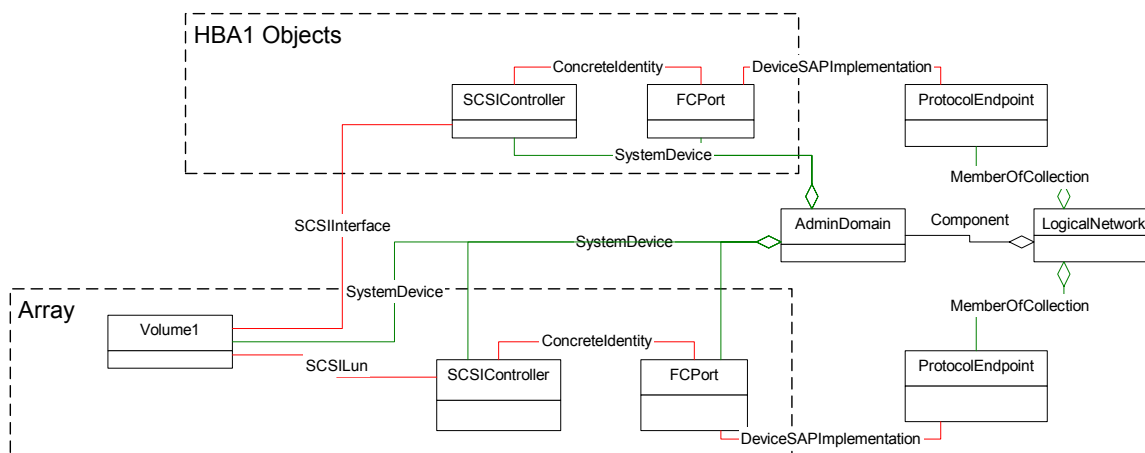


Figure 33 Host Discovered Objects Instance Diagram

3.3.3.2.4 Client Considerations

In typical configurations, the ports and logical units provided by this model will duplicate those found in storage (array or tale library) and switch agents. Although this profile has information about storage systems and the storage network, the information is not complete. For example, this agent may model several small arrays that are actually separate targets (ports) on a single array (or virtual targets resulting from LUN masking/mapping). Where available, the client should use information from an array agent to get a complete model for the array. Similarly, the logical networks modeled by this agent may actually be zones in fabrics; the client should use information from switch agents to get a complete fabric model.

In a non-fabric storage network (Loop or Direct Attached Storage) there is likely to be no agent for the Fabric profile. A client may derive similar information by integrating the models presented by Host Discovered Resource agents running on multiple hosts.

Since the storage system topology cannot be accurately inferred, storage system objects are associated to an **AdminDomain**, a “virtual” **ComputerSystem** that represents the collection of objects in the **LogicalNetwork**. In particular, **SystemDevice** associates all logical units to the **AdminDomain** and the **AdminDomain Name** property is used as the **SystemName** property for all **LogicalDevice** subclasses.

A client associates objects between profiles using durable identifiers (as described in other profiles). If no storage system agent is available, the model from this agent may suffice, but some details may not be available.

Discovered storage system resources can be partitioned into two groups, objects related to a port (**FCPort**, **SCSIController**, and **ProtocolEndpoint**) and logical units (**StorageVolume**, **TapeDrive** and the **SCSILUN** association). Discovery of logical units can be resource intensive and disruptive to the host system (consider arrays with thousands of logical units). The agent should not allocate resources on logical unit discovery unless requested by a client; this request is communicated by following the **SCSILUN** associations from a **SCSIController** to its logical units. The client algorithm for Discovered Resources would be

```
Enumerate AdminDomains
```

```
Consider just those with "HBA Discovered Resources" in Roles[]
```

```
Follow the Component association to the LogicalNetwork
```

```
Foreach MemberOfCollection association, follow it to a ProtocolEndpoint
```

```
    Follow the DeviceSAPImplementation association to a FCPort
```

```
    // This gives the client a list of PortWWNs on the network.
```

```
    // If these all map to PortWWNs from array/storage agents, the
```

```
    // client may opt to stop probing.
```

```
If the client wishes to also discover LUNs
```

```
    Follow the ConcreteAssociation to a SCSIController
```

```
    Follow each SCSILUN association to logical units
```

```
    If no SCSILUN associations are found,
```

```
        This is an initiator (another HBA port)
```

```
    Else
```

```
        Get Instance of LogicalDevices from the SCSILUN association
```

This algorithm allows the agent to dedicate resources to LUN discovery only when requested by a client. Note that if LUNs are not discovered, the model will not include **SCSILUN** or **SCSIInterface** associations; the client determines target/initiator roles and host/storage system topology by matching durable names with **FCPorts** in HBA and storage profiles.

A client may discover more complex multipathing by integrating the HBA profiles and Host Discovered Resources profiles from their respective agents. Here are some examples: If the client found two **FCPorts** which were **SystemDevices** of the same **ComputerSystem**, and found among the Discovered Resources of both, the same **FCPort** that was associated by **ConcreteIdentity** to a **SCSIController** in turn associated by **SCSILun** to a **LogicalDevice**, the

client would have demonstrated that the host represented by the **ComputerSystem** had multipathing via two HBA ports to a single Target port. If it found two **FCPorts** which were **SystemDevices** of the same **ComputerSystem**, and found among the Discovered Resources of each a different **FCPort** which was associated by **ConcreteIdentity** to a **SCSIController** in turn associated by **SCSILun** with the same **LogicalDevice**, the client would have demonstrated that the host represented by the **ComputerSystem** had multipathing via two HBA ports and two Target ports to a single Target Logical Unit.

3.3.3.2.5 Agent Considerations

The Host SAN Resources profile is based on information available through the HBA API Phase 1 discovery interfaces. Its implementation therefore may be HBA vendor independent.

The **AdminDomain** for the **LogicalNetwork** has no underlying identification. The agent should set the **AdminDomain Name** property to the Port WWN of the discovering HBA port and the **NameFormat** property to "FC". This allows a client to determine which port was used to discover the particular **LogicalNetwork**. The **AdminDomain Roles[]** array must contain a "HBA Discovered Resources" entry. This allows the client to determine which **AdminDomains** are related to this profile.

The agent must set **ProtocolEndpoint** properties **ProtocolType=other** and **OtherProtocolType="FC-4=20"**. The **ProtocolEndpoints** for the local HBA port and its attached remote ports are all aggregated into a **LogicalNetwork**. The agent must set **LogicalNetwork** properties **NetworkType=other**, **OtherTypeDescription="FC-4=20"**, **Name=discovering FCPort WWN with ":FC-4=20" appended**, and **NameFormat="Discovering FCPort WWN with :FC-4=20 appended"**.

The agent must separate LUN discovery so that a client can limit resources as described in the algorithm under Client Considerations above. In particular, the agent should not issue SCSI "Report LUNs", "Inquiry", or "Read Capacity" unless a client follows **SCSILun** associations.

If the client does ask for **SCSILun** associations, **LogicalDevice** subclasses are instantiated for all the SCSI Logical Units reported by the "Report LUNs" command, then SCSI Inquiry. The agent chooses the **LogicalDevice** subclass based in the SCSI Inquiry device type:

	LogicalDevice Subclass
DirectAccess SCSI type	StorageVolume
SequentialAccess	TapeDrive
All others	LogicalDevice

Table 6: SCSI Device Type Mapping

Note that this agent cannot determine whether a Direct Access device is a physical disk or a virtualized volume; for consistency the agent will always instantiate a **StorageVolume**. Other than disks and tapes, there are many vendor-specific implementations, so a generic **LogicalDevice** is instantiated.

SCSI Inquiry VPD commands are issued to get **LogicalDevice** durable names as described in the array and tape library profiles. These names can be used to identify multi-path configurations; this is modeled with multiple **SCSILun** associations from **FCPort/SCSIController** pairs to a common **LogicalDevice**.

If the same logical unit is discovered on multiple **LogicalNetworks**, the agent should create a single instance and use **SCSILun** associations to **SCSIControllers**. Logical unit objects may have **SCSILun** associations to **SCSIControllers** that are associated to different **AdminDomains** (because they are in different **LogicalNetworks**). But a logical unit object must be associated to a single **AdminDomain**. The agent should pick one of the **AdminDomains** and use it for **SystemDevice** associations and determination of the **SystemName** property of the logical unit objects.

3.3.3.2.6 Indications

This agent cannot accurately report health events; so all `AlertIndications` are optional. But configuration changes should be reported with `InstIndications`.

Mandatory

- `InstIndication`
 - Creation of `StorageVolumes` (similar for other logical units)


```
SELECT * FROM CIM_InstCreation
WHERE SourceInstance ISA CIM_StorageVolume
SELECT * from CIM_InstModification
WHERE SourceInstance ISA CIM_StorageVolume
```
 - Deletion of `StorageVolumes` (similar for other logical units)


```
SELECT * FROM CIM_InstDeletion
WHERE SourceInstance ISA CIM_StorageVolume
```
 - Creation, Deletion of ports


```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort
```
- `Alert` – none mandatory

3.3.3.2.7 Required Classes

Class	Req	Notes
<code>AdminDomain</code> (p. 214)	Y	The “virtual <code>ComputerSystem</code> ” for <code>LogicalDevice</code> <code>SystemName</code> and <code>SystemDevice</code> associations
<code>Component</code> (p. 217)	Y	<code>LogicalNetwork</code> to <code>AdminDomain</code>
<code>ConcretelDentity</code> (p. 219)	I	<code>FCPort</code> to <code>SCSIController</code> if any
<code>DeviceSAPImplementation</code> (p. 222)	Y	Associates <code>PortocolEndpoint</code> and <code>FCPort</code>
<code>FCPort</code> (p. 226)	Y	
<code>HostedCollection</code> (p. 231)	Y	<code>LogicalPortGroup</code> (Node) to <code>ComputerSystem</code>
<code>LogicalDevice</code> (p. 237)	I	In non-direct/sequential unit types are discovered
<code>LogicalNetwork</code> (p. 239)	Y	
<code>LogicalPortGroup</code> (p. 239)	N	An FC Node
<code>MemberOfCollection</code> (p. 240)	Y	<code>FCPort</code> to <code>LogicalPortGroup</code> (Node)
<code>MemberOfCollection</code> (p. 240)	Y	<code>ProtocolEndpoint</code> to <code>LogicalNetwork</code>
<code>ProtocolEndpoint</code> (p. 246)	Y	Network aspects of an FC Port
<code>SCSIController</code> (p. 249)	Y	SCSI aspects of an FC Port
<code>SCSIInterface</code> (p. 250)	I	Initiator <code>SCSIController</code> to <code>LogicalDevice</code> or subclass if any
<code>SCSILUN</code> (p. 251)	C	Target <code>SCSIController</code> to <code>LogicalDevice</code> or subclass if any
<code>StorageVolume</code> (p. 277)	I	If SCSI “Direct-access” unit types are discovered
<code>SystemDevice</code> (p. 288)	Y	Any <code>LogicalDevice</code> subclass to <code>ComputerSystem</code>
<code>TapeDrive</code> (p. 289)	I	If SCSI “Sequential-access” unit types are discovered

3.3.3.3 Management Appliance

3.3.3.3.1 Description

A **Management Appliance** is a computer system dedicated to running management software. In most cases the management software is accessed remotely through Web interfaces. This model is designed to allow for the discovery of the appliance and the applications running on it.

3.3.3.3.2 Schema Diagram

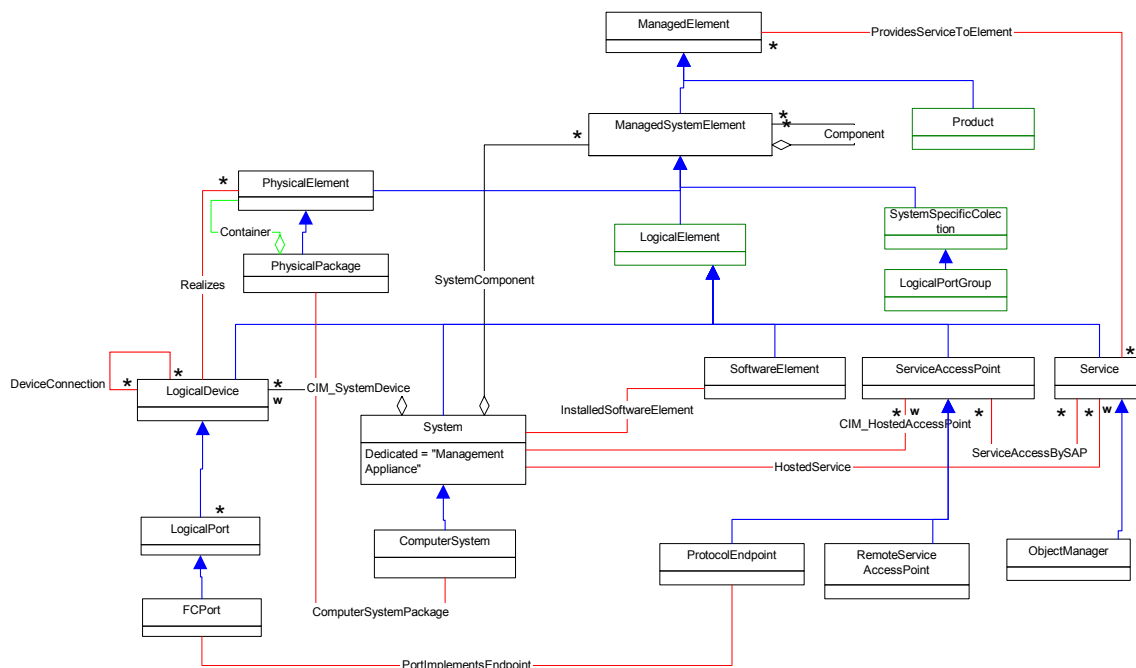


Figure 34: Management Appliance Schema Diagram

3.3.3.3.3 Instance Diagram

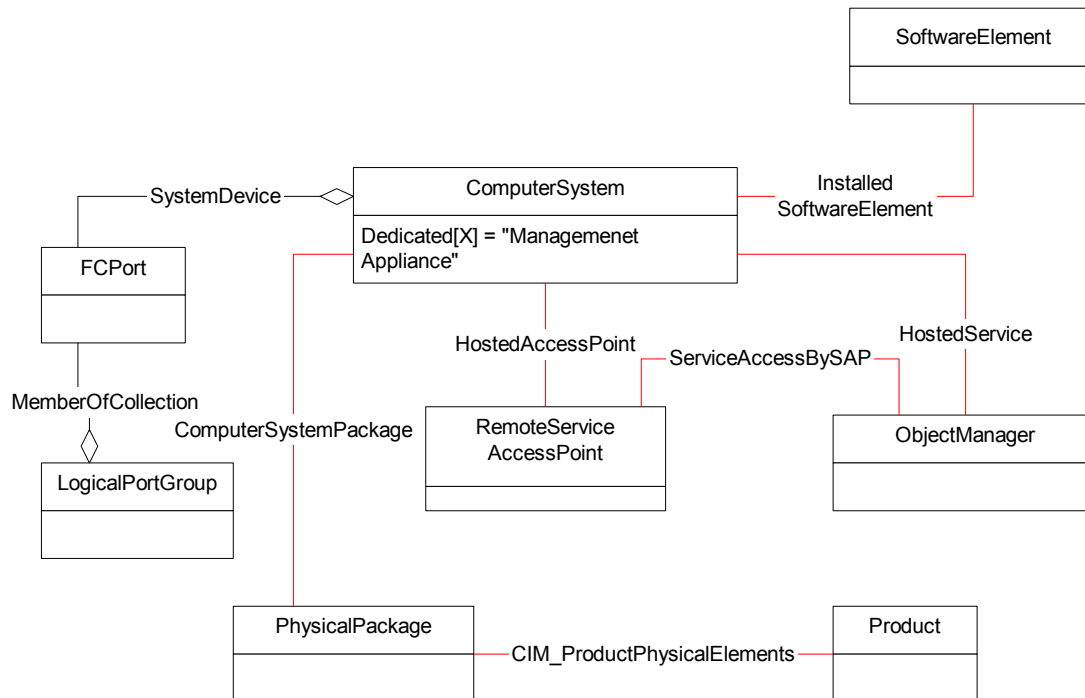


Figure 35: Management Appliance Instance Diagram

3.3.3.3.4 Client Considerations

3.3.3.3.4.1 Basic Design

The Management Appliance model consists of 4 major groups of classes (Core, Port, Physical, and Software).

The **ComputerSystem** class is the core of the model. It is identified as a Management Appliance by the **Dedicated** attribute being set to “Management Appliance”.

The **FCPort** class and **LogicalPortGroup** classes represents the connection of the Management Appliance to the SAN. The **FCPort** 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 **PhysicalPackage** class and **Product** class represent the physical aspects of the Management Appliance. These classes contain attributes that can be used to identify the hardware system. This information includes serial number, model number, and vendor name.

The **SoftwareElement** class represents the management utilities that are running on the appliance. The instance diagram shows the **SoftwareElement** class sub-classed to represent a CIMOM. The **ObjectManager** class is a subclass of **SoftwareElement** and represents the CIMOM. The **WebAccessPoint** class contains the URL to access the CIMOM.

3.3.3.3.4.2 Applications

The Client will be able to get a list of available applications by locating the **ComputerSystem** class and enumerating the group of **SoftwareElement** classes by traversing the **InstalledSoftwareElement** associations.

3.3.3.3.5 Agent Considerations

3.3.3.3.5.1 Applications

The purpose of the Management Appliance is to be a platform to run management applications on. Each of the applications running on the appliance should be modeled with both a **SoftwareElement** class and a **RemoteServiceAccessPoint**. The **SoftwareElement** describes the application including name, version, and other. This class is often sub-classed for each type of application. The **RemoteServiceAccessPoint** or **WebAccessPoint** class contains a network address or URL to access the application.

3.3.3.3.6 Indications

- Mandatory

- Instance Indications

- Create/Delete

```
SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_FCPort

SELECT * FROM CIM_InstDeletion
        WHERE SourceInstance ISA CIM_FCPort

SELECT * FROM CIM_InstCreation
        WHERE SourceInstance ISA CIM_ComputerSystem

SELECT * FROM CIM_InstDeletion
        WHERE SourceInstance ISA CIM_ComputerSystem
```

- Modification

```
SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_FCPort
        AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]

SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_ComputerSystem
        AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]
```

- Optional

- Instance Indications

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SoftwareElement

SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SoftwareElement
```

- Modification

```
SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_Service
        AND SourceInstance.OperationalStatus[0] <>
        PreviousInstance.OperationalStatus[0]

SELECT * FROM InstModification
        WHERE SourceInstance ISA CIM_SoftwareElement
        AND SourceInstance.SoftwareElementState <> PreviousInstance.
        SoftwareElementState
```

3.3.3.3.7 Required Classes

Class	Req	Notes
	Y	Between Switch and Port
ComputerSystem (p. 217)	Y	
ComputerSystemPackage (p. 218)	I	
FCPort (p. 224)	Y	
HostedAccessPoint (p. 231)	Y	
HostedCollection (P. 231)	Y	LogicalPortGroup to ComputerSystem
InstalledSoftwareElement (p. 232)	Y	
LogicalPortGroup (p. 239)	Y	
MemberOfCollection (p. 240)	Y	into LogicalPortGroup
ObjectManager (p. 241)	I	
PhysicalPackage (p. 243)	I	
Product (p. 245)	I	
RemoteServiceAccessPoint (p. 248)	I	
ServiceAccessBySAP (p. 252)	Y	
SoftwareElement (p. 253)	Y	one for each application
SystemDevice (p. 288)		

Table 7: Required Classes for Management Appliance

3.3.4 Storage Systems

3.3.4.1 Disk Arrays

3.3.4.1.1 Description

The Disk Array model accommodates a variety of disk storage implementations including RAID arrays, virtualization appliances, and JBOD arrays. The key classes are storage volumes (visible to consumers) and extents (internal to the array), FC ports and controllers.

3.3.4.1.2 Schema Diagram

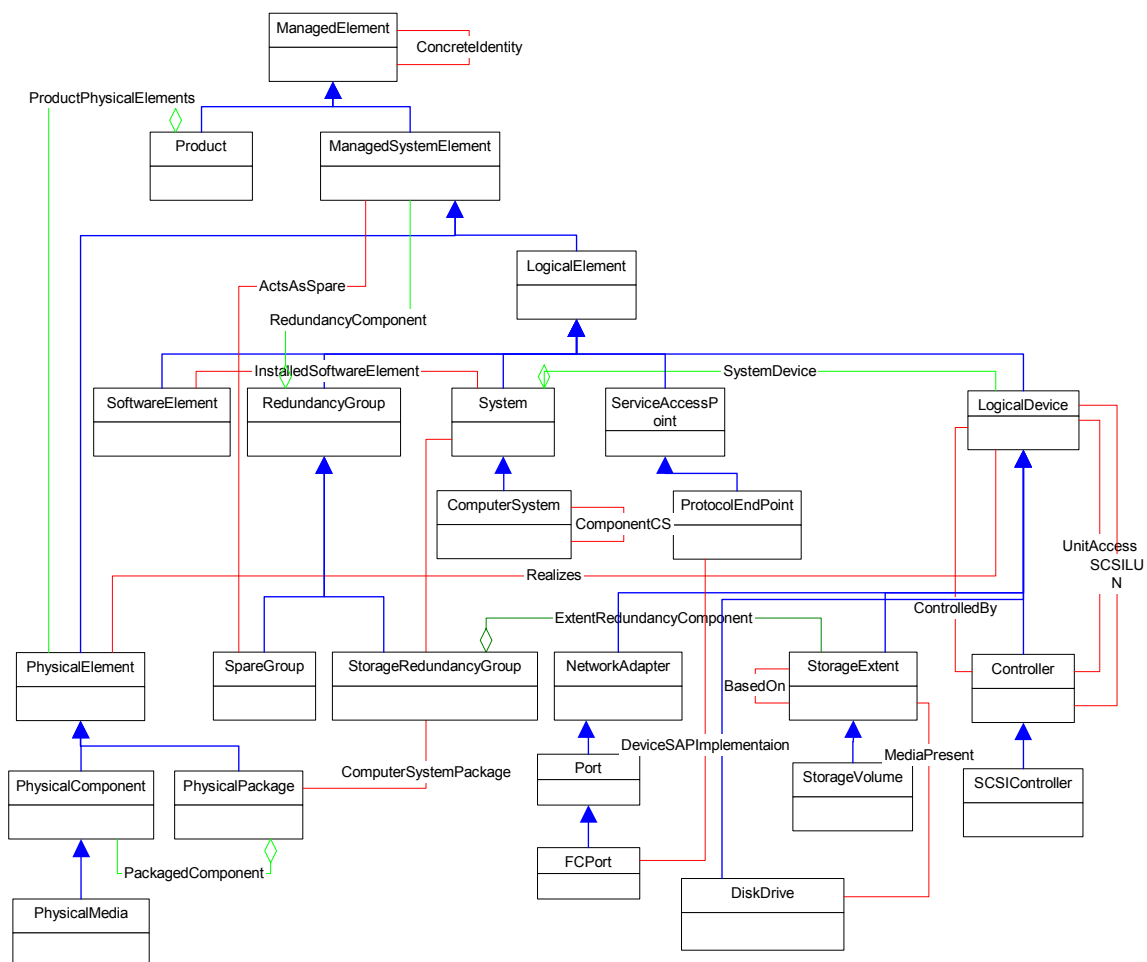


Figure 36: Disk Array Core Schema Diagram

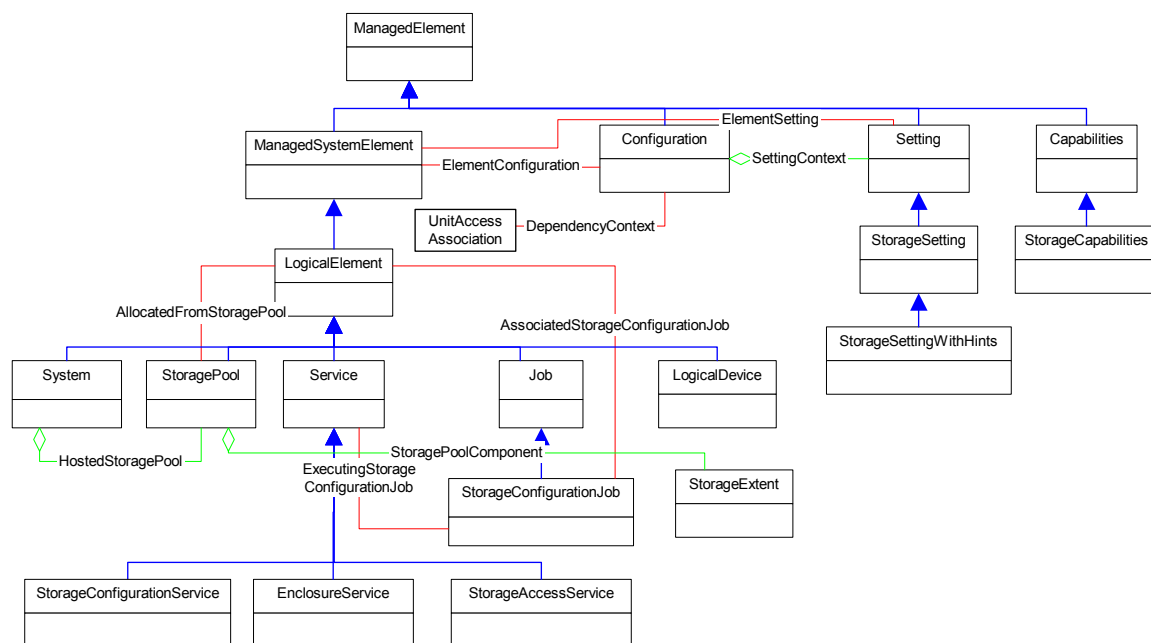


Figure 37: Disk Array Service Schema Diagram

3.3.4.1.3 Instance Diagrams

3.3.4.1.3.1 RAID Array Instance Diagram

The RAID array instance diagram applies to arrays with RAID virtualization and virtualization appliances. This instance diagram includes core array objects. Instance diagrams with **StorageConfigurationService** and **StorageAccessService** objects and details on certain array aspects are included in the client and agent considerations sections.

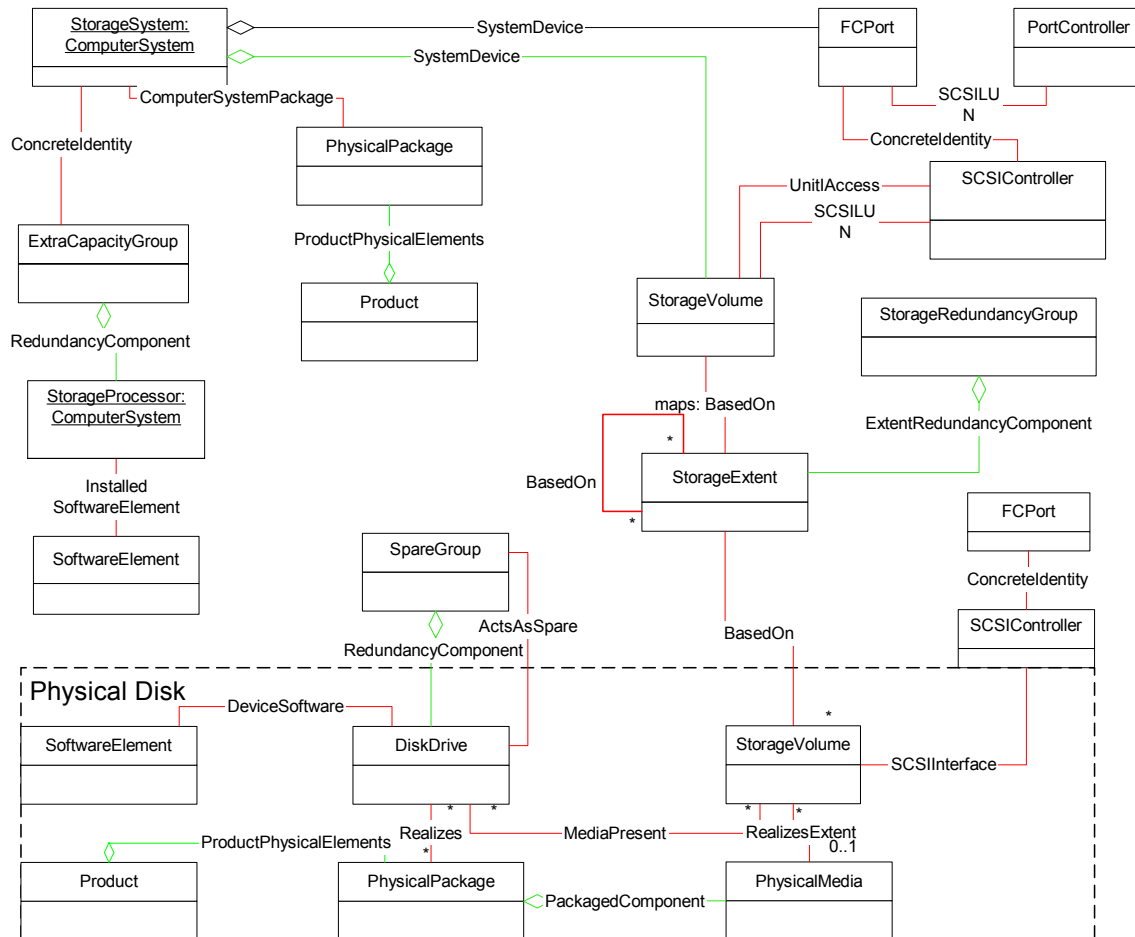


Figure 38: Disk Array Instance Diagram

3.3.4.1.3.2 JBOD Instance Diagram

A JBOD (Just a Bunch Of Disks) is a disk array without a RAID processor. This model represents JBODs that also include a limited management interface. This interface may use SCSI Enclosure Services (SES) as an interface or a proprietary interface. These interfaces may be in-band or out-of-band (TCP/IP based). In practice, the management capabilities vary with the implementations; this model provides clients with an interface to enumerate the attached disks and provides an optional **EnclosureService** for proprietary interfaces. Note that **EnclosureService** is optional; enumeration of devices should be done in the model itself. The service should be used for proprietary methods for diagnostics, on-line/off-line, etc.

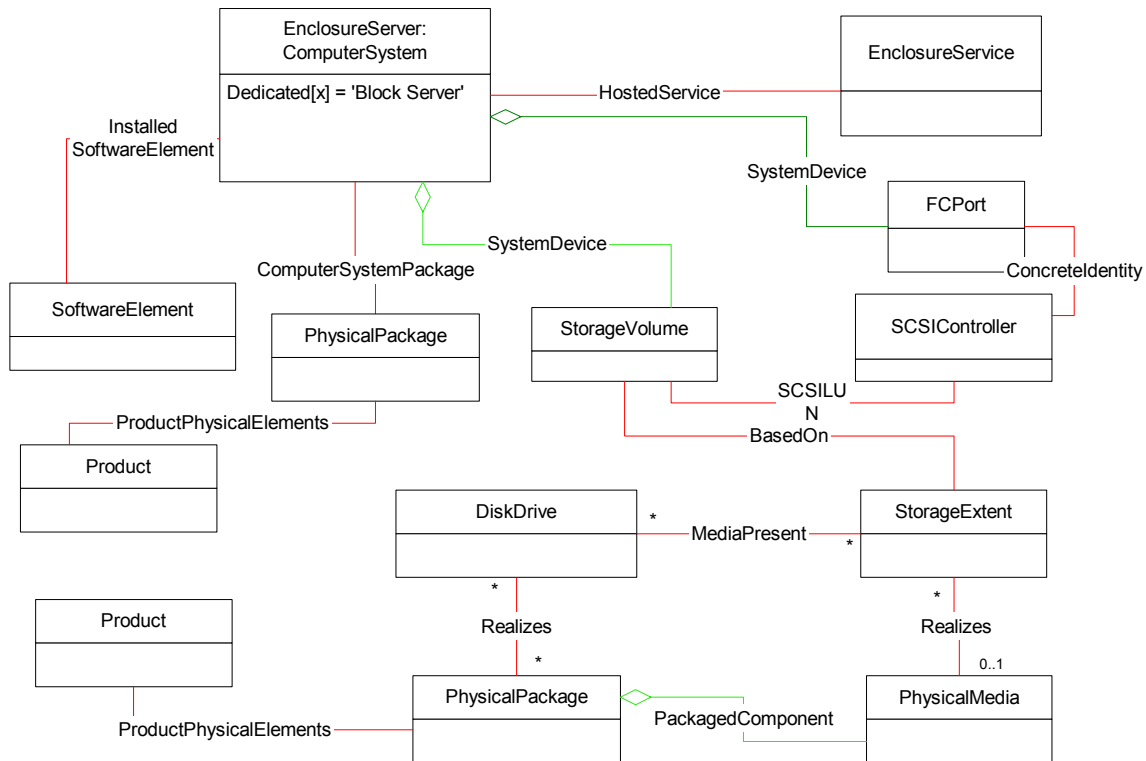


Figure 39: JBOD Array Model

3.3.4.1.4 Client Considerations

Discovering a Disk Array

A disk array is modeled as a **ComputerSystem**. The term “cluster” is used for systems with multiple loosely-coupled processors; the individual processors known as “component” **ComputerSystems**. If the array is a cluster, the model includes a **ComputerSystem** representing the cluster with a **ConcreteIdentity** association to a **RedundancyGroup** object. The cluster **ComputerSystem** has **ComponentCS** associations of **ComputerSystem** objects representing individual processors and the **RedundancyGroup** has an aggregation (**RedundancyComponent**) to the individual **ComputerSystem**. This allows management interfaces to operate on the entire array where appropriate and to also see which physical elements (such as ports) are associated with individual processors. A **Service** (such as an access control service) is associated with the entire array; so it is associated with the cluster **ComputerSystem** rather than its component **ComputerSystems**.

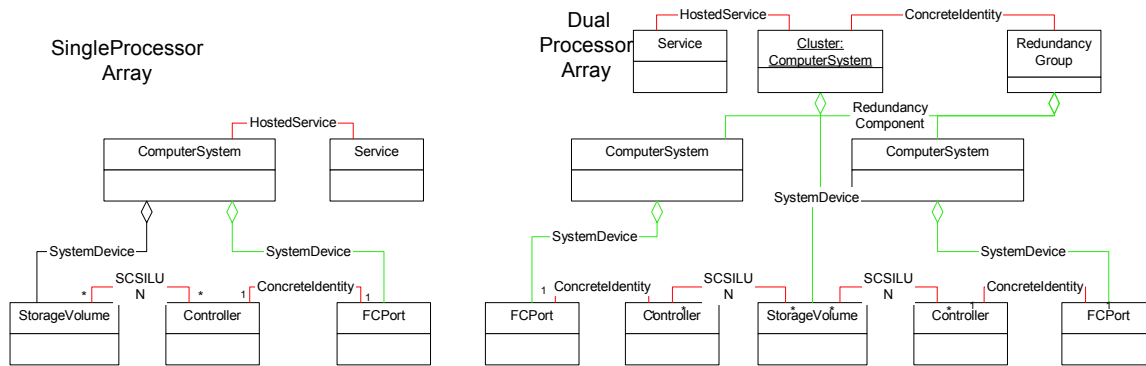


Figure 40: Single/Dual Processor Models

To discover arrays, a client starts by looking up **ComputerSystems** with **Dedicated** property set to “BlockServer”. Follow **ParticipatingCS** aggregations to understand multi-processor configurations. The **ConcreteIdentity** association to **RedundancyGroup** is only present for “virtual” cluster Systems.

The **System NameFormat** attribute identifies how the **Name** field is generated. Disk array **ComputerSystem** names are network host names (**NameFormat** = “IP”), node names (**NameFormat** = “NodeWWN”), platform IDs (**NameFormat** = “T11PlatformID”), or Vendor+Model+SerialNumber (**NameFormat** = “VendorModelSerial”)

Find Asset Information

Information about the entire array is modeled in **PhysicalPackage**. **PhysicalPackage** may be subclassed to **Chassis**; the more general **PhysicalPackage** is used here to accommodate array implementations that are deployed in multiple chassis. **PhysicalPackage** has an associated **Product** with physical asset information such as **Vendor** and **Version**.

List Volumes, Disks, and Pools

StorageExtent represents a manageable unit of storage; it may represent all the storage in a physical disk, a portion of a disk, or some type of virtual disk (which may be a collection of other **StorageExtents**). Some disk products have externalized extents with certain properties (the values you’d see in SCSI Inquiry responses) and internal extents that are apparent only through management interfaces. For example, some cylinders may be hidden from the external view. Similarly, a RAID system may allow an internal RAID group to be divided into smaller, exported extents. **StorageVolume** is a subclass of **StorageExtent** that represents an externalized extent. **StorageExtent** also represents a region of contiguous storage on a **StorageVolume**. A storage system exports an entire **StorageVolume**; the consumer of that volume may opt to treat it as several extents.

Simple Disk Model

A simple disk is modeled as several objects in CIM. As with all CIM modeling, the physical and logical aspects are in separate classes (connected with a **Realizes** association). Media is separated from physical transport to allow consistent modeling of fixed and removable media devices.

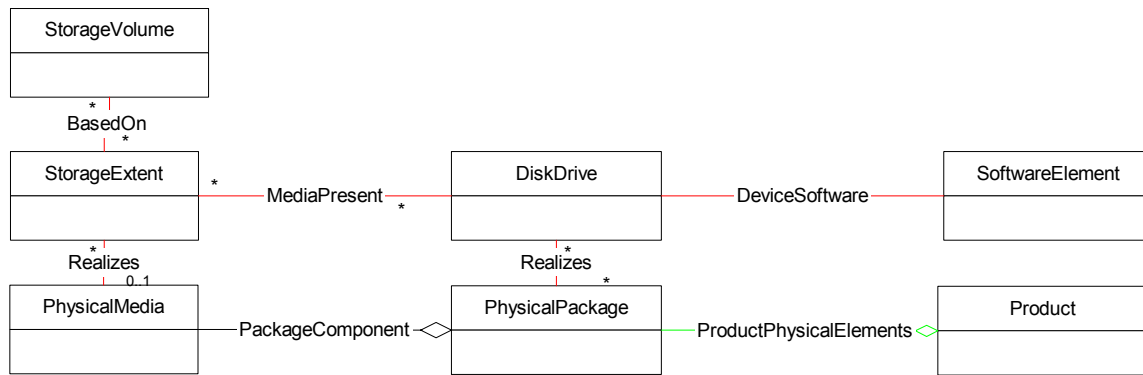


Figure 41: Simple Disk Model

SoftwareElement is optional and represents disk firmware. **Product** represents the assets for the entire disk package and includes vendor, model, and revision identifiers.

StorageVolumes are aggregated from a **System** class with **SystemDevice**.

Find the Durable Name for Volumes

Different implementation use different approaches to uniquely identify SCSI units (**LogicalDevices**). The agent should try these standard interfaces in this order to find a durable volume name. The best name is put in the **StorageVolume** **Name** field. The **NameFormat** attribute of **LogicalDevice** (and subclasses) identifies how the name field is generated. The client should use the same name format to assure a consistent model.

Inquiry VPD page 83 is documented in the SCSI Primary Commands specification. It allows a device to report a list of identifiers in a variety of formats. Identifier type 3 is an IEEE standard that is most commonly used for device identification. The ANSI Name Address Authority (NAA) specifies the format. Association set to 0 indicates this ID represents the logical device rather than a single port. NAA specifies that high order 4 bits define the format used in the rest of the identifier. Other NAA values and identifiers types may be used in older implementations. If the volume does not report page 83, page 80 is a serial number; this value can be merged with vendor and model strings from standard inquiry to generate a unique ID. Some vendors store a serial number in the vendor-specific data in the standard inquiry data. The last option is a FibreChannel WWN that may map 1-1 to a device in JBOD configurations.

Description	NameFormat	Notes
VPD page 83 LU identifier type 3h, Association=0, NAA 0110b	VPD83NAA6	recommended format (16 bytes long) when IDs are generated dynamically
VPD page 83 LU identifier type 3h, Association=0, NAA 0101b	VPD83NAA5	recommended format (8 bytes) for IDs determined in the factory
VPD page 83 LU identifier type 3h, Association=0, NAA 0010b	VPD83NAA2	
VPD page 83 LU identifier type 3h, Association=0, NAA 0001b	VPD83NAA1	
VPD page 83 LU identifier type 2h, Association=0	VPD83Type2	
VPD page 83 LU identifier type 1h, Association=0	VPD83Type1	
VPD page 83 LU identifier type 0h, Association=0	VPD83Type0	
VPD page 80 LU serial number + Vendor + Model	VPD80	only if serial number refers to devices rather than the enclosure
Standard Inquiry serial number + Vendor + Model	INQVS	Vendor-specific - first 8 bytes of Vendor-Specific field
FC Node WWN	NodeWWN	only if associated SCSIController has a single LUN

Table 8: LogicalDevice Durable Names

Discovering RAID groups and Extent Subsets

BasedOn is an association between two **StorageExtents**; it provides the ability to model virtualization by describing how underlying extents can be divided and how extents (or sub-extents) merged into a new, virtualized extent. This is accomplished by specifying properties on the **BasedOn** association. The **StartingAddress** and **EndingAddress** properties define a portion of the underlying extent and **OrderIndex** defines the order which underlying extents are assembled. The function of these properties is easy to understand with examples.

In the diagrams for these examples, the storage consumer is above and the source is below the depicted extents. A simple RAID5 group is a collection of disks. The resulting RAID group is a **StorageExtent** that is based on several underlying disks – which are also **StorageExtents**. **RedundancyGroup** is an aggregation of the underlying extents.

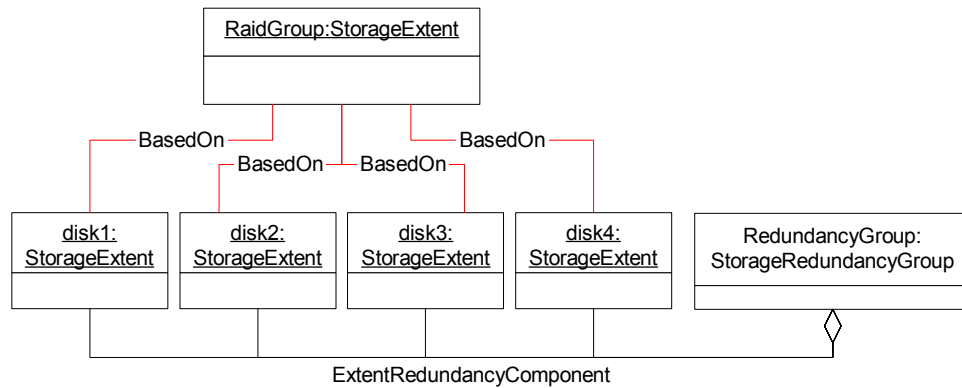


Figure 42: Raid Group Model

BasedOn can also describe a subdivision of an extent using the **StartingAddress** and **EndingAddress** properties.

Some RAID implementations include multiple transformations similar to each of the examples above. For example, a large RAID group may be subdivided into separate volumes.

All of these virtualization implementations are defined in terms of multiple instances of **StorageExtent**. The properties of **StorageExtent** and the **BasedOn** association allow a client to understand the differences between the various implementations. **StorageVolume** is a subclass of **StorageExtent** indicating which extents are externalized from a storage system.

BasedOn associations may extend between storage systems. Consider an appliance that takes volumes from underlying RAID or JBOD systems and concatenates these into larger volumes. The same model elements still apply. The following diagram depicts how a virtualization system can be layered over another virtualization system or a simple disk model.

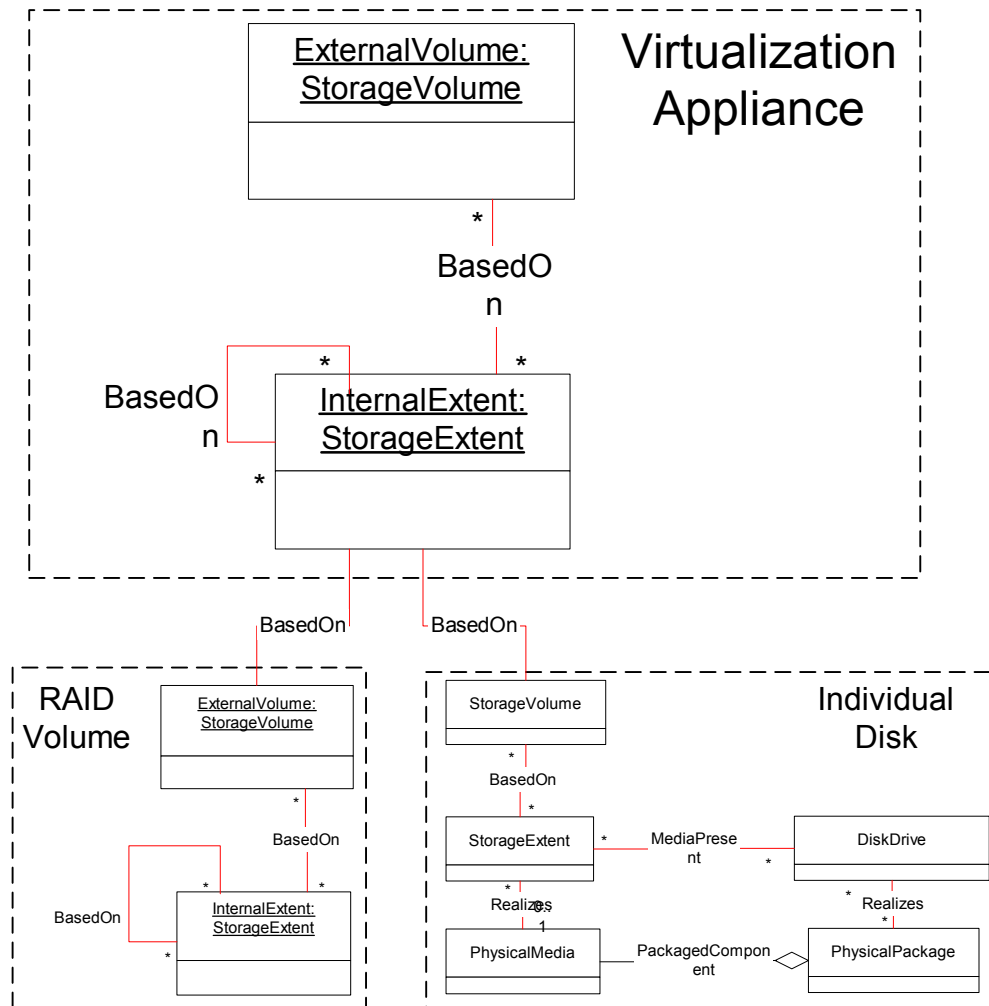


Figure 43: Virtualization Across Multiple Systems

Find Port Information

FCPorts are aggregated from ComputerSystems using SystemDevice. In an array with multiple storage processors, ports are aggregated from the component ComputerSystem; this aggregation allows a client to see which ports are associated with a particular processor and to understand possible single points of failure.

FCPort has a one-to-one ConcreteIdentity association to a SCSIController that represents the SCSI aspects. SCSIController has SCSI Lun associations to LogicalDevices (such as StorageVolumes or DiskDrives).

SCSIControllers can serve as initiators (for example, a port in an HBA) or as targets (ports in devices). A RAID array model may include both; they can be differentiated by the association between the SCSIController and LogicalDevices. Target SCSIControllers have SCSI Lun associations to LogicalDevices. Initiator SCSIControllers may have SCSIInterface associations to LogicalDevices.

Ports may share properties; for example, ports on an HBA card may share the same firmware. This is modeled with an optional PortController associated to each port with ControlledBy.

Find System Status

All of the logical and physical element objects include a status property called OperationalStatus. The defined values are explained in the CIM_Core MOF:

Indicates the current status of the element. Various functional and non-functional statuses are defined.

Functional statuses are “OK” (value=2), “Degraded” (3), “Stressed” (4) and “PredictiveFailure” (5). Stressed indicates that the element is functioning, but needs attention. Examples of Stressed states are overload, overheated, etc. Predictive Failure indicates that an element is functioning nominally but predicting a failure in the near future.

Non-functional statuses are “Error “ (value=6), “Non-Recoverable Error” (7), “Starting” (8), “Stopping” (9), “Stopped” (10), “In Service” (11), “No Contact” (12) and “Lost Communication” (13). “Error “and “Non-Recoverable Error” are self-explanatory. “In Service” describes an element being configured, maintained, cleaned, or otherwise administered. This status could apply during reload of a user permissions list, or other administrative task. “No Contact” indicates that the current instance of the monitoring system has knowledge of this element but has never been able to establish communications with it. “Lost Communication” indicates that the **ManagedSystemElement** is known to exist and has been contacted successfully in the past, but is currently unreachable. “Stopped” indicates that the element is known to exist, is not operational (e.g., it is unable to provide service to users), but it has not failed. It has purposely been made non-operational. The element may have never been OK, the element may have initiated its own stop, or a management system may have initiated the stop.),

Asynchronous status changes are signaled through Alert Indications. More details on indications are in section 3.3.4.1.6 below.

Device Credentials

The device credentials are modeled using the CIM classes **SharedSecretService** and **SharedSecret**. The **ComputerSystem** class represents the device, and the **SharedSecret** object contains the credentials in its properties.

For more information, see Clause 4.3, “Modeling Device Credentials”, below.

Map/Mask (Storage Access) Configuration

Many disk arrays provide an interface for the administrator to specify which initiators (Hosts or HBA WWNs) can access a specific volume. The effect is that the given volume will only be visible to SCSI commands that originate from the specified initiators. There may also be a capability to select which local device ports can be used, an access mode (read-write or read-only), and the SCSI Logical Unit Number seen by the initiators. The **UnitAccess** association models this information;

In a disk array system, **UnitAccess** associates a target **SCSIController** and a **StorageVolume**. It provides initiator name and name-type properties. The name-type property defines whether the initiator name represents a node or port WWN or a host name.

A client can enumerate **UnitAccess** associations by target **SCSIController**, **StorageVolume**, Initiator, or a combination. For example, enumerating all **UnitAccess** associations for a particular target controller and volume lists each initiator that has access. Alternatively, enumerating by initiator and target **SCSIController** provides a list of volumes accessible to that initiator (similar to a SCSI ReportLUNs command).

StorageAccessService provides interfaces for creating and deleting **UnitAccess** associations. A client can locate (and test for the existence of) this service by following the **HostedService** association from the array cluster or system. Depending on the capabilities of the array, different methods are available on this service.

The first method is for implementations where Logical Unit Numbers cannot be changed, the LUN for each **UnitAccess** is determined by the RAID system.

```
Expose(Device, Target, InitiatorID, InitiatorIDFormat, AccessMode)
    LogicalDevice: a reference to a LogicalDevice (e.g. StorageVolume)
    Target: a reference to a Controller on the storage system that is in
    turn associated to a port
    InitiatorID: a Node or Port WWN or host name
    InitiatorIDFormat: the type of Initiator (PortWWN, NodeWWN, Host)
    AccessMode: either read-write, read-only, no-access, or default (the
    value of the DefaultAccessMode property)
```

The result of the `Expose()` method is calling the underlying hardware interface and the creation of a **UnitAccess** association. Most **UnitAccess** properties are defined by the parameters in the `Expose()` method.

Systems that allow the administrator to specify the exported Logical Unit Number provide this method:

```
MapExpose(Device, Number, Target, Initiator, InitiatorIDFormat, AccessMode)
    Number: the Logical Unit Number assigned to the UnitAccess
```

And the other parameters are as defined for `Expose()`.

The result of `MapExpose` is creation of a **UnitAccess**. The LUN parameter defines the LUN property of **UnitAccess**. There cannot be any overlap in the Logical Unit Numbers exposed by a target SCSIController to an initiator; overloading an existing LUN is an error.

Systems that allow the administrator to set the LUN for all initiators provide a `Map` method.

```
Map(LogicalDevice, Number, Target)
    The parameters are as defined for Expose().

Deny(Device, Target, InitiatorID, InitiatorIDFormat);
```

`Deny` is an optional method for systems with an interface to deny access from a particular initiator.

The result of `Map` is creation of a **UnitAccess** association with `Initiator` set to `NULL` (implying that any initiator can access the unit).

`Unexpose(UnitAccess)` removes a **UnitAccess** Association.

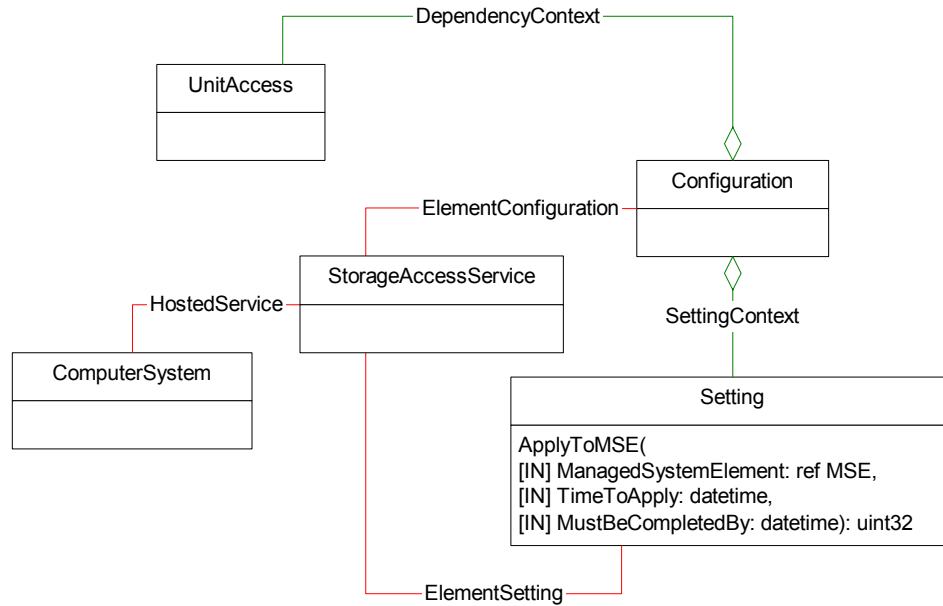
`AvailableMethods` is a string list of methods defined by the agent (typically either `Expose` and `Map` are provided or `MapExpose` is provided, but not all three). It allows a client to see which methods `StorageAccessService` provides on this array.

If an array can accept different `InitiatorIDFormat` values (for example, node and port WWNs), clients must use `InitiatorID` and `InitiatorIDFormat` consistently. If the `InitiatorIDFormat` is specified incorrectly, the results will not be as expected.

LUN Masking Batch Commit Considerations

Some RAID systems provide an `Expose` interface, but treat the changes as pending until an explicit `Commit` method is called. The existing `Configuration` and `Setting` classes can model this behavior. The pending **UnitAccess** associations are aggregated from `Configuration`.

Figure 44: Batch LUN Masking Objects



The **Configuration** and **Setting** objects are associated with the service (rather than **ComputerSystem**) so they can be unambiguously identified. The **Configuration** and **Setting** (and related associations) are not instantiated by providers that do not use the commit behavior.

Working with Storage Pools

The figure below shows the model for volume creation; this is a generic approach that may be used in addition to (or in place of) an implementation model. This generic model allows a client application to provide a simple interface to less experienced administrators or for use in automation. **StorageConfigurationService** contains methods to allow manipulation of **StoragePool** and **StorageCapabilities** classes. **StoragePool** is a generic element that represents some assignable storage.

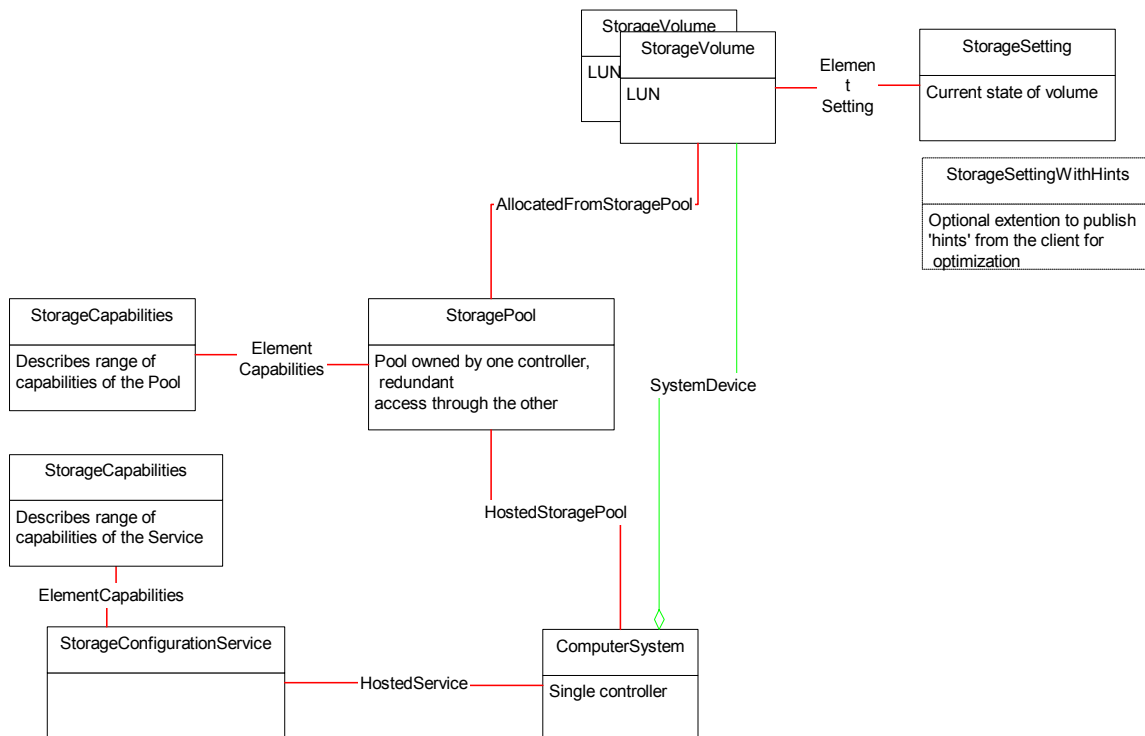


Figure 45: Storage Configuration

The **StorageConfigurationService** class contains methods to allow creation, modification and deletion of **StoragePool** and **StorageVolume**. 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 clues to the system as to how it expects to use the storage for optimization purposes. For example, if the array supports the creation of Pools which can tolerate the loss of two disks then the 'spindle redundancy' attribute will include 2 in its range of supported values. The client would create an instance of **StorageCapabilities**, set 'spindle redundancy' to 2 and pass a reference to the class to the **StorageConfigurationService.CreateStoragePool** method.

When creating a **StorageVolume**, an instance of **StorageSetting** is passed as a parameter to the **StorageConfigurationService.CreateStorageVolume** method. This forms an objective for that element to attempt to meet. The current 'service level' being achieved is reported via the **StorageInformation** class.

Storage Pools and Volume Creation

A Storage Pool is an abstract representation of storage suitable for configuration and allocation or "provisioning". A Storage Pool is simply a 'blob' of unallocated storage. However, it may have preformatted into a form (such as a RAID group) that makes volume creation easier. It is associated with a set of capabilities held in the **StorageCapabilities** class that reflect the configuration parameters that are possible for volumes created from this pool. Either **StorageVolumes** or **StoragePools** can be allocated from a **StoragePool** (the result of which is indicated with the **AllocatedFromStoragePool** association). Usually, there is also a base **StoragePool** that is optionally created from underlying **StorageExtents** and these are indicated using the **StoragePoolComponent** association.

StorageVolumes are created from **StoragePools** via a **StorageConfigurationService**'s **CreateVolume()** 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 **StorageVolume.OperationalStatus** is OK. A Client may also follow the progress of the operation using the **StorageConfigurationJob** class and its properties.

This approach 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 **StorageCapabilities** class describes the actual capabilities of the available storage in the **StoragePool**. The **StorageSettings** class is provided by clients to describe the desired configuration of the allocated storage. The parameters exposed and controlled via the **StorageCapabilities/Setting** 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.
- **Spindle Redundancy.** This describes how many disk spindles can fail without data loss (including a spare, but not more than a single global spare). Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 global (to the system) spares would be 3.
- **Delta Reservation.** This is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations.

In add addition to these settings, a subclass of **StorageSetting** is available that includes some 'hints'. These allow a client to provide extra information to 'tune' a **StorageVolume**. If a client chooses to supply these hints when creating a **StorageVolume**, the **StorageSystem** can either use them in determining a matching configuration or it can choose to ignore the hints.

The example below shows the classes and associations needed to model a single Pool with two **StorageVolumes**.

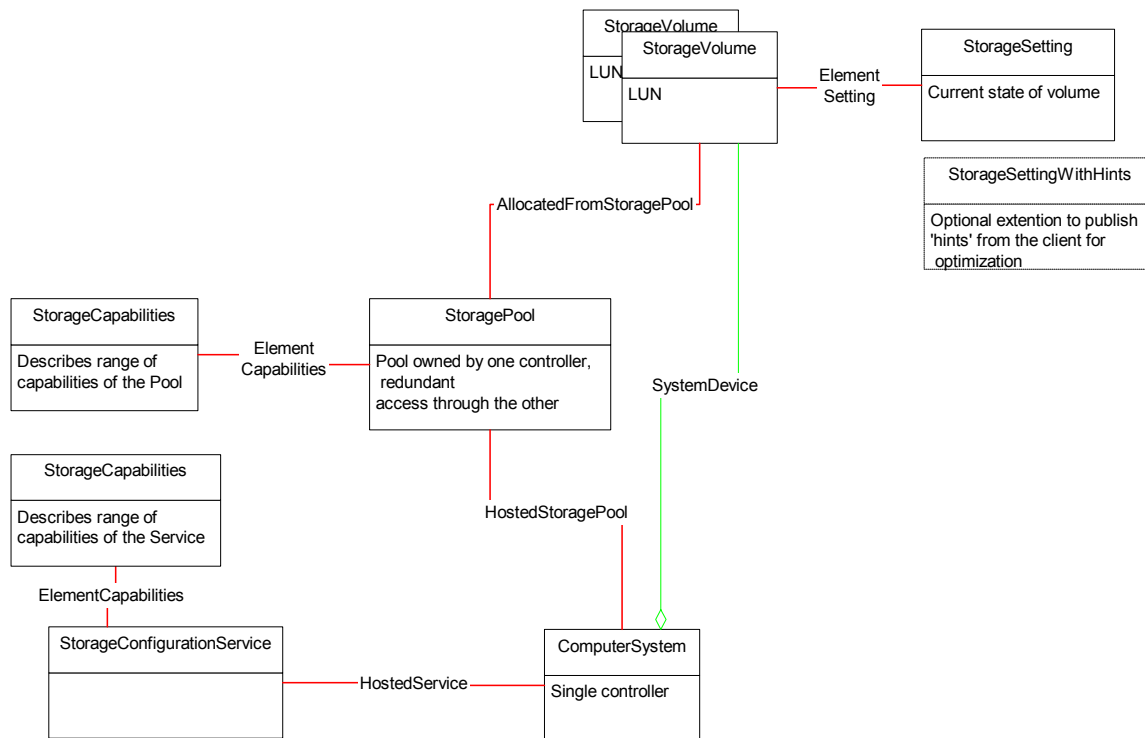


Figure 46: Storage Pool Example

The storage configuration service provides a means to create pools and volumes. The following methods are available in the service.

- **CreateStoragePool:** Create a pool of storage with some set of Capabilities described by the input StorageCapabilities. The source of the storage can be other pool(s) or storage extents.
- **CreateStorageVolume:** Create a StorageVolume with a specific StorageSetting from an source StoragePool.
- **ModifyStorageVolume:** Change the StorageSetting and/or size for a volume
- **ModifyStoragePool:** Change the size and/or capabilities of a storage pool.
- **DeleteStoragePool:** Delete a storage pool and return the freed up storage to the underlying entities.
- **DeleteStorageVolume:** Delete a storage
- **KillJob:** Stop an executing storage configuration job.

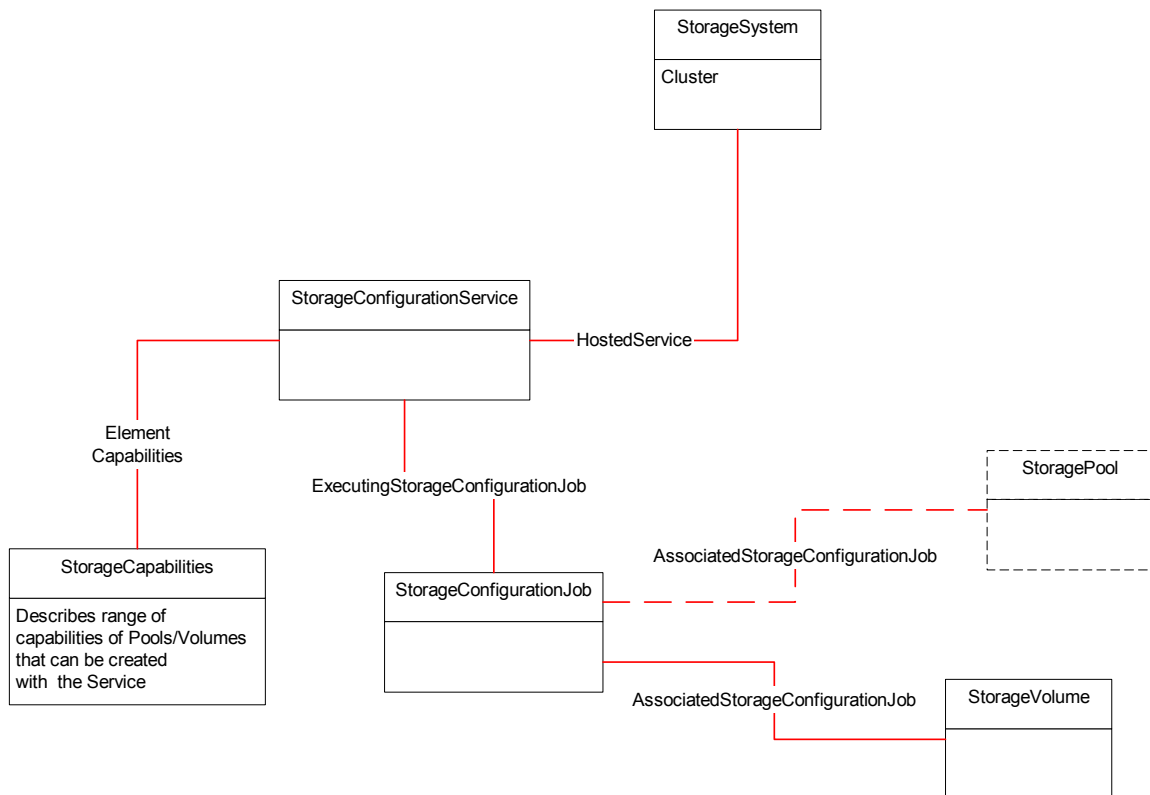
InstanceType in the attached StorageCapabilities objects can be queried to determine what types of objects can be created. There will be one instance of StorageCapabilities for each type of object. The default values can be determined from this as well (you can use 'null' when passing a Capability or Setting into one of the methods to select default values).

If the operation will take a while (longer than an HTTP timeout), a handle to a newly minted StorageConfigurationJob will be returned. This allows the job to continue in the background. Note a few things:

- The job may be queued. You may have multiple outstanding jobs against a pool for instance. The job status shows this.
- The job is weak to the Service (shown via ExecutingStorageConfigurationJob) and is also linked to the object being modified/created via AssociatedStorageConfigurationJob. 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 things happen quickly, a null can be returned as a handle.

Figure 47: Storage Configuration



In the example, jobs are not used since configuration changes are quick. Also only Volumes can be created, the pools are fixed, so there is on one **StorageCapabilities** object.

3.3.4.1.5 Agent Considerations

This section describes how optional behavior should be modeled. This allows a client to quickly determine whether an agent provides a capability, then how to discover or change those aspects of the model.

An agent developer must decide which optional aspects of the model will and will not be modeled. There are several reasons you may opt to not model certain aspects of a disk array. Information about part of the model may not be available; for example, there may be information about which of multiple Ethernet ports is actually active. The agent designed may opt to hide proprietary portions of the model or simply opt for a simpler model that can be deployed more quickly.

Simple Volume Model

One option is to simply model the exported volumes without modeling the underlying physical disks. This prevents a client from understanding physical disk problems and the virtual-physical mapping that could assist in troubleshooting media problems. But this may be a valid option if other tools already provide this capability or for virtualization appliances where the mapping can change frequently.

The minimum information needed is an identifier and capacity for the virtual models; this can be modeled with a **StorageVolume** objects. In particular, the Name property should be determined as described in the Durable Name section of Client Considerations (see Clause 3.2.3).

To model the vendor, model, and firmware IDs, connect the virtual **StorageVolume** to a **Product** with a **ConcreteIdentity** association. If the volume is mapped (see LUN Masking below), the SCSI Logical Unit Number (LUN) may vary with initiators and is modeled as the **UnitNumber** property of the **UnitAccess** association. If the volume is not mapped, its LUN is set in the **DeviceNumber** property of the **SCSILun** association. If the unit number is specified in both the **SCSILun** and **UnitAccess** association, the value in **UnitAccess** takes precedence. This allows a Logical Unit to have a default value that may be overridden for specific initiators.

The **StorageVolume DeviceID** property holds the array internal ID – usually an integer between zero and the maximum supported number of volumes.

Physical Disk Model

Physical disks are modeled as several classes, some classes being optional.

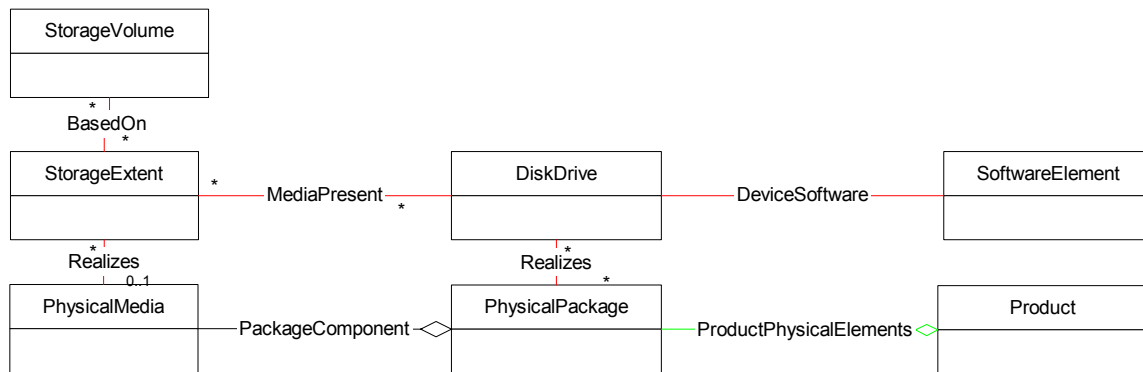


Figure 48: Physical Disk Model

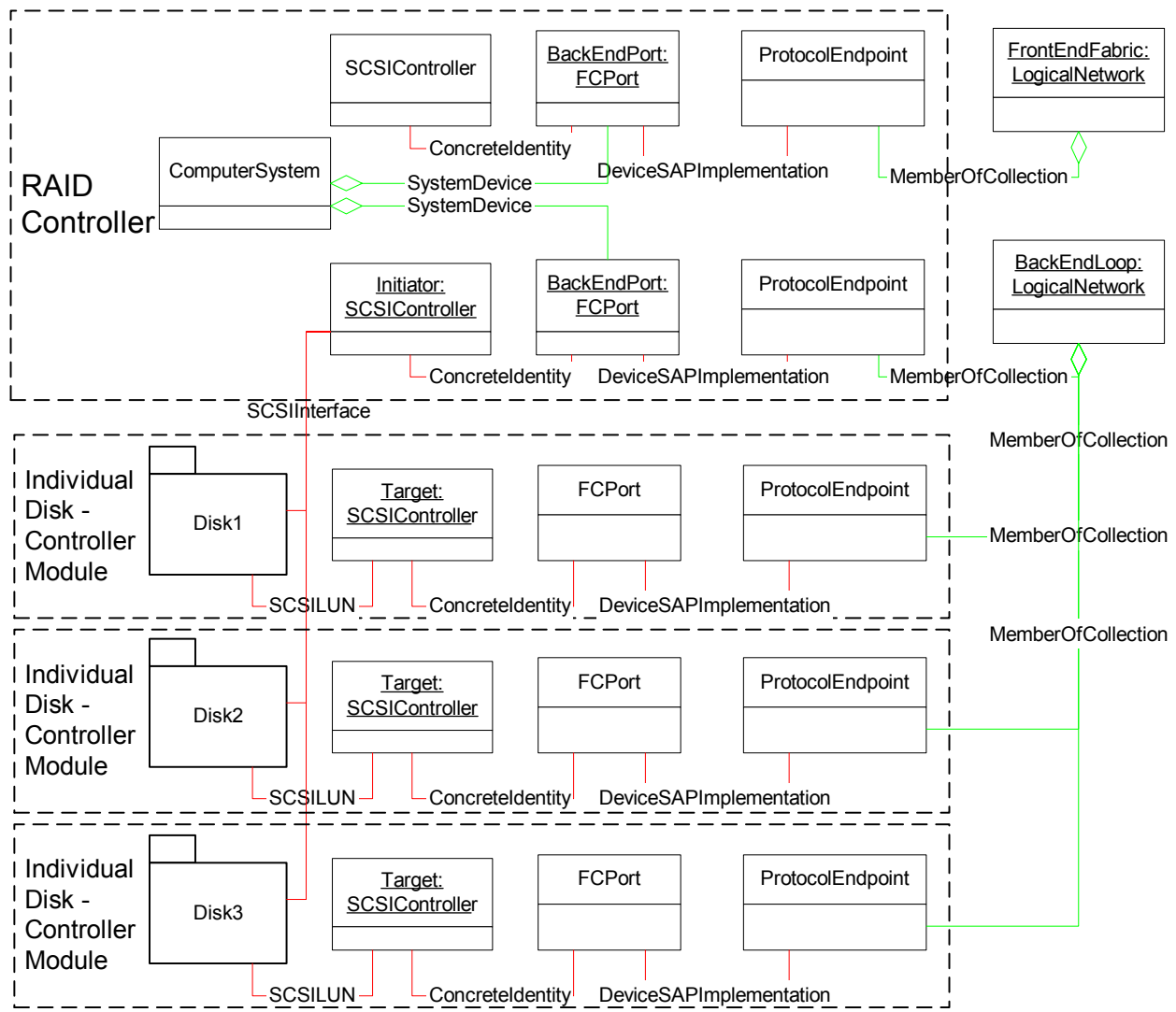
SoftwareElement models firmware and is optional. As with a volume, **Product** models asset information for the disk.

Modeling Array Internal Connections

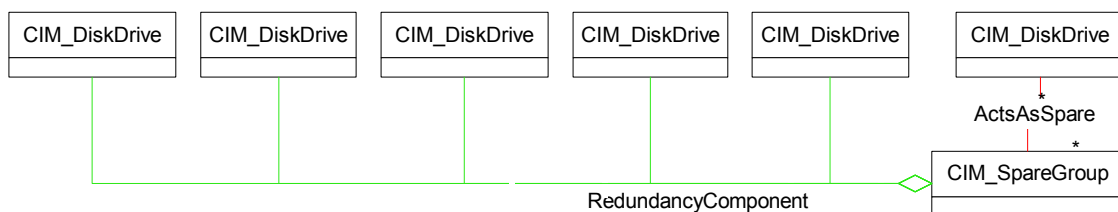
Disks have capabilities hidden from general use. For example, a disk may have hidden capacity or non-standard interfaces used for maintenance. In this case you can model a separate **StorageExtent** representing the internal view with a **StorageVolume** based on the extent providing the external view.

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.

When an array's backend connectivity is included, the model has similarities to a Router (See Clause 3.3.2.4). The RAID controller itself has front-end ports (connected to customer hosts or switches) and back-end ports (connected to the internal disks). Here's an instance diagram for three disks in an array, connected by a FC loop. The group of physical disk classes is represented by a UML package. There is a "back-end" FC loop internal to the array and a "front-end" fabric where hosts would connect.



Spare Disks



A spare disk is modeled with the ActsAsSpare association to a SpareGroup – which has aggregation associations to other disks.

Figure 50: Spare Disk

This **SpareGroup** object associates one or more spares with a group of active disks. The members of this group are implementation and configuration dependent. In some cases, this group might be the disks in a enclosure, disks in a RAID group, or disks of a particular make and model. The cardinality on **ActsAsSpare** allows multiple spares per group and also allows a spare to participate in multiple groups.

RAID Group

A RAID group is modeled with a **StorageExtent** representing the RAID group and a **StorageRedundancyGroup** with **ExtentRedundancyComponent** aggregations to each component extent.

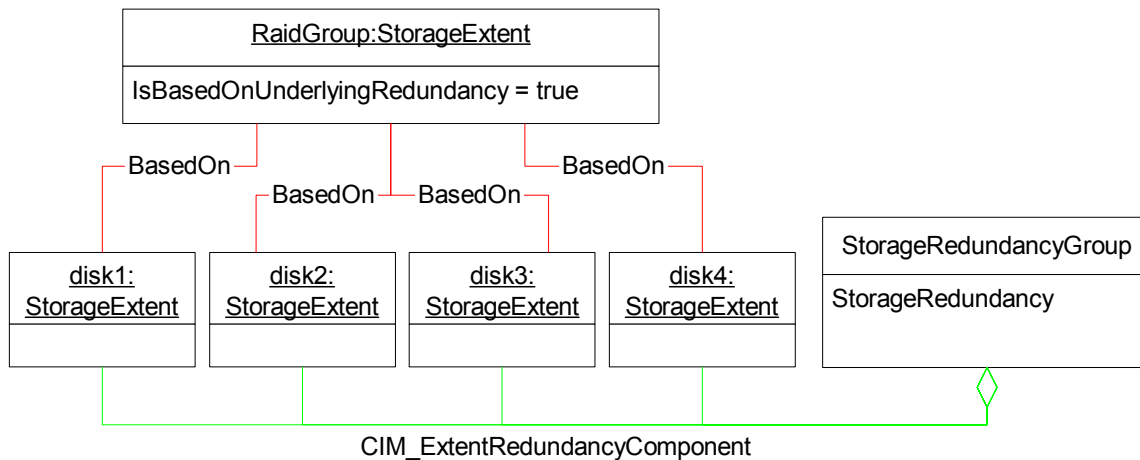


Figure 51: Storage Redundancy Model

The extent representing the RAID group has the **IsBasedOnUnderlyingRedundancy** property set. If this extent is exported as a volume, it is subclassed as a **StorageVolume**. It can also be left as a **StorageExtent** that is then divided and exported as multiple volumes.

The state of the redundancy group is held in the **StorageRedundancy** property of the **StorageRedundancyGroup**. Possible values are "No Additional Status", "Reconfig In Progress", "Data Lost", "Not Currently Configured", "Protected Rebuild", "Redundancy Disabled", "Unprotected Rebuild", "Recalculating", and "Verifying".

Firmware

Firmware is modeled as **SoftwareElement**. **InstalledSoftwareElement** is used for firmware associated with a **System**; **DeviceSoftware** associates **SoftwareElement** to a **DiskDrive**, **Port**, or **PortController**.

JBOD (non-RAID) Arrays

The simplest JBOD array model is a collection of physical disk models (see **Physical Disk Model** above). Many JBOD arrays include a minimal processor with a management interface. This processor may use the SCSI Enclosure Service (SES) interface, or a proprietary in-band or Ethernet interface. In practice, these arrays have a variety of diagnostic capabilities and no particular profile is common other than the ability for an agent to discover the components in the array. This is reflected with a common model to describe JBOD arrays, plus a **Service** for proprietary interfaces.

The JBOD management interface is modeled as a **System**. **SystemDevice** associations aggregate **Disks** and **Ports** to the **System**.

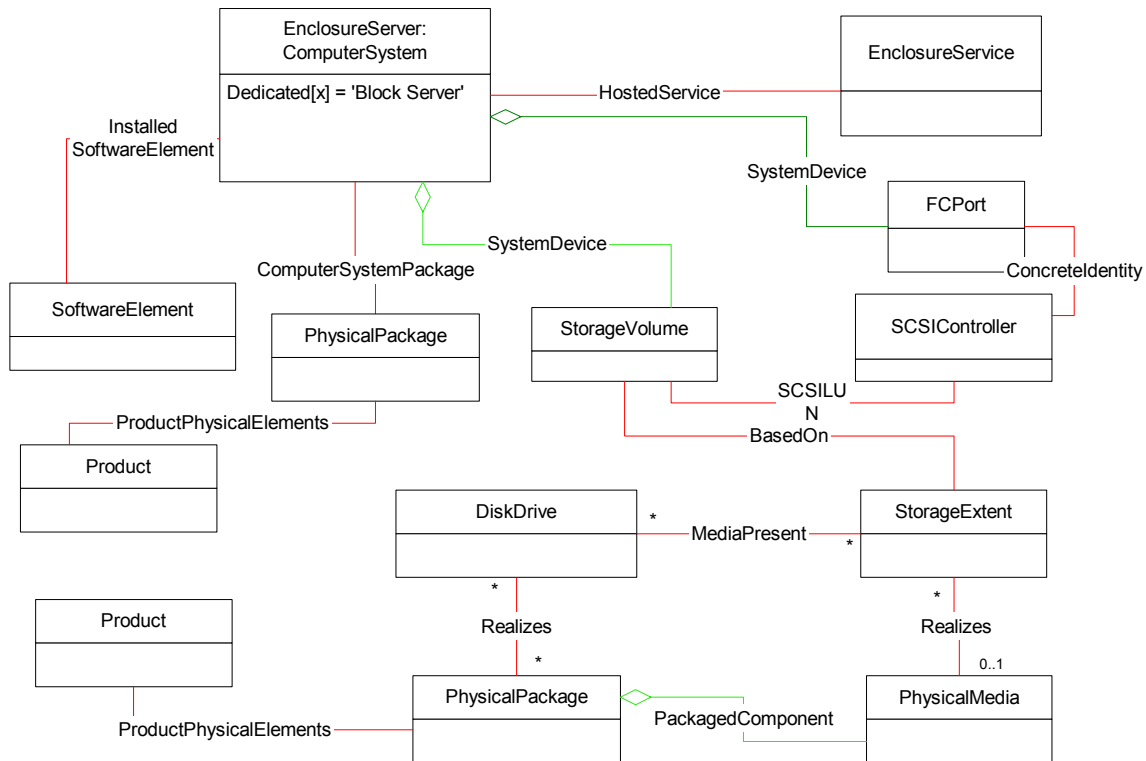


Figure 52: JBOD Model

Virtualization Appliance Considerations

Virtualization Appliances take **StorageExtents** and expose **StorageVolumes** to consumers – typically file-systems or databases. Virtualization Appliances add flexibility in allocating volumes; often allowing allocation of arbitrary size volumes and may provide redundancy, remote mirroring, or snapshots. There are many variations in actual implementations, so there is not a common profile. Figure 43: Virtualization Across Multiple Systems shows how **BasedOn** associations model striping and concatenation. This section describes how other common Virtualization Appliance behavior should be modeled.

Symmetric (or *in-band*) Virtualization Appliances have the appliance connected between the underlying storage and consumer. *Asymmetric* (or *out-of-band*) Virtualization Appliances have the consumer directly connected to the storage; the appliance is also connected to the consumer hosts.

The following diagram models an asymmetric virtualization appliance. The appliance has **FCPorts** with a proprietary connection to Host HBA ports that are also connected to underlying storage. Volumes exported by the Virtualization Appliance are defined in maps shared by the appliance and HBA. The map tells the HBA which **SCSIInterface** associations and device addresses are actually used to enact the consumers I/O requests.

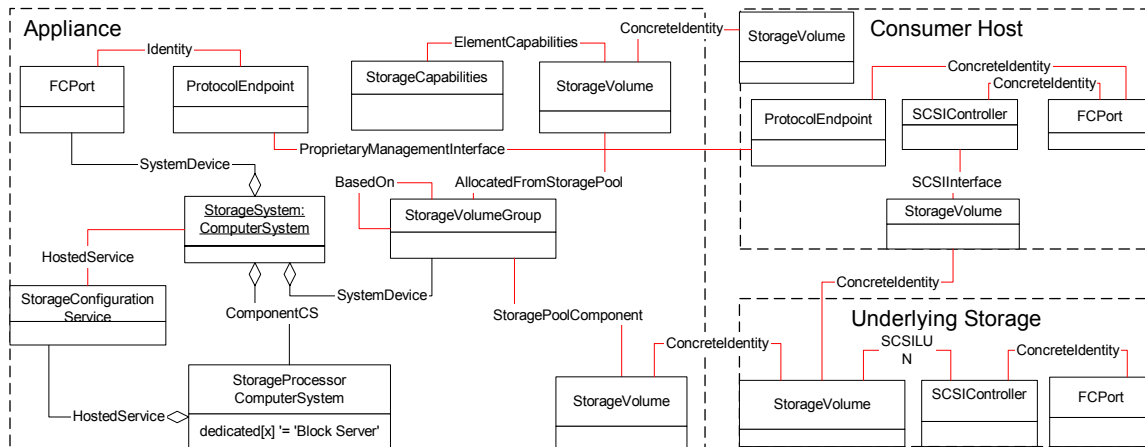


Figure 53: Asymmetric Virtualization Appliance

Symmetric Virtualization Appliances typically have front-end ports for connection to consumer hosts and back-end ports connected to underlying storage. The model consists of classes representing the appliance system, SCSIController/FCPort classes for front-end and back-end ports, StorageVolume imported from underlying storage and virtualized StorageVolumes exported to consumers. Note that the model is similar

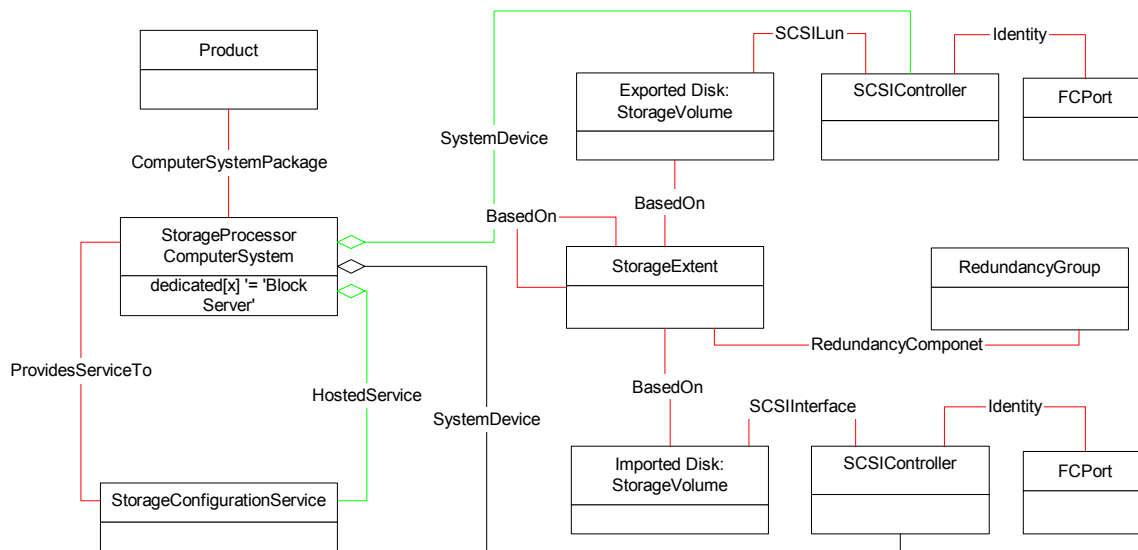


Figure 54: Symmetric Virtualization Appliance

StorageAccessService

This service provides the interfaces for LUN Mapping and Masking, discussed above in **Map/Mask (Storage Access) Configuration** under Client Considerations. The Expose/Unexpose methods cause the agent to call the underlying hardware masking interfaces and to update the model. An array agent will probably only provide one of the Expose methods; Expose() and Map() are used on arrays where the LUN value is the same to all initiators and paths. MapExpose() is used on arrays where the LUN is assigned as part of LUN Masking. The service includes a string property **AvailableMethods** with a list of the methods that are valid for this agent.

Map and MapExpose allow a client to specify the Unit Number exposed for a volume. SCSI requires that unit numbers be unique for a particular initiator/target combination. If the agent specifies a Unit Number that is already in use, the agent should return invalid parameter.

The Expose methods include InitiatorId and InitiatorIdFormat parameters. InitiatorIdFormat defines the types of InitiatorID, PortWWN, NodeWWN, or Hostname. If the client specifies an unsupported format, the agent should return invalid parameter.

3.3.4.1.6 Indications

Mandatory

- InstIndication

- Creation of StorageVolumes

```
SELECT * FROM CIM_InstCreation
      WHERE SourceInstance ISA CIM_StorageVolume
```

```
SELECT * from CIM_InstModification
      WHERE SourceInstance ISA CIM_StorageVolume AND
      SourceInstance.OperationalStatus[0] == OK
```

(OperationalStatus test in case volume object is created before it is (re) constructed)

- Deletion of StorageVolumes

```
SELECT * FROM CIM_InstDeletion
      WHERE SourceInstance ISA CIM_StorageVolume
```

- Creation, Deletion of arrays

```
SELECT * FROM CIM_InstCreation
      WHERE SourceInstance ISA CIM_ComputerSystem
```

```
SELECT * FROM CIM_InstDeletion
      WHERE SourceInstance ISA CIM_ComputerSystem
```

- Creation, Deletion of ports

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_FCPort
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_FCPort
```

- Changes to Volume/Target-controller configuration

```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_SCSILUN
```

```
SELECT * FROM CIM_InstDeletion WHERE SourceInstance ISA CIM_SCSILUN
```

- Status changes in Required elements

```
SELECT * FROM InstModification
      WHERE SourceInstance.OperationalStatus[0] <>
      PreviousInstance.OperationalStatus[0] AND SourceInstance
      ISA CIM_StorageVolume
```

The client can then look at SourceStatus.CreationClassName and SourceInstance.OperationalStatus to determine how to handle the indication. Specific elements/statuses that must be indicated:

- StorageVolume Degraded
- StorageVolume other bad status

- Array, FCPort not okay (replace ISA CIM_StorageVolume with appropriate class name)

Context Mandatory – Mandatory if the element type is modeled

- InstIndication
 - DiskDriveCreation/Deletion


```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_DiskDrive
(similar for InstDeletion)
```
 - StoragePool Creation/Deletion


```
SELECT * FROM CIM_InstCreation WHERE SourceInstance ISA CIM_StoragePool
(similar for InstDeletion)
```
 - A StorageConfigurationJob completes normally


```
SELECT * FROM CIM_InstIndication
WHERE SourceInstance ISA CIM_StorageConfigurationJob AND
SourceInstance.OperationalStatus == "Stopping"
```
 - A StorageConfigurationJob completes with a failure


```
SELECT * FROM CIM_InstIndication
WHERE SourceInstance ISA CIM_StorageConfigurationJob AND
SourceInstance.OperationalStatus == "Error"
```
 - StorageExtentCreation/Deletion


```
SELECT * FROM CIM_InstCreation
WHERE SourceInstance ISA CIM_StorageExtent
(similar for InstDeletion)
```
 - UnitAccess Creation/Deletion


```
SELECT * FROM CIM_InstCreation
WHERE SourceInstance ISA CIM_UnitAccess
(similar for InstDeletion)
```
 - Status changes in optional elements (same filter as above, includes optional class types)


```
SELECT * FROM InstModification
WHERE SourceInstance.OperationalStatus <> OK
```

The client can then look at `SourceStatus.CreationClassName` and `SourceInstance.OperationalStatus` to determine how to handle the indication. Specific elements/statuses that must be indicated:

- StorageExtent not okay
- StoragePool not okay
- StorageVolumeSettings not okay
- DiskDrive not okay

Optional

- InstIndication

- Creation/Deletion of **PhysicalPackage**, **PortController**, etc – subscription to creation, deletion, and modification of ANY physical element

```
SELECT * FROM CIM_InstIndication
WHERE SourceInstance ISA CIM_PhysicalElement
(use subclasses to narrow scope)
```

- **AlertIndication**

- Proxy Agent/provider reports failure of un-modeled underlying element

```
select * from AlertIndication
WHERE PropbaleCause == "Underlying Resource Unavailable"
```

3.3.4.1.7 Required Classes

Class	RAID	JBOD	Virtualization	Description
BasedOn (p. 215)	Y	Y	Y	
ComputerSystem (p. 217)	Y	Y *	Y	*a JBOD with no System should be modeled as individual physical disks
Configuration (p. 219)	I	N	I	Same as Setting
DiskDrive (p. 223)	N	Y	N	
ExtraRedundancyGroup (for multi-processor arrays)	I	I	I	Required if implementation has multiple processors/systems
FCPort (p. 224)	Y	Y	Y	
FCPort (p. 224) (for an initiator)	I	N	I	For implementations that model backend storage
StorageAccessService (p. 258)	I	N	I	Required in implementations with access control
PhysicalMedia (p. 242)	N	Y	N	
PhysicalPackage (p. 243) (for systems)	Y	Y	Y	
PhysicalPackage (p. 243) (for DiskDrive)				
PortController	N	N	N	
Product (p. 245) (for ComputerSystem)	Y	Y	Y	
Product (p. 245) (for DiskDrive)	Y	Y	Y	
SCSIController (p. 249) (representing target)	Y	Y	Y	
Setting (p. 253)	I	N	I	Same as above
SCSIController (p. 249) (representing initiator)	N	N	N	
Setting (p. 253)	I	N	I	Required in implementations with access control batch commit
SoftwareElement (p. 253) (for disks)	N	N	N	
SoftwareElement (p. 253) (for systems)	N	N	N	
SpareGroup (p. 257)	N	N	N	

Class	RAID	JBOD	Virtualization	Description
StorageCapabilities	I	N	I	Same as StorageConfigurationService
StorageConfigurationJob (p. 266)	I	N	I	Same as StorageConfigurationService
StorageConfigurationService	I	N	I	For implementations that provide dynamic volume creation service
StorageExtent (p. 258)	N	Y	N	
StoragePool	I	N	I	Same as StorageConfigurationService
StorageRedundancyGroup (p. 279)	I	N	I	For implementations that expose underlying extents of virtualized volume
StorageSetting (p. 280)	I	N	I	Same as StorageConfigurationService
StorageVolume (p. 275)	Y	Y	N	
UnitAccess (p. 289)	I	N	I	Same as StorageAccessService

Table 9: Required Classes for Disk Arrays

3.3.4.2 Tape Library

3.3.4.2.1 Description

The schema for a `StorageLibrary` provides the classes and associations necessary to represent various forms of removable media libraries. This profile is based upon the CIM 2.6 model and defines the subset of classes that supply the necessary information for robotic tape 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 may be found in the appendix and are from the CIM 2.6 MOF. Only the classes unique to tape libraries are described by this Profile. Other classes that are common to multiple Profiles may be found elsewhere in this specification.

The relevant objects for a tape library should be instantiated in the name space of the provider (or agent) for a tape 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 Tape Libraries durable names are defined for the following resources:

- `FCPort`
- `ChangerDevice`
- `TapeDrive`

The durable names are defined in a following subsection of this profile. All other objects do not require durable names and will have instances within a single name space.

3.3.4.2.2 Schema Diagrams

The following sub-sections provide a collection of coherent views for a Storage Media Library, each presenting a different perspective into the underlying schema. The views mimic the partitioning of the overall CIM model as distributed by the DMTF, e.g. Device view, Physical view, etc. The overlapping views each contain this profile's primordial `StorageLibrary` class.

3.3.4.2.2.1 Storage Library System View

This diagram presents a global view of a Storage Media Library and how it connects into the overall schema via the **StorageLibrary** class. Other diagrams in this section provide related views offering more details pertinent to the perspective being described.

One or more Chassis objects through instances of the **LibraryPackage** association realize a Storage Media Library. These physical elements provide aggregation points for the other physical components comprising the library. The section on Storage Media Library Physical View expands upon this facet of the model.

The logical devices comprising the Storage Media Library are readily determined from the **SystemDevice** association off of the **StorageLibrary** instance. The section on Storage Media Library Device View expands upon this aspect of the model.

Product information for Storage Media Library instances of **StorageLibrary** are accessed through the **ProductPhysicalElement** association off of the **Chassis** object that serves as the physical realization for the library. Corresponding **Product** instances may be associated with the realization of **TapeDrive** logical devices.

Services that are pertinent to Storage Media Library instances are located via the **HostedService** association. See the following section on the Storage Media Library Software/Service View for more details regarding services and device firmware.

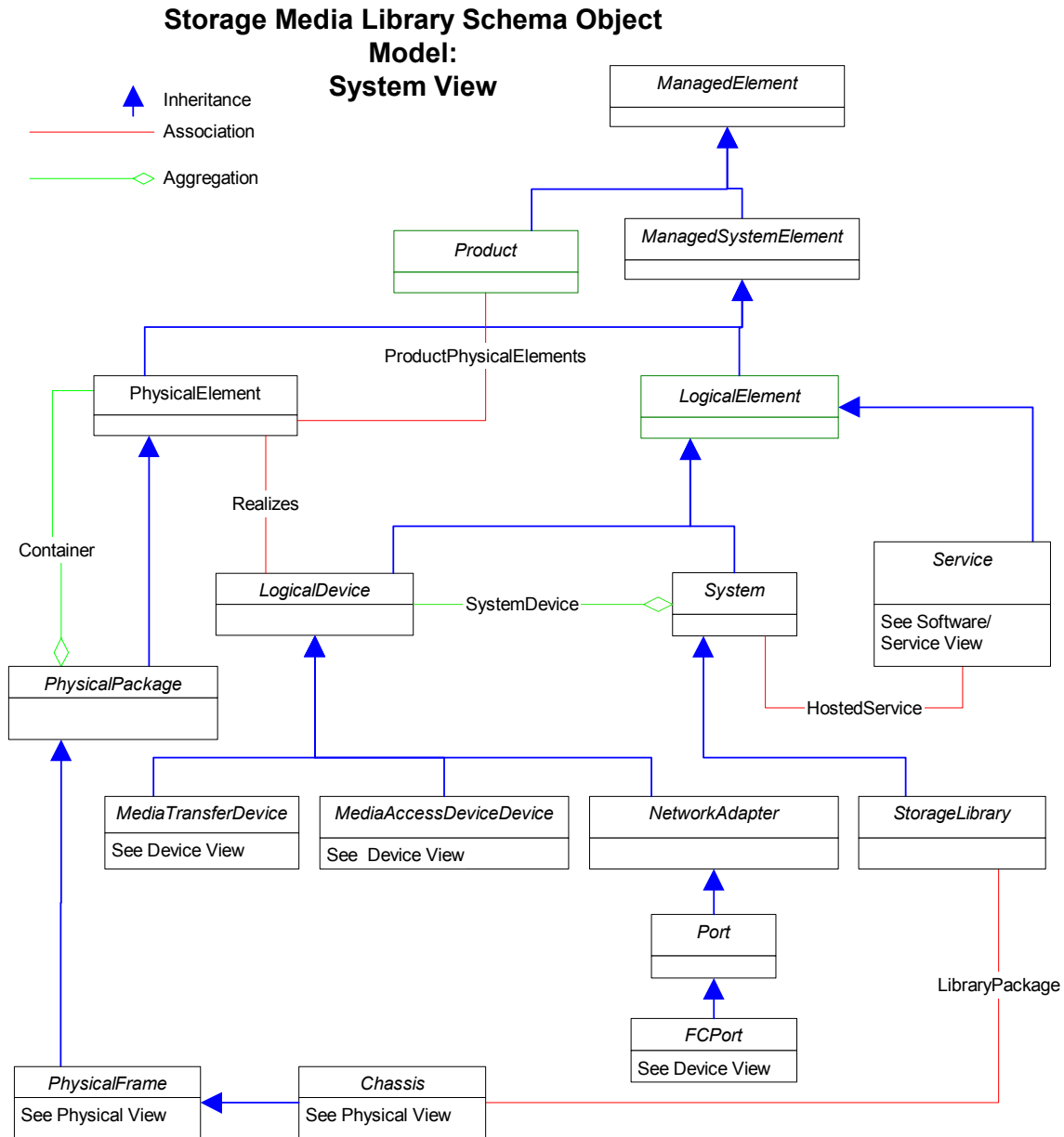


Figure 55: Symmetric Virtualization Appliance

3.3.4.2.2.2 Storage Media Library Device View

The Logical Devices that a Storage Media Library may contain are shown in this diagram along with the necessary associations that facilitate navigation and access to the device instances. Some devices from the CIM model for Storage Media Libraries have been omitted from this profile in order to simplify the development of agents and client applications. Future versions of this profile can consider re-introducing some of these devices as needs dictate. The following devices have been omitted from this profile:

- Picker
- LabelReader
- Door
- StorageExtent

- **TapePartition**

Not all Storage Media Library logical devices may exist within a particular Storage Media Library instance of **StorageLibrary**. However, the logical devices depicted in the following diagram should be represented if there is a physical instance of the device. Very few tape libraries do not have tape drives; therefore any tape drives accessed by the library's changer devices must be instantiated by the agent.

A good example of this is the **InterLibraryPort** device. Many libraries do not contain this device. However, when an **InterLibraryPort** is present it should be presented by the library Agent conforming to this profile. Likewise for the **ChangerDevice**, **LimitedAccessPort** and the **TapeDrive**.

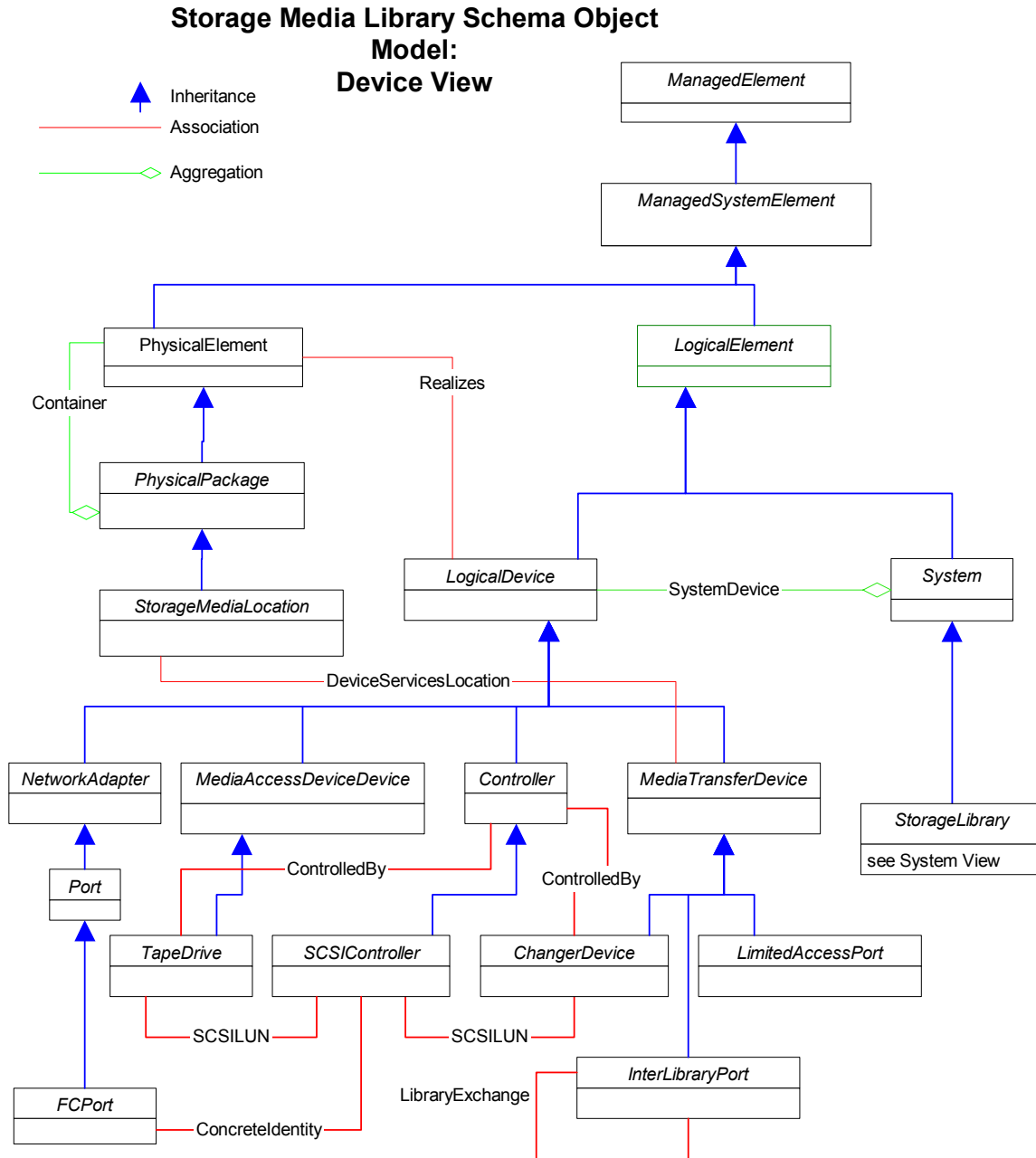


Figure 56: Storage Media Library Device View

3.3.4.2.2.1 Controllers for Media Changers

The mechanism for controlling storage media library robotics is captured in the *ChangerDevice* class. A storage media library's *ChangerDevice* will frequently support *SCSI Media Changer Commands* and will be represented by the library's agent with a *SCSILun* association to a *SCSIController*. The *SCSILun* association is a subclass of *ControlledBy*.

The agents for HBAs(initiator) having visibility to the *SCSIController* will create a *SCSIInterface* association to the *ChangerDevice* instance. These instances will link objects in different name spaces and be correlated by the Durable Name mechanism for library changer devices.

However, some tape libraries will not support SCSI Media Changer commands and will not have the corresponding SCSI Interface. For these changer devices, the agent can use a **ControlledBy** association to a **Controller** instance. A consequence of this is that another mechanism for discovering the library's control path is used. See the section on client considerations for control path discovery.

For **ChangerDevice** instances supporting *SCSI Media Changer Commands* there should be a corresponding Logical Unit for the changer device that has visibility to the Storage Area Network. See the discussion on Durable Names for identification of these Logical Units within the SAN.

3.3.4.2.2.2 Controllers for Tape Drives

The target portion of the data path for a Tape Drive is represented by the association from a **TapeDrive** instance to its **Controller** via a **ControlledBy** association. For SCSI tape drives there will be a **SCSILun** association to the **TapeDrive** subclass of **Controller**. A **ConcretelIdentity** association is used to connect the **Controller** with its corresponding **FCPort** in Fibre channel SCSI drive configurations. For non-SCSI **TapeDrive**'s, e.g. ESCON, the association will utilize **ControlledBy** to the appropriate subclass of **Controller**.

The agents for HBAs(initiator) having visibility to the tape drive's **SCSIController** will create a **SCSIInterface** association to the **TapeDrive** instance. These instances will link objects in different name spaces and be correlated by the Durable Name mechanism for **TapeDrive** devices. See the following section regarding correlation of control path and data path for Tape Drives.

3.3.4.2.2.3 Physical View

This perspective into a Storage Media Library shows the classes and associations that allow navigation and determination of the physical objects comprising the library and its devices.

A library's logical devices should each be associated with a physical element. The **Realizes** association serves this purpose and is the mechanism for tying logical and physical elements together. There are a variety of **PhysicalPackage** subclasses and the particular class used by an agent to represent each logical device is not specified by this profile.

3.3.4.2.2.3.1 Physical Media

The Physical Media within a tape library are determined through the use of the **PackagedComponent** association between a library's **Chassis** instances and the set of **PhysicalTape** instances. By navigating this association all physical tapes within a library may be determined, regardless of their location within the library. The **PhysicalMediaInLocation** association is optional in this profile but may be used to associate a piece of physical media with its current physical location.

A tape mounted in a Tape Drive is considered to be present within the library. Tapes located within **LimitedAccessPort** devices and **InterLibraryPort** devices should not be considered as present within the library as they are in a state of movement to/from the library. Consequently for **PhysicalTape** objects in these devices the **PackagedComponent** association should not exist.

3.3.4.2.2.3.2 StorageMediaLocation

Locations within a storage media library where physical media may be placed are modeled by the **StorageMediaLocation** class. Examples of storage media locations are slots(a.k.a. cells), media access devices and transfer devices. Each instance of a storage media location is tagged with one a type attribute containing one of these values. Two different kinds of associations may be used to navigate from a **LogicalDevice** to the corresponding storage media locations it serves.

1. For media access devices the **Container** association may be used to capture the relationship between a logical tape drive device and the storage media location of the drive itself. This association is important for being able to determine the physical media types compatible with the drive.
2. For media transfer devices the **DeviceServicesLocation** association is used to determine the various locations serviceable by the device.

This profile does not require that the location of physical media be represented. With an active library this information will be very dynamic. A partial implementation that may be useful would be to only provide instances for physical media mounted in Tape Drives as an indicator of physical tape availability and drive availability.

3.3.4.2.2.3.3 Capacity Constraints

Whenever appropriate the physical elements of a Library should be associated with instances of **ConfigurationCapacity** through the **ElementCapacity** association. Recommended instances would be for:

- Drive Capacity of the Library, i.e. the minimum and maximum number of physical tape drives that may be housed within the library. This should be represented through an instance of **ConfigurationCapacity** associated with each of the library's **Chassis** objects.
- Slot Capacity for a Library. This should be represented through an instance of **ConfigurationCapacity** associated with each of the library's **Chassis** objects. Summation of the relevant capacity attributes for each instance would provide the appropriate library totals.
- Slot Capacity for each **LimitedAccessPort** instance.
- Slot Capacity for each **InterLibraryPort** instance.

3.3.4.2.2.3.4 Media Location

3.3.4.2.2.3.5 Asset Information

Asset information for the library can be found from the physical elements associated with a library's **Chassis** objects and the physical elements that realize each of the logical devices. The **Product** class also maintains asset information that may be of interest and can be accessed through the **ProductPhysicalElement** association. Software asset information can be found through the **SoftwareElement** instances.

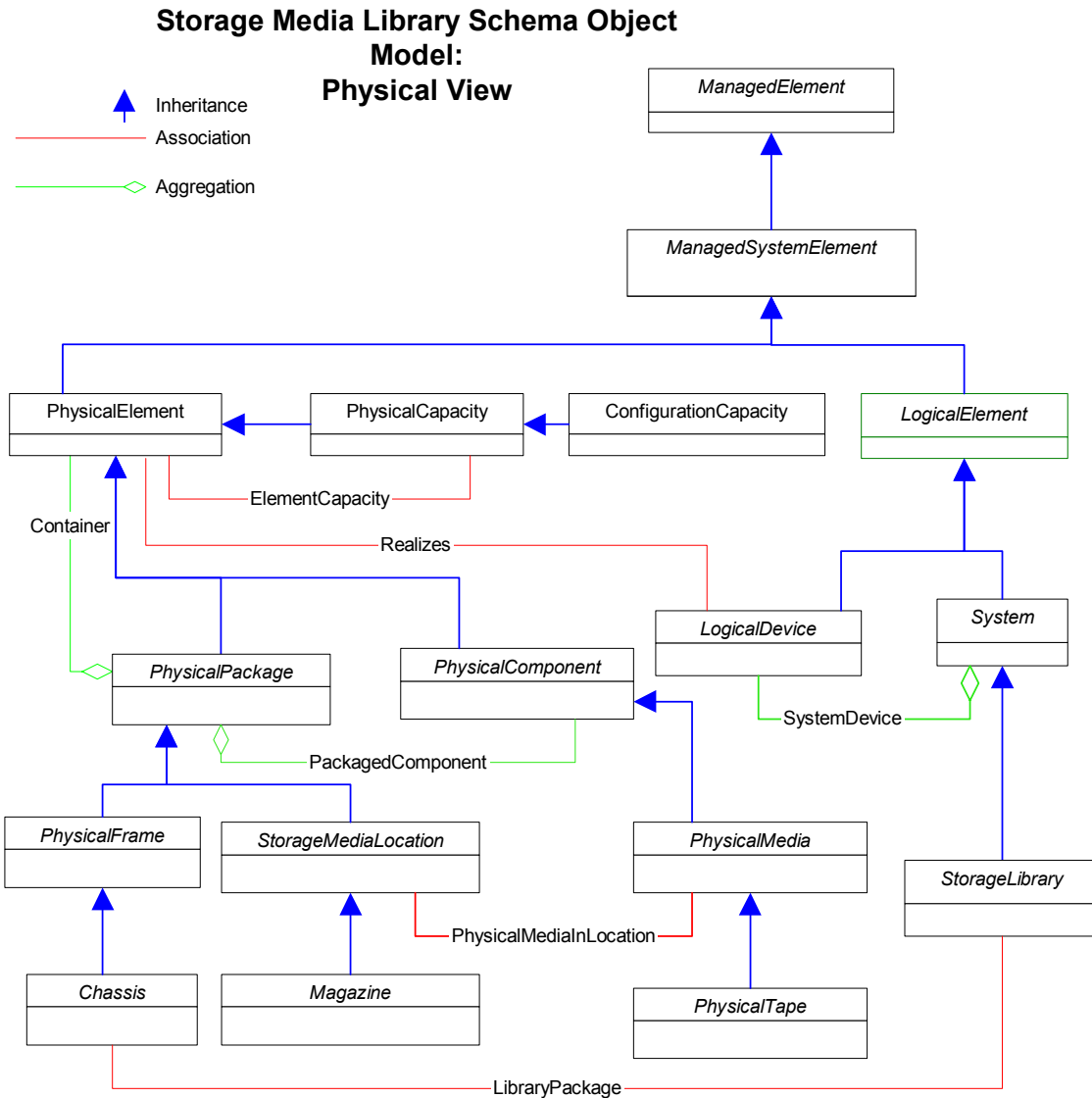


Figure 57: Storage Media Library Schema: Physical View

3.3.4.2.2.4 Software/Service View

Storage Media Libraries often contain manageable software elements. This perspective into the schema shows how various software elements and services are represented in the schema and their corresponding associations.

3.3.4.2.2.4.1 Device Firmware

The firmware for components of a Storage Media Library are pertinent to the logical devices found within the library. The devices that should have associated firmware information are the library's **ChangerDevice** instances and **TapeDrive** instances. The firmware information is captured in the **SoftwareElement** class that is associated with the corresponding logical device via an instance of **DeviceSoftware**.

3.3.4.2.2.4.2 Hosted Services

Software services that are provided by the Storage Media Library are represented by instances of **Service**. Examples of hosted software services within a Storage Media Library might be an NDMP server, SNMP agent, Web Server, Library Control Software as defined by the *IEEE Standard for Media Management System*, or proprietary services for library control. The specific services provided by a Storage Media Library are located through instances of the **HostedService** association originating from an instance of **StorageLibrary**.

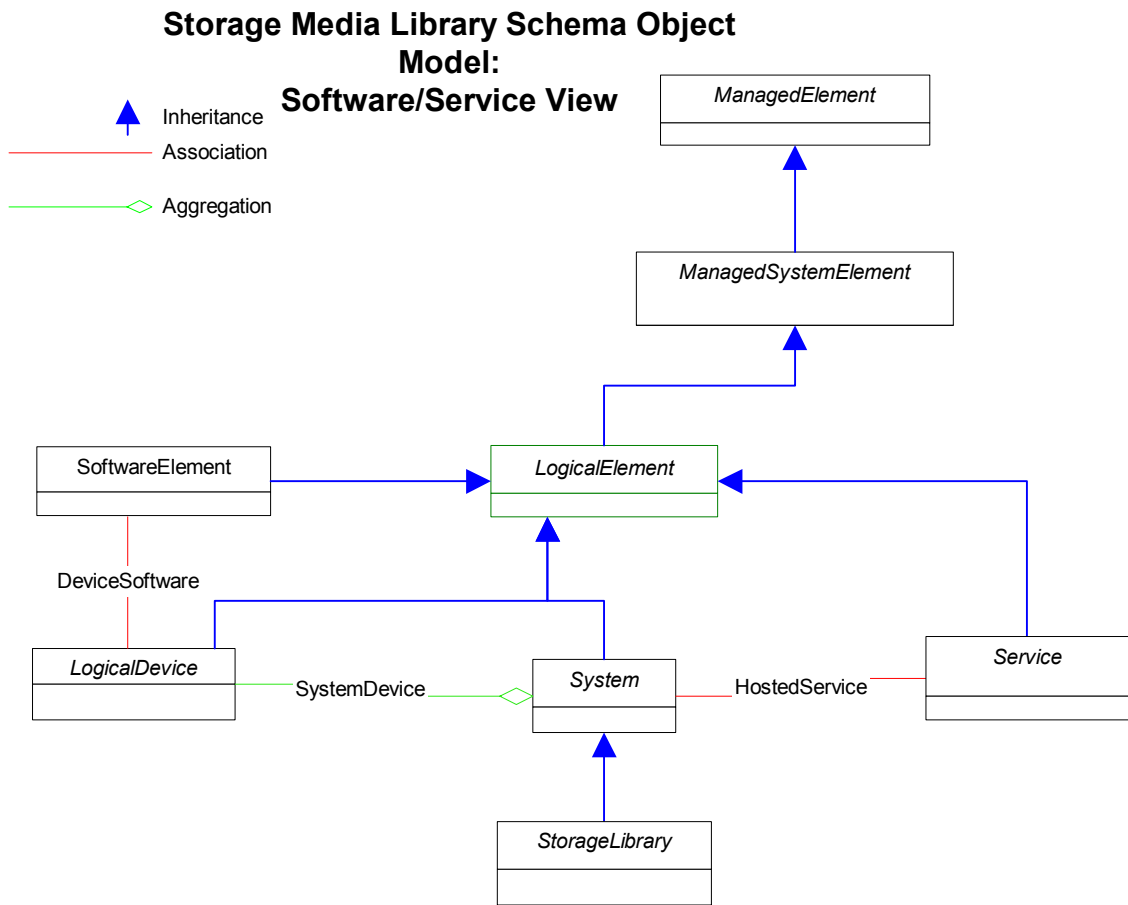


Figure 58: Storage Media Library Schema: Software/Service View

3.3.4.2.3 Instance Diagram

The following instance diagram represents a basic Storage Media Library with Fibre channel connections for the media changer and tape drive devices. The library contains a single physical tape which is mounted in the tape drive as depicted by the **MediaInLocation** association from the tape drive to a physical tape instance.

Storage media locations within the library are optional in the profile and consequently no “slot” type locations are depicted in the example.

Tape Library Instance Diagram

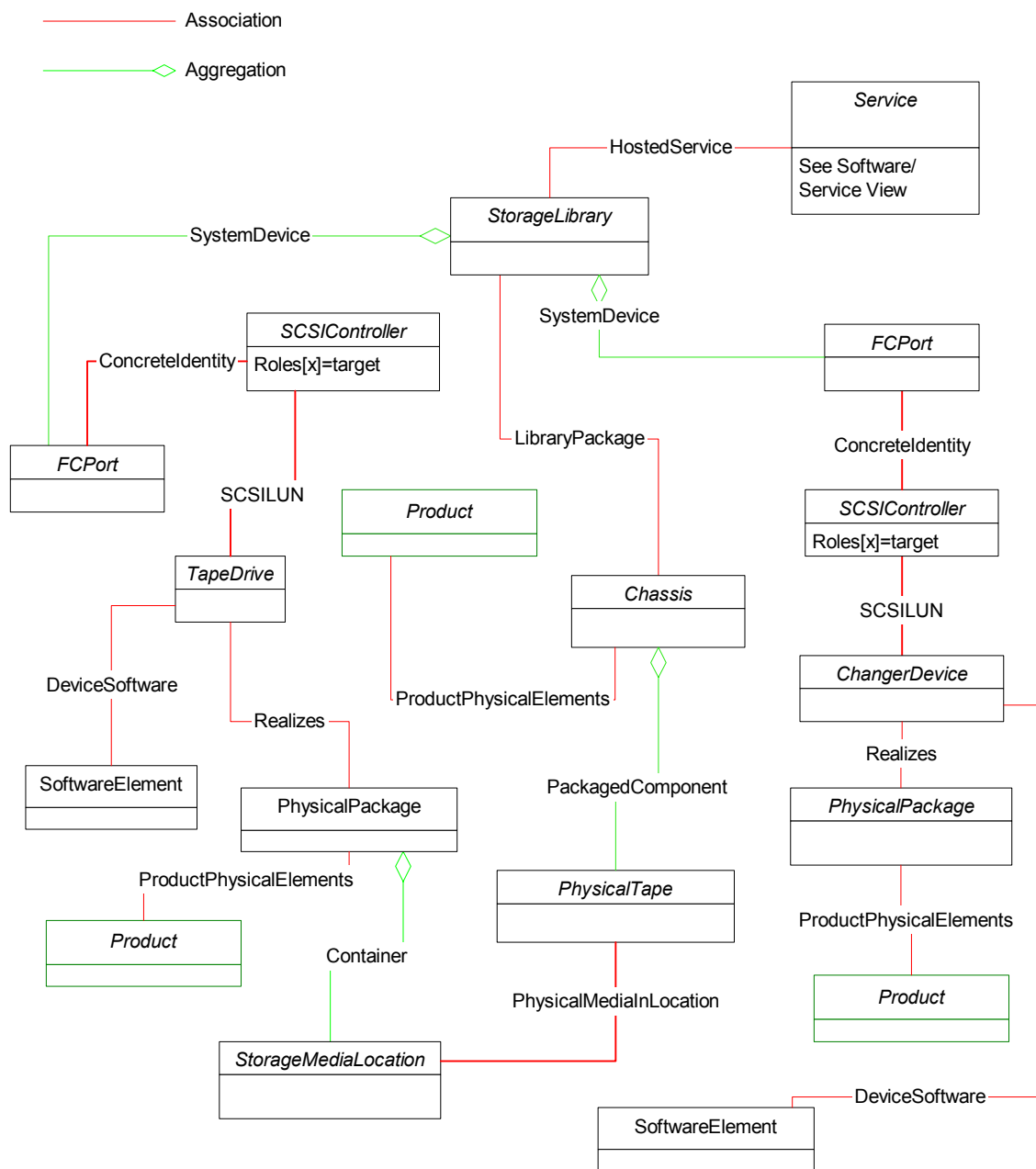


Figure 59: Tape Library Instance Diagram

3.3.4.2.4 Client Considerations

This section provides some helpful information for writing management applications and suggests techniques for addressing common use cases.

3.3.4.2.4.1 Discover a Storage Media Library

Discovery of Storage Media Libraries is achieved by looking up instances of **StorageLibrary**. **StorageLibrary** is subclassed from **System** and will have a corresponding **Name** and **NameFormat** attribute. **StorageLibrary** names are network host names (**NameFormat** = "IP"), node names (**NameFormat** = "NodeWWN") or Vendor+Model+SerialNumber (**NameFormat** = "VendorModelSerial").

3.3.4.2.4.2 Determine Library Physical Tape Capacity

The physical tape capacity of a library is the number of physical tapes that may be stored in the currently installed configuration of a Storage Media Library. This capacity may be determined by enumerating the number of **StorageMediaLocation** instances that are associated with each of the library's **Chassis** objects. Minimum and maximum slot capacities for a Storage Library are modeled in the **ConfigurationCapacity** described earlier in the section on Capacity Constraints.

Support for this use case is an optional part of the profile that may not be supported by each agent implementation.

3.3.4.2.4.3 Determine Physical Tape Inventory

For each **Chassis** instance associated with an instance of **StorageLibrary** via **LibraryPackage**:

- Enumerate the **PhysicalTape** instances via the **Chassis** instance's **PackgedComponent** associations.

3.3.4.2.4.4 Discover Tape 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 **SCSIController** instance associated via **SCSILun** to the **ChangerDevice** then the client should conclude that an alternate mechanism for controlling the library is required. This mechanism will 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

3.3.4.2.4.5 Determine Library Drive Capacity

The current drive capacity of a library may be determined by enumerating the **TapeDrive** instances through the **SystemDevice** association of the library. The number of drives discovered should be within the range indicated by the minimum and maximum capacity attribute found on the library **Chassis**' instances of **ConfigurationCapacity** for tape drives.

3.3.4.2.4.6 Determine Tape Drive Data Path Protocol

Most automated tape libraries will have a common data path protocol to all drives within the library. However, there are libraries that support mixed data path configurations, e.g. SCSI (parallel or Fibre Channel) and ESCON. Clients should be aware of this as there may not always be a **SCSILun** association on each **TapeDrive** instance within a library.

3.3.4.2.4.7 Durable Names for a Storage Media Library

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.

3.3.4.2.4.8 Find asset Information

Information about the entire tape library is modeled in the **Chassis** instances associated with the **StorageLibrary**. **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 **StorageLibrary** 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** association.

3.3.4.2.5 Agent Considerations

3.3.4.2.5.1 Tape Inventory

To be useful to client management applications, the agent for a Tape Library resource must accurately represent the required elements of a Storage Media Library and their proper state. In order to provide consistent Tape Inventory it is important that **PackagedComponent** association instances between a Storage Media Library's **Chassis** and **PhysicalTape** instances be maintained.

Entry and exit of **PhysicalTape** instances from the Storage Media Library will require updating this set of associations. Details on this procedure will vary from library to library but at a minimum will require an update each time a library is powered up. Other considerations involve updates whenever a **LimitedAccessPort**, **InterLibraryPort** or **Door** changes state.

3.3.4.2.5.2 Hosted Services

It is not uncommon for libraries to include the following services:

- **Web Server** – typically supports administration and configuration of the library.
- **SNMP Agent** – for resource monitoring and management within legacy System Management Frameworks, or even to support the Bluefin proxy agent.
- **NDMP Services** – NDMP may be present within the library to support the NDMP Backup Process

As an additional out-of-band management service, client management applications would be well served if they can locate these services via the agent's implementation of a corresponding instance of **Service**.

3.3.4.2.5.2.1 Media Changer Control Software

There are a variety of protocols for controlling tape libraries, the most predominant method being the SCSI Media Changer Commands defined by the NCITS' T10 Technical Committee. The ability to determine the type of control software required by a library is an important use case for clients. For this reason, it is imperative that the agent for a library resource instantiate the appropriate subclass of **Controller** for the **ChangerDevice** instances. Library vendors may subclass **Controller** for specifying proprietary library controllers for media changer devices. An example of a proprietary controller would be a StorageTek Library Management Unit for a PowderHorn 9310 tape library.

3.3.4.2.5.3 Mixed Media Libraries

This profile fully supports mixed media style libraries. A mixed media library is a library that supports **PhysicalMedia** with varying properties, e.g. DLT media as well as LTO media. The **StorageMediaLocation** class' **MediaTypesSupported** property specifies the type of media accepted. Agent developers should implement the **Container** association from a **TapeDrive** to a **StorageMediaLocation** so that there is a mechanism in place for determining media and drive compatibility.

3.3.4.2.5.4 Inter Library Ports

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.

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.

3.3.4.2.6 Indications

See the earlier section on “Events - CIM Indications” for details regarding the indications class hierarchy.

3.3.4.2.6.1 Instance Indications

It is highly recommended that Agents be designed to support CIM indications. The following guidelines present a starting set of mandatory indications and optional indications to be supported by library agents. Additional indications that offer more detailed alert type indications specific to libraries need to be defined in subsequent versions of this specification.

Tape Alerts are an example of a specification source for the formal definition of additional indications that should be mapped into CIM Indications specific to the Library profile.

3.3.4.2.6.1.1 Mandatory Instance Indications

The Library profile classes that should generate instance indications(**InstCreation** and **InstDeletion**) are listed below.

Mandatory **InstCreation/InstDeletion Instance Indications**

- **StorageLibrary**
- **PhysicalTape**
- **TapeDrive**
- **InterLibraryPort**
- **ChangerDevice**
- **LimitedAccessPort**
- **FCPort**

The following filter **Query** attribute values on **IndicationFilter** instances should be supplied by the agent for the above classes:

```
"SELECT * FROM CIM_InstCreate WHERE SourceInstance ISA ClassName"
```

```
"SELECT * FROM CIM_InstDeletete WHERE SourceInstance ISA ClassName"
```

Mandatory **InstModification Indications**

The library agent should create **InstModification** events whenever there is a change to the **OperationalStatus** attribute for the classes listed above. Refer to the appendix for the relevant status values that should be used by the agent for each of the classes. Additionally, the agent should also supply the following filter **Query** attribute value on **IndicationFilter** instances:

```
"SELECT * FROM CIM_InstModification
      WHERE SourceInstance ISA ClassName AND
      PreviousInstance.OperationalStatus <> SourceInstance.OperationalStatus"
```

Mandatory Alert Indications

Specific subclasses of **AlertIndication** are not defined by this profile. Consequently, agents do not need to create **IndicationFilter** instances for alert indications. However, agents are free to generate instances of this class of indications for vendor specific alerts, thresholds or SNMP Traps but clients should be aware that these are vendor specific events. Future versions of this specification will address this class of events for libraries.

3.3.4.2.6.1.2 Optional Instance Indications

Optional InstCreation/InstDeletion Instance Indications

The following classes are optional with respect to agent generation of events for instance creation and deletion:

- Chassis – only instances with **LibraryPackage** associations to the parent **StorageLibrary**.
- SoftwareElement
- Service instances that are in association with **StorageLibrary**

The following **Query** attribute on **IndicationFilter** instances should be supplied by the agent for the above classes:

```
"SELECT * FROM CIM_InstCreate WHERE SourceInstance ISA ClassName"
"SELECT * FROM CIM_InstDeletete WHERE SourceInstance ISA ClassName"
```

Optional InstModification Indications

The library agent should create **InstModification** events whenever there is a change to the **OperationalStatus** attribute for the classes listed above. Refer to the appendix for the relevant status values that should be used by the agent for each of the classes. Additionally, the agent should also supply a **Query** attribute on **IndicationFilter** instances that correspond to these modification events as follows:

```
"SELECT * FROM CIM_InstModification
      WHERE SourceInstance ISA ClassName AND
      PreviousInstance.OperationalStatus <> SourceInstance.OperationalStatus"
```

Optional Alert Indications

Alert indications are optional, but a starter set of interesting events are listed below:

- **StorageLibrary** instances indicating a need for an audit may have an alert generated. The following **Query** attribute on an **IndicationFilter** instance should be provided by the agent for this alert:

```
"SELECT * FROM CIM_Alert
      WHERE class=CIM_TapeDrive AND
      SourceInstance.AuditNeeded == true"
```

- **StorageLibrary** instances indicating a “full” condition may have an alert generated. The following **Query** attribute on an **IndicationFilter** instance should be provided by the agent for this alert:

```
"SELECT * FROM CIM_Alert
      WHERE class=CIM_TapeDrive AND
      SourceInstance.Overfilled == true"
```

- **TapeDrive** instances that require cleaning may have an alert indication generated whenever their cleaning attribute indicates a need to clean the drive. The following **Query** attribute on an **IndicationFilter** instance should be provided by the agent for this alert:

```
"SELECT * FROM CIM_Alert
      WHERE class=CIM_TapeDrive AND
      SourceInstance.NeedsCleaning == true"
```

- **PhysicalTape** instances for cleaning media may trigger an alert indication when the number of cleanings used exceeds the maximum cleanings supported by the media. The following **Query** attribute on an **IndicationFilter** instance should be provided by the agent for this alert:

```
"SELECT * FROM CIM_Alert
      WHERE class=CIM_PhysicalTape AND
      SourceInstance.CleanerMedia == true AND
      SourceInstance.MaxMounts > SourceInstance.MountCount"
```

3.3.4.2.7 Required Classes

The following table alphabetically lists the classes and associations from that are pertinent to Storage Media Libraries. Many of these classes have been indicated as optional for this Profile in order to simplify development of agents and client applications. Only CIM classes addressed by this profile are listed. Consequently, the list does not represent a complete set of Storage Media Library classes from the CIM 2.6 schema.

Some of the classes below are super-classes of others. If a subclass is indicated as “Required” then it follows that the super-class is also required. Many of the obvious super-classes and associations are not repeated in this table (e.g., **ManagedElement**).

Class	Req	Notes
ChangerDevice (p. 216)	Y	
Chassis (p. 217)	Y	
ComputerSystem (p. 218)	Y	
ConfigurationCapacity (p. 220)	I	
ControlledBy (p. 220)	N	
Controller (p. 221)	N	
DeviceServicesLocation (p. 223)	N	
DeviceSoftware (p. 223)	Y	
ElementCapacity (p. 225)	I	
FCPort (p. 227)	Y	
HostedService (p. 232)	N	
InterLibraryPort (p. 236)	N	
LibraryExchange (p. 236)	N	
LibraryPackage (p. 237)	Y	
LimitedAccessPort (p. 237)	N	
PackgedComponent (p. 241)	Y	
PhysicalMedia (p. 242)	Y	
PhysicalMediaInLocation (p. 243)	N	
PhysicalPackage (p. 244) (super)	Y	
PhysicalTape (p. 244)	Y	
Product (p. 245)	Y	
ProductPhysicalElements (p. 246)	Y	
Realizes (p. 247)	Y	subclass of dependency association from LogicalDevice to PhysicalElement
SCSIController (p. 249)	Y	
SCSIInterface (p. 250)	N	
SCSILUN (p. 251)	Y	
Service (p. 252)	N	
SoftwareElement (p. 257)	Y	
StorageLibrary (p. 276)	Y	
StorageMediaLocation (p. 275)	I	
StorageVolume (p. 287)	N	
SystemDevice (p. 288)	Y	
TapeDrive (p. 289)	Y	

Table 10: Required Classes for Tape

3.4 Cross Profile Considerations

3.4.1 Overview

Many client applications are required to 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.

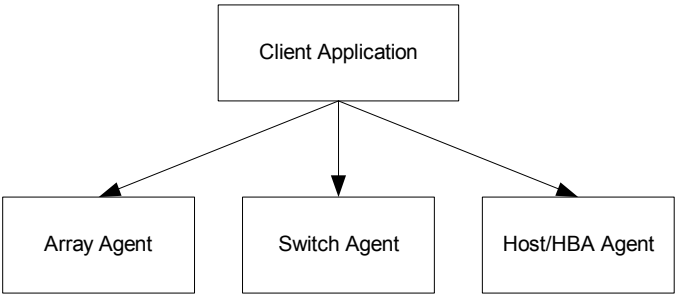


Figure 60: System Diagram

3.4.1.1 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.

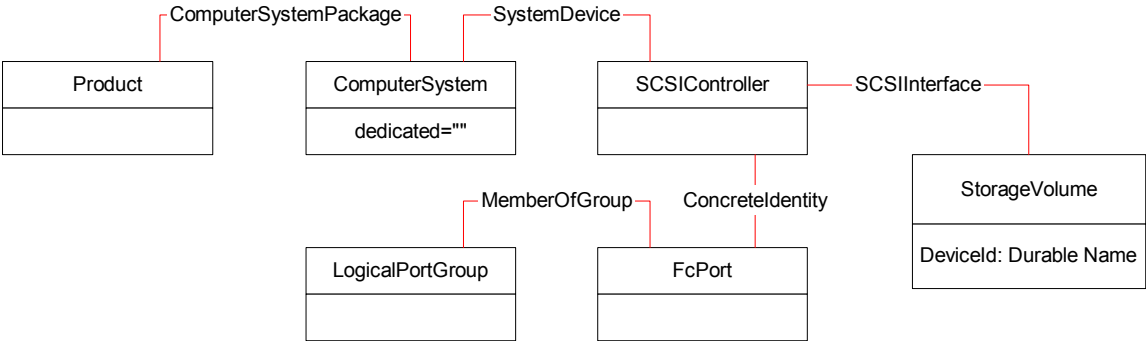


Figure 61: Host Bus Adapter Model

3.4.1.2 Switch Model

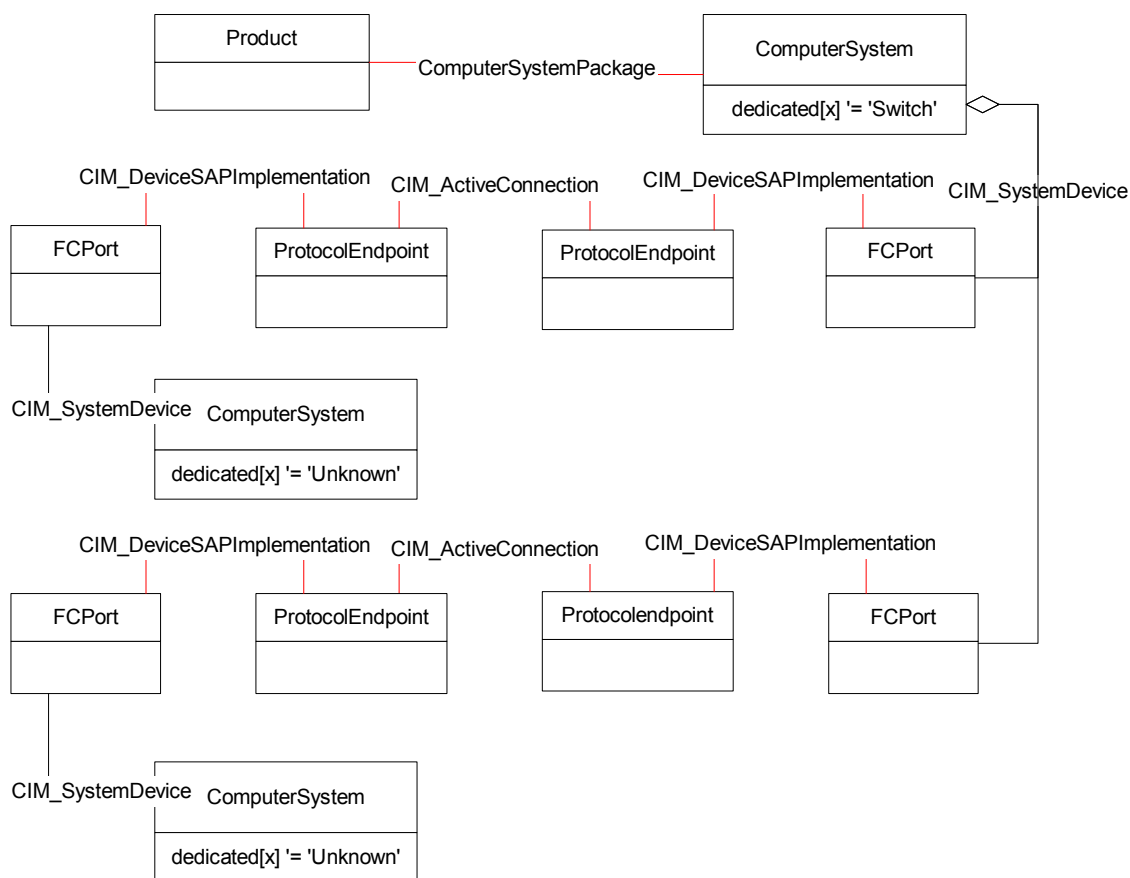


Figure 62: Switch Model

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** with the dedicated property set to “Unknown” which the **FCPorts** requires.

3.4.1.3 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.

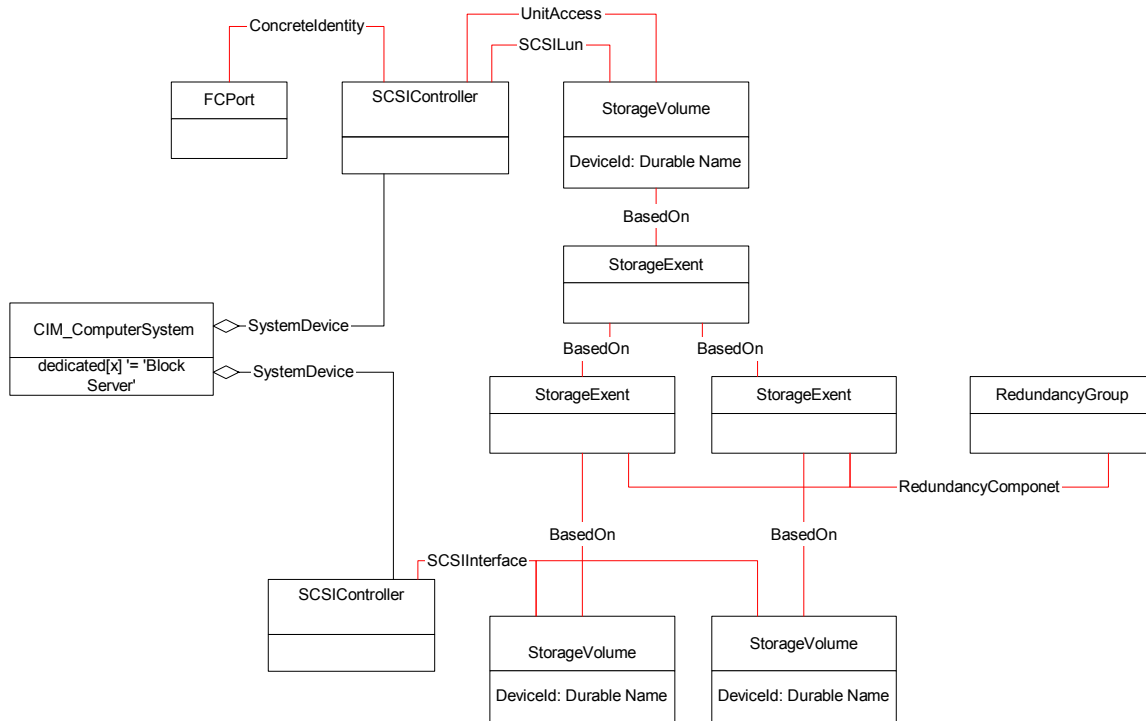


Figure 63: Array Instance

3.4.2 Fabric Topology (HBA, Switch, Array)

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. Bluefin 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 **SystemDevice** associations to **FCPort** objects and **SCSIController** objects. For each **SCSIController** object follow the **ConcretelDentity** associations to **FCPort** objects. Use the information in the **FCPort** objects found to determine the Durable Name for the **FCPort** object. The Durable Name will be 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 **SystemDevice** associations to the **FCPort** objects that represent the ports of the switch. Next look for **ActiveConnection** associations. These associations represent links between the switch and other elements. Follow the **ActiveConnection** associations to **FCPort** 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.

3.4.3 Durable Identifiers

Mapping objects across profiles and namespaces depends on “durable identifiers”. Below is a table of identifiers used in the examples below.

Class	Reference	Example
FCPort	Clause 3.2.3	WWN
StorageVolume	Clause 3.3.4.1.4	VPD page 83 LU id

Table 11: Cross Profile Durable Identifiers

3.4.4 Storage Connections (HBA, Array)

The Array profile includes objects and associations that represent the serving of SCSI volumes to the SAN. The HBA model represents the access of these volumes. To link them together locate the **StorageVolume** objects and use Durable Names to link them together.

To locate the volumes being accessed through the HBA use SLP to find agents that support the HBA profile. Use intrinsic methods to enumerate **ComputerSystem** objects on these agents. Use the “Dedicated” attribute to identify “host” systems. Use other attributes to identify the correct host. Next follow **SystemDevice** associations to **SCSIController** objects. The **SCSIController** objects represent the HBAs on the host. From the **SCSIController** objects follow **SCSIInterface** associations to **StorageVolume** objects. The **StorageVolume** object attributes are used to determine the Durable Name for the volume. The **SCSIInterface** association contains information to determine the SCSI address and server of the volume.

To find the array that is serving the volume use SLP to locate agents that support the array profile. Use intrinsic methods to enumerate **ComputerSystem** objects from the agents. Then use the “Dedicated” attribute to identify "Block Server" systems.". Use **SystemDevice** associations to find **SCSIController** objects. Then follow **ConcreteIdentity** associations to find **FCPort** objects. Match attributes in the **FCPort** object with information from the **SCSIInterface** attributes associated with the HBA. When a match is found use **SCSILun** and **UnitAccess** associations from the **SCSIController** object to locate **StorageVolume** objects. The **StorageVolume** objects can be matched to the **StorageVolume** objects from the Host/HBA profile using durable identifiers.

Clause 4: Security

4.1 Introduction

Security requirements can be divided into the five categories of authentication, authorization, confidentiality, and integrity (including non-repudiation). Authentication is verifying the identity of an entity (client or agent). Authorization is deciding if an entity is allowed to perform an operation. Confidentiality is restricting information to the intended recipients. Integrity is guaranteeing that information has not been modified.

This version of the specification is primarily concerned with authentication and confidentiality. Authorization is not covered and is implementation dependent. Valid implementations may omit the authorization check altogether (in which case any authenticated client can perform any operation), or they may perform an authorization check. When an authorization check is performed, the information concerning which client is allowed to perform which operations may be obtained from a local proprietary database, or from an authorization server using a standard protocol such as LDAP, or by some other means. Specification of an interoperable method for this is left for future work.

Other issues not covered by this specification are threat models and protection against specific types of attacks, such as denial of service attacks, replay attacks, buffer overflow attacks, etc. Integrity is not covered. The development of threat models and the specification of appropriate measures to counter the threats and to satisfy other security requirements such as integrity is left for future work.

Security concerns occur in two areas of a Bluefin implementation. First, a device such as a switch may require a login before discovery can be performed, or before operations such as zoning can be performed. The information needed to perform this login is generically referred to as “device credentials”. A Bluefin Agent or provider needs to obtain these credentials in order to talk to the device, and they should be provided confidentially.

Second, a Bluefin Client may need to authenticate itself to a Bluefin Agent or Object Manager. 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. Authenticating the client is the first step in determining what that Client is allowed to do.

4.2 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 SSL protocol is described in the IETF Internet Draft, “The SSL Protocol Version 3.0”.

The Transport Layer Security Protocol Version 1.0 (TLS) is defined by IETF RFC2246. Any references to SSL in this document also allow TLS implementations, as long as the TLS implementation incorporates a mechanism by which it can back down to SSL 3.0, for the sake of interoperability.

There are two levels of authentication described in section 4.4 of “Specification for CIM Operations over HTTP, Version 1.1”, basic and digest authentication. 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 SSL. 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. The specification states that CIM clients and CIM servers must not use basic authentication on an insecure network, and should support digest authentication.

SSL provides for confidentiality and integrity in communication. An initial handshake defines a private key, which is used to encrypt the data with a symmetric algorithm, such as DES or RC4. A keyed secure hash, such as SHA or MD5 is used to check message integrity. For interoperability, the initial handshake defines the algorithms to be used for message encryption and hashing.

4.3 Modeling Device Credentials

The device credentials are modeled using the CIM classes `SharedSecretService` and `SharedSecret` or `PublicKeyManagementService` and `UnsignedPublicKey` (see Figure 64 - Device Credentials). The `ComputerSystem` class represents the device, and the `SharedSecret` or `UnsignedPublicKey` object contains the credentials in its properties.

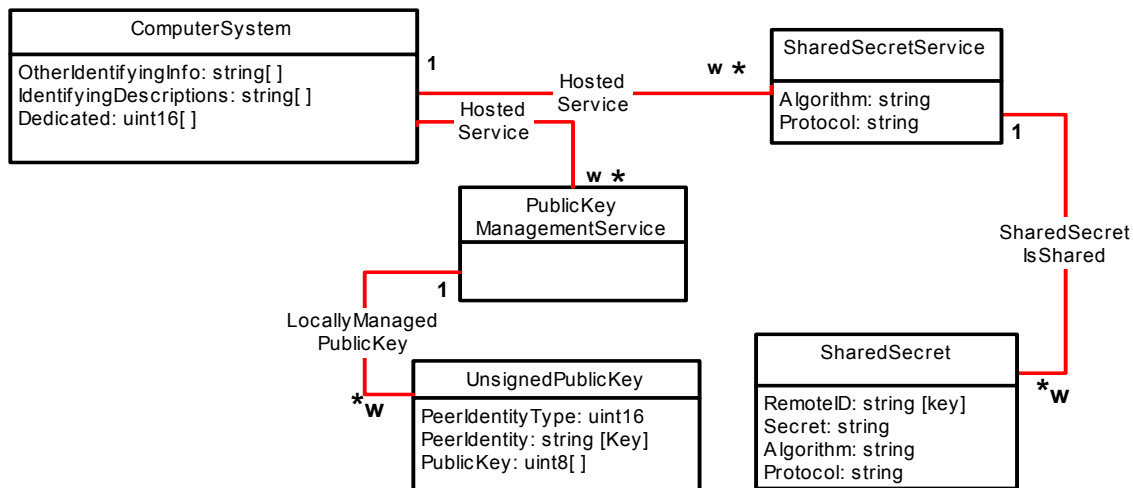


Figure 64 - Device Credentials

A Bluefin Client can pass the device credentials to the Agent or Object Manager by instantiating the `SharedSecret` object, using the CIM intrinsic method `CreateInstance()`. The Bluefin Agent or Provider uses the information from this object to talk to the device.

4.4 Requirements

Bluefin Agents, Object Managers and Clients **MUST** conform to section 4.4 of “Specification for CIM Operations over HTTP, Version 1.1”. In addition, Bluefin Agents, Object Managers and Clients **MUST** support Digest Authentication and **SHOULD** support SSL. This specification determines the protocol for authentication between a Client and the Object Manager or Agent, but not the mechanism of authentication used by the Object Manager or Agent.

Client authentication to the Object Manager or Agent is based on an authentication provider. A provider plug-in allows for differing authentication schemes. Possible mechanisms include host-based authentication, Kerberos, PKI, or other.

A Bluefin Agent **MAY** be configured with the device credentials necessary to talk to the device. If an Agent supports SSL, the Client **MUST** use SSL to pass device credentials to the Agent. When new device credentials are passed to an Agent, the device credential information in the device **MUST** be updated immediately.

Only the Bluefin Agent responsible for communicating with the device has access to the properties of the **SharedSecret** object. No other Bluefin Client may read the **Secret** property of this object as it **MUST** be implemented Write-Only.

4.5 Agent Considerations

The Agent **SHOULD** securely store the device credentials local to the Agent. A proxy agent may need to store the credentials on disk so that they are available upon reboot. In this case the credentials **SHOULD NOT** be stored in the clear, but **SHOULD** be encrypted for confidentiality.

The device credentials **SHOULD** be transmitted securely from the Agent 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.

Clause 5: Service Discovery

5.1 Definitions

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

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.

Directory Agent (DA): A process that caches SLP service advertisements registered by Service Agents and forwards the service advertisements to User Agents on demand.

SA Advertisement (SAAdvert): Information describing a service that consists of the Service Type, Service Access Point, lifetime, and Attributes.

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

Service Access Point: The network address and port number of a process offering a service.

Service Acknowledgement (SrvAck): A reply to a SrvReg request.

Service Agent (SA): A process working on behalf of one or more services to advertise the services.

Service Agent Server (SAServer): A process working on behalf of one or more Service Agents to listen on a particular port number for SLP service requests.

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: The class of a network service represented by a unique string (for example a namespace assigned by IANA).

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 Type Template: A formalized, computer-readable description of a 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.

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.

5.2 Overview

The Service Location Protocol Version 2 (SLPv2), defined by [IETF RFC2608](#), 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 must re-register as the registrations expire. If no Directory Agent is present, User Agents can request service information directly from the Service Agents.

5.3 SLP Messages

SLP v2 divides the base set of SLP messages into required and optional subsets. It also includes a new feature, an extension format. Extension messages are attached to base messages.

Service Agents (SAs) and User Agents (UAs) are required to support Service Request, Service Reply, and DA Advertisement message types. Service Agents must additionally support Service Registration, SA Advertisement, and Service Acknowledgement message types. The remaining message types are optional for Service Agents and User Agents. Directory Agents (DAs) must support all message types with the exception of SA Advertisement. Table 12 lists each base message type, its abbreviation, function code, and required/optional status.

Table 12: 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 Acknowledgement	SrvAck	5	R	R	O
Attribute Request	AttrRqst	6	R	O	O
Attribute Reply	AttrRply	7	R	O	O
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		R	O

5.3.1 Message Header

Each SLP v2 message is prefixed with a required, common header. A key field in the message header is a three-byte field that specifies the number of bytes from the beginning of the message header to the first extension message. Extension messages will appear after the message body. If there are no extension messages, the extension length field is zero.

5.3.2 Protocol Extension Block

The protocol extension block allows expansion of the protocol without modifying the base message set. The message header (3.5.1) contains an offset from the beginning of the message to the first protocol extension block, which always appears after the body of the message. The protocol extension block is optional. Service URL Entries

URLs are packaged in a service URL entry. The service URL is a required part of the protocol because the service URL entry appears in required as well as optional messages. If a service URL entry appears in a reply, it may not be truncated if the reply overflows the packet MTU. This ensures that any return is usable even if the entire reply will not fit in the packet. Error Conditions and Status Codes

Non-zero status codes are returned only for unicast messages. Multicast messages that result in errors are silently discarded. This means that a Service Agent never returns a non-zero status code to a User Agent unless the User Agent contacts it directly using a unicast.

Table 13 lists all of the status codes with a description of the condition each code represents.

Table 13: SLP v2 Status Codes

ERROR TYPE	STATUS CODE	DESCRIPTION
No error	0	The request was processed without an error.
LANGUAGE_NOT_SUPPORTED	1	There was no matching registration in the locale of the request, although there was at least one in a different locale.
PARSE_ERROR	2	The message contained a syntax error.
INVALID_REGISTRATION	3	The Service Agent's registration was invalid. Example include a zero lifetime or an omitted language tag.
SCOPE_NOT_SUPPORTED	4	A request containing a specific scope name was sent to a Directory Agent or Service Agent that does not support the scope.
AUTHENTICATION_UNKNOWN	5	The Directory Agent or Service Agent received a request for an SPI it does not support.
AUTHENTICATION_ABSENT	6	The Directory Agent did not receive expected URL and attribute list authentication blocks.
AUTHENTICATION_FAILED	7	The Directory Agent detected a verification failure in an authentication block.
VER_NOT_SUPPORTED	9	The version number in the message header is unsupported
INTERNAL_ERROR	10	This indicates a problem with the receiving agent rather than the received message. The Directory Agent or Service Agent returns this status when a serious internal error prevented it from fully processing the request. The error could be a bug in the agent or some system resource limitation.
DA_BUSY_NOW	11	The Directory Agent is too busy to respond. The Service Agent or User Agent should retry the message using exponential backoff.

OPTION_NOT_UNDERSTOOD	12	The Directory Agent or Service Agent received an unknown option from the mandatory range.
INVALID_UPDATE	13	The Directory Agent received an update for an advertisement that is unregistered or with an inconsistent service type.
MSG_NOT_SUPPORTED	14	This indicates a problem with the receiving agent rather than the received message. The Service Agent received an optional message type such as AttrRqst or SrvTypeRqst, and it only implements the required messages.
REFRESH_REJECTED	15	The Service Agent tried to refresh a service with a refresh interval less than the minimum bound contained in the Directory Agent's advertisement.

5.3.3 Required Messages

Service Agents and User Agents are required to support Service Request (SrvRqst), Service Reply (SrvRply), and Directory Agent Advertisement (DAAdvert). Service Agents are additionally required to support Service Registration (SrvReg), Service Acknowledgement (SrvAck), and Service Agent Advertisement (SAAdvert).

5.3.3.1 Service Request (SrvRqst)

A Service Request (SrvRqst) message is issued by a User Agent to find services. It may also be issued by a User Agent or Service Agent during active Directory Agent discovery.

In addition to a service location header field and two fields for the previous responder's list, the SrvRqst contains four string fields: service type, scope list, query, and SPI if security is on. The presence of the SPI string indicates the client is interested in only authenticated advertisements with the same SPI.

An advertisement must match the service type, one of the scopes contained in the scope list, the query, and the locale of the request to be included in the return. The service type can be any one of the following three: a concrete service type, an abstract service type, or a generic URL. If the service type is a concrete type or a generic URL scheme, a service advertisement matches only if the full service type or scheme name matches. If the service type is an abstract service type, then an advertisement matches if it has the same abstract service type regardless of its concrete type. See Clause 5.5.1.1.1 Abstract Service Type for an example of both concrete and abstract service type comparison and matching.

In addition to matching the service type, an advertisement must be registered in at least one of the same scopes as the SrvRqst. The query has the LDAP v3 syntax and is evaluated over the attributes of the advertisements registered in either the Directory Agent or Service Agent. The locale of the advertisement must match the locale of the request. The only exception to this requirement is if the service type in the query has no attributes and the query is empty (nanoSA implementation). In that case, the locale is disregarded and only the service type and scope are used to determine a match.

5.3.3.2 Service Reply (SrvRply)

The Directory Agent or Service Agent responds with a Service Reply (SrvRply) message to all Service Request (SrvRqst) messages except those with service types "service:directory-agent" and "service:services-agent".

When an SrvRply is sent by UDP, a URL entry is included only if it does not overflow the packet length. A User Agent may use the URL entry blocks in the reply if the reply overflows the packet, or it may contact the replying agent by TCP in order to obtain the full reply. If the request contained an SPI, the URL entries include authentication blocks for the URLs having that SPI.

The following error codes, specific to a Service Reply message type, may be returned:

LANGUAGE_NOT_SUPPORTED: The Directory Agent or Service Agent returns this error code if the agent contains an advertisement with a matching service type and at least one matching scope but the advertisement is registered in a different locale, excluding the dialect part, and no other advertisement in the locale of the request matches.

PARSE_ERROR: This error code is returned if a syntax error occurs in the service type name, query, scope list, or in the base message format.

SCOPE_NOT_SUPPORTED: This error code is returned if the scope list is omitted or contains scopes not supported by the Directory Agent or Service Agent (receiving agents).

AUTHENTICATION_UNKNOWN: This error code is returned if the request contains a SPI that the receiving agent cannot verify.

5.3.3.3 Service Registration (SrvReg)

The Service Registration (SrvReg) message type allows Service Agents to register new advertisements, update existing advertisements with new or changed attributes, and to refresh URL lifetimes. The reply message type to a SrvReg is a Service Acknowledgement (SrvAck).

Setting the F or “fresh” flag in the message header causes the registration to replace any existing registrations. Leaving the “fresh” flag off causes the registration to update existing registrations.

The service type field in the SrvReg message is required to match the service type of the service URL if the service URL is defined by the service: scheme. If the URL is defined by another scheme, the service type can be different, allowing registrations of generic URLs under service type names different from their scheme name.

Every SrvReg must contain a list of scope names in which the advertisement should be registered. Service Agents are required to register in all scopes with which they are configured. If no other scope is available, the Service Agent must use the default scope name “DEFAULT”.

If the Service Agent supports any SPIs, an authentication block is included for each SPI supported. The language tag in the header of the SrvReg establishes the locale of the service advertisement and must be recorded for matching purposes.

The SrvReg message may also cause incremental updates if the “fresh” flag in the header is not set. The rules for incremental update are:

- A new service URL lifetime replaces the old.
- New attributes are added to the advertisement.
- Old attributes with new values specified in the message have their old values completely replaced by the new values.
- Old attributes that are not mentioned in the message are unchanged.

Incremental updates of an authenticated advertisement are not allowed because authentication blocks cannot be recalculated by the Directory Agent for updated advertisements. All SrvReg messages for exiting authenticated advertisements must have the “fresh” flag set.

The following error codes, specific to a Service Registration message type, may be returned:

PARSE_ERROR: This error code is returned if a syntax error occurs in the URL, attribute list, or message format.

INVALID_REGISTRATION: A problem occurred with a registration that has the “fresh” bit set. Examples of such problems include: a zero lifetime for the URL, no language tag in the header, a mismatch between the service type field and service: URL’s service type, or the Service Agent’s attempt to register the same URL under two different service type names.

AUTHENTICATION_UNKNOWN: The Directory Agent received a registration contain an SPI that it does not support.

AUTHENTICATION_FAILED: The Directory Agent tried to authenticate the registration, but the verification failed. This error is also returned when a Directory Agent receives an incremental update for an authenticated advertisement.

INVALID_UPDATE: The same problems listed in INVALID_REGISTRATION occurred except that the “fresh” bit was not set in the header. In addition, the following will generate this error: no existing advertisement to update, the service type string in the update does not match the string in the original registration, or the scopes do not match the original scopes. To change the service type or scopes, the Service Agent must first deregister, then re-register the advertisement.

5.3.3.4 Service Acknowledgment (SrvAck)

A Directory Agent returns a Service Acknowledgement (SrvAck) message to a Service Agent in response to a Service Registration (SrvReg) or Service Deregistration (SrvDereg) message. The only field other than the required header is a two-byte status code.

5.3.3.5 Directory Agent Advertisement (DAAdvert)

A Directory Agent responds to multicast Service Requests (SrvRqsts) for service type “service:directory-agent” with a Directory Agent Advertisement (DAAdvert) message. It is one of two cases where the response to an SrvRqst is not a Service Reply (SrvRply) with the other case being a Service Agent solicitation. In addition to the solicited DAAdverts previously described, unsolicited DAAdverts are also periodically multicast and received by Service Agents and User Agents performing passive Directory Agent discovery.

The status code field is set to SLP_OK in unsolicited DAAdverts that are multicast, but it can be set to an error code in response to a unicast SrvRqst for “service:directory-agent”. A multicast SrvRqst for “service:directory-agent” that results in an error is simply dropped. Possible error codes are the same as for SrvRqst.

The “DA stateless boot timestamp” field lists the time of the last stateless boot for the Directory Agent. A receiving Service Agent will use this value to determine if it needs to register with the Directory Agent.

The “DA URL String” field contains the Directory Agent’s URL information consisting of a service: URL with the Directory Agent service type, “service:directory-agent”, and the host address of the Directory Agent in IP v4 dotted decimal notation. If the DAAdvert was multicast the Directory Agent URL may be from a different service scheme, in which case the receiving agents will drop it.

The DAAdvert will always contain a list of scopes supported by the Directory Agent. The scope list is never empty. An optional short list of attributes may be included.

If security is on, a list of SPIs that the Directory Agent can verify are included. Authentication blocks containing the SPIs for verifying the DAAdvert are also included.

5.3.3.6 Service Agent Advertisement (SAAadvert)

User Agents employ the Service Agent Advertisement (SAAadvert) message to discover Service Agents and their scopes in networks where there are no Directory Agents. A User Agent will multicast a Service Request (SrvRqst) for service type “service:service-agent” if the User Agent is not configured with scope or it cannot find a Directory Agent. In reply, the User Agent will receive a list of SAA adverts. A User Agent may also contact a Service Agent through unicast using the Service Agent’s address from the Service Agent URL.

The fields of the SAAadvert are identical to the DAAadvert. The URL field contains the SAAadvert service: URL with service type “service:service-agent” and the Service Agent’s host address in IPv4 dotted decimal notation. The Service Agent should support at least the attribute “service-type” so User Agents can query for SAA adverts by service type.

5.3.4 Optional Messages

The Service Deregistration, Service Type Request, Service Type Reply, Attribute Request, and Attribute Reply message types were made optional in SLP v2 to encourage lightweight, embedded implementations. The messages are optional only for Service Agents and User Agents. They are required for Directory Agents.

5.3.4.1 Service Deregistration (SrvDereg)

The Service Deregistration is optional because an advertisement times out when its lifetime expires if the Service Agent does not deregister it. In that case the advertisement is available via the Directory Agent longer than it would be if explicitly deregistered. If a Service Agent implements SrvDereg, it should deregister its advertisements when the service represented is no longer available so that service location information is accurate.

The attribute query list contains an attribute query for selecting attributes to remove. If the list is empty, the entire advertisement is removed. An advertisement that is deregistered completely is deregistered in all scopes and for all locales under which the URL appears. If attributes are removed, the advertisements having attributes that match the attribute query are affected regardless of locale, but the attributes are deleted in all scopes. If the advertisement contains an attribute authentication block, deletion of individual attributes is prohibited because attribute deletion would invalidate the authentication block associated with the advertisement.

If the Service Agent uses UDP, the Service Agent must retry the SrvDereg if the Directory Agent does not respond. A Directory Agent acknowledges an SrvDereg with a Service Acknowledgement (SrvAck) message.

The following error codes, specific to the Service Deregistration message type, may be returned:

PARSE_ERROR: A syntax error occurred in the URL, attribute query, or message format.

INVALID_REGISTRATION: A non-syntax error occurred in the message. For example, an omitted language tag or if the URL is not registered.

SCOPE_NOT_SUPPORTED: The Service Agent attempted to deregister with a scope list that is different from the list used to register the advertisement.

AUTHENTICATION_UNKNOWN: The URL contained SPIs not recognized by the Directory Agent.

AUTHENTICATION_ABSENT: The advertisement contained an authentication block of the URL, but none was included for the SrvDereg, or the reverse occurred.

AUTHENTICATION_FAILED: The authentication block failed to validate. Also, if a SrvDereg received for an authenticated advertisement includes an attribute query, the Directory Agent returns this error.

5.3.4.2 Service Type Request (SrvTypeRqst)

The Service Type Request (SrvTypeRqst) message type allows inquiries about registered service types that have a specific naming authority and are in a specific set of scopes.

5.3.4.3 Service Type Reply (SrvTypeRply)

The Service Type Reply (SrvTypeRply) message is returned in response to a Service Type Request (SrvTypeRqst) message. The service type names returned are full service type names, including the “service:”, unless the service type is a generic URL scheme name.

A diagram of the format for the SrvTypeRply message type may be found in [].

The following error codes, specific to the SrvTypeRply message type, may be returned:

PARSE_ERROR: A syntax error occurred in the naming authority name, scope list, or message format.

SCOPE_NOT_SUPPORTED: None of the scopes in the scope list was recognized by the Directory Agent or Service Agent.

5.3.4.4 Attribute Request (AttrRqst)

Attribute requests may be made either by URL or service type. If the requested service type has service URLs from the service: scheme, the string “service:” must be included. Abstract type names are also allowed, as are generic URL scheme names. If the service type name is an abstract type name, the result will contain all concrete types.

5.3.4.5 Attribute Reply (AttrRply)

The Attribute Reply (AttrRply) message returns the results of an Attribute Request (AttrRqst) message. Results returned in the AttrRply depend upon the contents of the attribute query field and service type or service URL field in the AttrRqst responded to. Service advertisements are candidates for return processing if one of the advertisement scopes matches a scope in the AttrRqst. If an attribute query is present, the AttrRply returns values for only those attributes with tags matching the tag patterns in the query. If the attribute query is empty, values for all attributes are returned.

If the AttrRqst is by service URL, attribute values for attributes with tags matching the query are returned just for that URL. If the AttrRqst is by service type, all attribute values for attributes with tags matching the query across all advertisements registered under that service type are returned. If the service type is an abstract type without any concrete type, then the attribute values are returned for all matched advertisements having that abstract type regardless of concrete type.

An AttrRqst by URL matches only if there are advertisements in the locale of the request. An AttrRqst by type ignores the locale of the request and all attributes for all locales are returned.

The following error codes, specific to the AttrRqst message type, may be returned:

LANGUAGE_NOT_SUPPORTED: The Directory Agent or Service Agent returns this error code if the agent contains an advertisement with a matching service type and at least one matching scope but the advertisement is registered in a different locale, excluding the dialect part, and no other advertisement in the locale of the request matches. This error is only returned for AttrRqst by URL.

PARSE_ERROR: A syntax error occurred in the URL, service type, or in the base message format.

SCOPE_NOT_SUPPORTED: This error code is returned if the scope list is omitted or contains scopes not supported by the Directory Agent or Service Agent (receiving agents).

AUTHENTICATION_UNKNOWN: This error is returned if the request contains an SPI that the receiving agent cannot verify.

AUTHENTICATION_ABSENT: This error code is returned if security is on and the request does not contain an SPI (or the reverse).

AUTHENTICATION_FAILED: The receiving agent returns this code if the request contains an SPI and is by type or contains an attribute query string.

5.4 Scopes

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 will only discover those services which 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".

There are two models of scope discovery: the administrative scope discovery model and the dynamic scope discovery model. An SLP agent will employ only one of the models.

5.4.1 Administrative Scope Discovery

Configuration of an SLP agent through DHCP or a static configuration file is using the administrative scope discovery model. In this model, a network administrator sets the scopes an agent see and no other scopes are accessible.

Directory Agents and Service Agents employ the administrative scope discovery model because they required to be configured with a scope. This requirement enables those agents to reply definitively to a request from a Directory Agent or User Agent with a list of scopes that they support.

5.4.2 Dynamic Scope Discovery

User Agents can support either the dynamic model or the administrative model, depending on the application. If a User Agent is not configured with any scopes, available scopes are discovered through Directory Agent or Service Agent discovery, which is the dynamic scope discovery model. If no scopes are discovered, the scope "DEFAULT" is used.

5.5 Services Definition

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 an 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.

5.5.1 service: URL

The **service:** Scheme defines a mechanism for creating new service type definitions without creating a new URL scheme. The **service:** URLs must conform to the generic URL grammar defined in RFC 2396.

A URL of the **service:** Scheme definition must provide a service type name and an address for locating the specific instance of the service. Optionally, the **service:** URL may include a URL path containing additional information useful in accessing the service being advertised. The URL path may be suffixed with a list of SLP attributes.

5.5.1.1 Service Types

Service types group network resources by function or service provided. The service type name will uniquely identify each type of service. Service types may either be abstract or concrete. Abstract types will aggregate different concrete service types that perform the same function.

5.5.1.1.1 Abstract Service Type

Abstract service types provide for the grouping of several concrete service types into one service type. An abstract service type will allow a User Agent that supports many concrete types to perform a single service request and receive replies that utilize more than one concrete service type. As such, the service type name of a concrete type URL consists of three components:

“service:” abstract-type-name “:” concrete-type-name

All abstract types must be followed by a service access information component which will specify how a client is to connect to a service.

A service called “wbem” is an example of an abstract type. In this case, the User Agent would make service requests for the “wbem” service. Any of the following URLs would be valid responses:

service:wbem:XMLserver://device1.domain.com

service:wbem:Export://device1.domain.com

service:wbem:LM://device2.domain.com

service:wbem:LM://device3.domain.com

To name a specific concrete protocol as part of a service request, the request must be made using the full service type name as well as the abstract type name. For example, a User Agent wishes to discover what resources are managed by the Lock Manager (LM), would issues a service request for the service type “service:wbem:LM” specifying both an abstract service type and a concrete implementation. Based on the earlier example, the URLs returned are:

service:wbem:LM://device2.domain.com

service:wbem:LM://device3.domain.com

The responses received are limited to those managed by the Lock Manager (LM).

5.5.1.1.2 Naming Authority

The service type name may include a naming authority, which is the group, or organization that formulated the service URL and service type template. The naming authority for a service type is designated by appending a string to the concrete or abstract service type. The naming authority is separated from the base type by a period. The naming authority is IANA. The specification assumes that the DMTF has received the abstract type “wbem” from IANA and that the DMTF interop working group has assigned the concrete service types. This specification does not include the naming authority in the service type name.

5.5.1.2 Service Access Information

The service access information port of the URL consists of the following:

“/” address-family “/” address-spec [“/” [url-path] [“,” attribute-list]]

The address-family and address-spec are required fields. The url-path and attribute-list fields are optional.

5.5.1.2.1 Address Family

The address-family indicates the network layer protocol to be used and dictates how the address-spec should be interpreted. An empty address-family field is denoted by a double slash, “//”, and indicates the address that follows is an IP address. This specification uses IP addresses exclusively, so the address family field is left empty.

5.5.1.2.2 Address Specification

The address specification will provide the network layer address for the device on which the service resides. The address-spec must conform to the addressing mechanism defined in the address family. For IP hosts, the address specification is either a host-name or the dotted decimal representation of an address. An IP address may contain an optional port number.

5.5.1.2.3 URL Path

The URL path is protocol-specific. The protocol that is being used with the URL will determine how the URL path is interpreted. For example, in the case of file access protocols, the url-path represents the filename and path on the server.

5.5.1.2.4 Attribute List

The attribute list is an required list of attribute tags and their corresponding values for an instance of a service.

The attribute-list is a semi-colon delimited set of attributes/value pairs, set off from the url-path by an initial semi-colon. Each attribute/value pair has the following form:

attribute-id “=” value

5.5.1.3 Generic URL Schemes

Service: scheme URLs begin with the string “service:” and associate the service type with a service template. Generic URLs can be registered as well, though they cannot be standardized with a template. An example is the URL:

<http://www.domain.org/management.html>

The attributes registered with this URL depend on what service it is advertising, but are not necessarily standardized. If the URL advertises a web page, the attributes will be different than if it advertises a printer. A storage device can use this to advertise its proprietary web management scheme, but the Generic URL Scheme is not utilized by this specification.

5.5.2 Service Type Templates

A service type template will define the attributes and **service:** URL syntax for a specific service type. The service template will be both readable by humans and able to be parsed by software.

Each attribute defined will include information about its data type and characteristics. The characteristics will indicate if an attribute is required or optional and the allowable values for the attribute. A description of the attribute is also included.

5.5.2.1 Service Type Template Syntax

Service templates may be created in any language, but the language used for standardization must be English. Although any legal UTF-8 character may be used in a template, a service template must be encoded using the 0x00-0x7F range of UTF-8 character encoding. This range corresponds to the US-ASCII character set.

In order to accommodate UTF-8 characters outside the range used by the US-ASCII encoding an escape sequence may be used to represent those out-of-range UTF-8 characters. A character outside the US-ASCII range is escaped using a “\ HEXDIGIT HEXDIGIT” format. For example the accent aigue would be represented as “\c3\a9”.

5.5.2.2 Template Description Attributes

The template begins with the “template-type” items, containing the service type scheme. This is the service type name without the “**service:**” prefix. In addition, a service template contains the following items:

1. Version
2. Description
3. URL syntax
4. Service-specific attributes

The first three items will describe the text of the template itself and must appear in this order. These items serve to introduce the rest of the template. The remaining items are service attributes, which describe the service advertisement.

5.5.2.2.1 Version

The version item provides version control within document definitions. When templates are initially formulated, they begin with a version number of 0.0. During the development of a template, the number to the right of the decimal (minor number) is incremented. Once the template is complete and standardized, it is assigned a version number 1.0.

As the template evolves, the version number changes to reflect additions and deletions. Addition of new optional attributes cause the minor number to be incremented. The addition of required attributes, the change in definition of an attribute, or the removal of an attribute all result in the major number being incremented.

5.5.2.2.2 Description

The description item is a section of human-readable text. It should provide a brief but informative description of what the service defined by the template does. If the service type is a protocol, the protocol specification may be referenced here. Protocols listed here do not need to be Internet standards; they may be proprietary protocols as well. URL Syntax

The URL syntax item describes the structure of the url-path for the service URL. The syntax is included in the template document using ABNF and follows the rules URL syntax described in RFC 2396.

Abstract service types defer this field to the template documents describing the concrete instances. In most cases, the url-path is dependent on the underlying protocol. As such, the concrete types should define url-paths for their specific instances.

5.5.2.3 Service Attributes

The remainder of the template will contain an attribute definition list that defines a service advertisement for the service. Attribute names and values will be chosen to represent those characteristics of the service that are most useful in selecting a service either programmatically or interactively. The attributes should allow a distinction to be made between two distinct network entities that provide similar, yet distinct services.

An attribute definition begins with the specification of the attribute's identifier and ends with a single empty line. Service-specific attribute definitions includes the following five parts, appearing in order after the attribute identifier:

1. The data type associated with the attribute
2. Attribute flags
3. A list of default values
4. A block of text
5. A list of allowed values

5.5.2.3.1 Attribute Identifier

The attribute identifier functions as the name of the attribute and must be unique within a template. It is acceptable for an attribute name to be reused with different definitions across multiple templates, but within a template an attribute name must appear only once.

Legal characters for an attribute identifier are limited to those characters that are valid LDAP v3 attribute tag characters. The character selection for attribute identifiers is further restricted if the attribute is included in the service: URL. All attribute names are case-folded, with uppercase and lowercase equivalent when differentiating attribute names.

5.5.2.3.2 Data Type

The attribute name is followed by a space, and equal sign, another space, and the name of a data type. Attributes must be one of five types: Boolean, integer, string, opaque, or keyword. The data type of a value restricts the attribute flags that may be applied as well as the legal values assigned.

5.5.2.3.2.1 Boolean

Boolean attributes can only be assigned the values TRUE or FALSE. Boolean values are case-folded. The allowed values field is unnecessary since Boolean attributes may have only one of two possible values. Boolean attributes have a default value of FALSE unless otherwise specified in the template.

5.5.2.3.2.2 Integer

Integer attributes have values that are either zero or positive or negative whole numbers that can be represented in 32 bits. There is no support for numerical attributes whose value is a floating-point number. Floating-point values must be stored as an opaque or string type. Integer attributes have a default value of zero unless otherwise specified by the template. Embedded white space is not permitted for integer attributes.

5.5.2.3.2.3 String

String attributes have values that are represented by a series of characters. String values are considered a sequence of nonwhite space tokens separated by white space. Strings are not delimited by double quote or null characters.

For query handling, SPL reduces runs of interior white space characters to a single US-ASCII space. Preceding or trailing white space is also removed. The SLP specification recommends that no processing be done to string values when returning to applications.

5.5.2.3.2.4 Opaque

Opaque attributes are values stored as an array of bytes. The value stored in an opaque attribute is not assumed to be consumed directly by a user, other than through client software that interprets the byte array returned in an Attribute Reply message.

An opaque attribute, by default, is an array of zero size unless otherwise specified by the template. An opaque attribute value begins with the non-UTF-8 escape sequence “\ff” and each byte of the opaque is translated into the escape sequence ‘/’ HEX HEX, where HEX HEX are hexadecimal digits for the byte.

5.5.2.3.2.5 Keyword

Keywords are attributes that have no value. Keywords consist of only a name and, optionally, some descriptive text that defines them. Keywords have no default value and are never present unless specified at registration time.

5.5.2.3.3 Attribute Flags

Four flags are used to indicate the characteristics of an attribute. The flags are represented by a single letter that may be either uppercase or lowercase. The convention in service templates is to use uppercase for all flags.

‘O’: The O flag indicates the attribute is optional. If this flag is not present, the attribute is required in every service registration.

‘M’: The M flag is used to indicate that an attribute may have multiple values. When multi-valued, all of an attribute’s values must be of the same data type as specified in the data type field.

‘L’: The L flag indicates that an attribute and its value are literal. It is not intended for translation into other languages. This may apply to either translation of the template to another language or the application of the template.

‘X’: The X flag indicates that clients should include the specified attribute in all requests for services. Omission of this attribute may not sufficiently differentiate one instance of a service from another. If services are obtained without utilizing this attribute as part of the query, the selected service may not be appropriate.

5.5.2.3.4 Default Values

The default value or list of values indicates the values associated with an attribute in the absence of the attribute in a service registration. For required attributes, the default value list dictates the value or values assigned to the attribute when no values are assigned to the attribute during a service registration.

5.5.2.3.5 Descriptive Text

This optional text, in human-readable format, includes a brief, informative description of the attribute. It may describe the relation of the attribute to the service, include a definition of the attribute, or provide suggested uses for the attribute. It is primarily designed for display by interactive tools.

The descriptive test is set off from the surrounding definition by a pound sign or cross-hatch character (“#”) at the beginning of each line. Line breaks at the end of each line are preserved. Within the block of descriptive test, all indentation and formatting are also preserved. The text should never be treated as a comment when being parsed because it is an integral part of the attribute definition.

5.5.2.3.6 Allowed Values

If the allowed values list is present, assignment to attributes is restricted to members of the list. Attributes with an allowed values list are required to have default values or a default values list.

The syntax of the allowed values list is identical to the default values list. The handling of string and list of strings, along with the mechanism of escaping newlines, is consistent with the specification of the default values list.

5.6 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 (Clause 5.7) or Directory Agents (Clause 5.8). Further description of a Client and its role may be found in Clause 7.2 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 MUST issue requests as Directory Agents are discovered.

5.6.1 Configuration

UAs can be configured with DA information either statically or by DHCP. If a UA is not configured by either of these methods, it MUST actively discover Directory Agents (DAs). Otherwise, the UA may get the wrong set of services. If active Discovery of DAs fails, then a UA MUST actively discover Service Agents (SAs).

5.6.2 Discovery of Directory Agents and Service Agents

User Agents (UAs) that have been configured with Directory Agent (DA) information must unicast to communicate with those DAs. If a UA has not been configured with DA information, the UA should do active discovery of DAs using the Multicast Convergence Algorithm (MCA). The Multicast Convergence Algorithm specifies when to stop multicasting requests (see RFC2608, Section 6.3).

UAs can do passive discovery by listening for unsolicited multicast DA Advertisements. This allows UAs to immediately discover DAs that have come online since the last active discovery. UAs can also do periodic active DA discovery as a means of discovering new DAs. See Figure 65 for an example of both active and passive DA discovery.

All UAs that receive the unsolicited DAAdvert SHOULD examine its DA stateless Boot Timestamp. If it is set to zero (0), the DA is going down and no further messages should be sent to it.

A UA that cannot discover any DAs with the proper scope must multicast requests to SAs using multicast convergence. The technique will be the same as active DA discovery except only SAs will respond with SAAdverts.

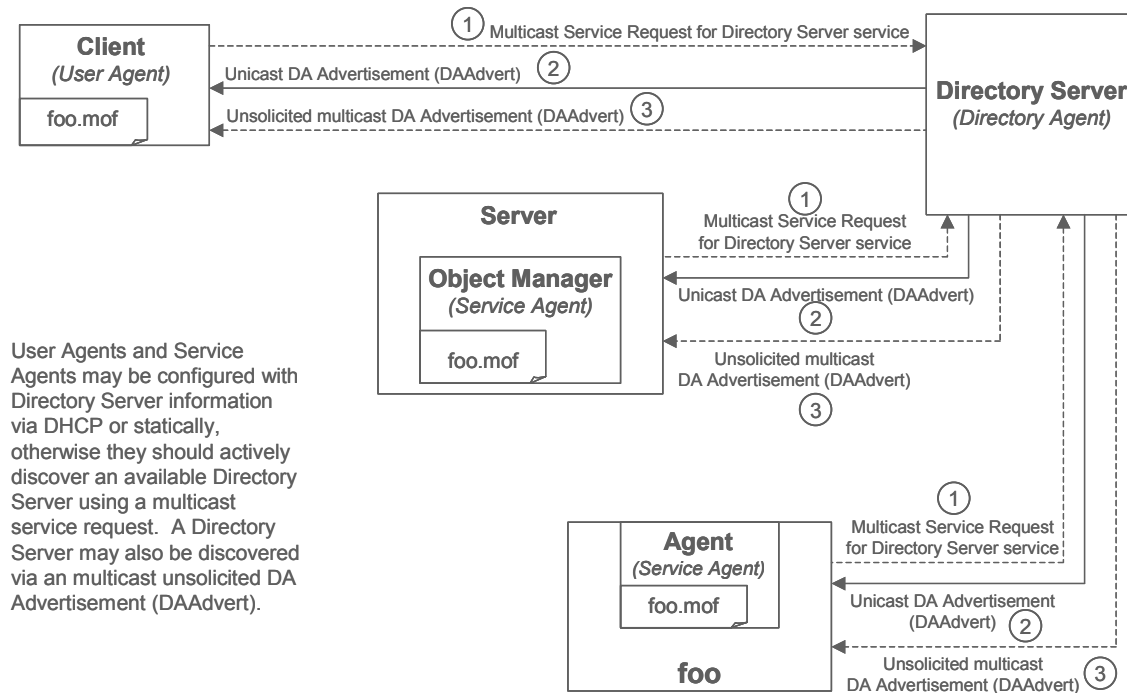


Figure 65: Directory Agent (DA) Discovery – Active and Passive

- 1: A User Agent (Client) or Service Agent (Object Manager or Agent) multicasts a Service Request for Directory Server service.
- 2: The Directory Agent responds with a unicast DA Advertisement of its service.
- 3: The Directory Agent **MUST** send an unsolicited multicast DAAdvertisement once per CONFIG_DA_BEAT.

5.6.3 Scope

A User Agent (UA) is normally assigned a scope string which may consist of multiple scopes and the UAs can only discover services within that specific scope. If the UA is configured with no scope, it can discover all available scopes and allow the Client to issue requests for any service available on the network.

5.6.4 Service Requests

After a User Agent (UA) has discovered at least one Directory Agent (DA), it **MUST** unicast a Service Request (SrvRqst) to that DA specifying the characteristics of the service that the Client requires. The UA will receive a Service Reply (SrvRply) specifying the location of all services that this DA has received advertisements from which satisfy the request (Figure 66).

If the UA cannot find a DA, it **MUST** multicast the SrvRqst. The UA could receive one or more unicast SrvRply messages from Service Agents (SAs), which advertise for the requested service (Figure 66).

UAs **MUST** be prepared for the possibility that the service information they obtain from DAs is stale. Some of the returned service information may represent services that are no longer running.

If a DA or SA fails to respond to a unicast UDP message in CONFIG_RETRY seconds, the message should be retried using exponential backoff. If a DA or SA fails to respond after CONFIG_RETRY_MAX seconds, the sender should consider the receiver to have gone down.

5.6.5 Minimal Implementation

The SLP specifies rules for implementing an absolutely minimal, or "nano" User Agent (UA). A nanoUA supports only basic service discovery in a particular scope. A nanoUA need only support the service type and scope fields. It must be able to do DA discovery and unicast to DAs if they are discovered.

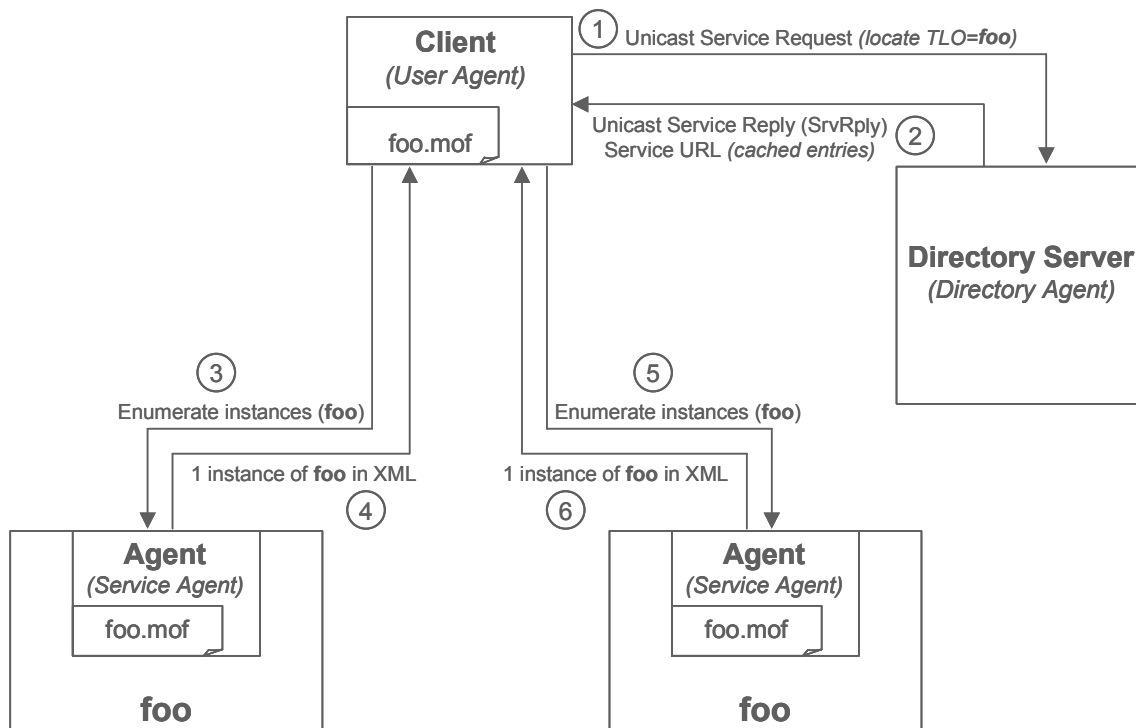


Figure 66: Service Agent Discovery using a Directory Agent

- 1: The User Agent (Client) sends a unicast Service Request to the Directory Agent specifying the characteristics of the service required (TLO=foo).
- 2: The Directory Agent replies with a unicast Service Reply Service URL containing the location of each Service Agent that has registered the requested service.
- 3, 5: The Client sends an 'Enumerate Instances (foo)' to each Service Agent identified in #2.
- 4, 6: Each Service Agent responds with one instance of 'foo' in XML.

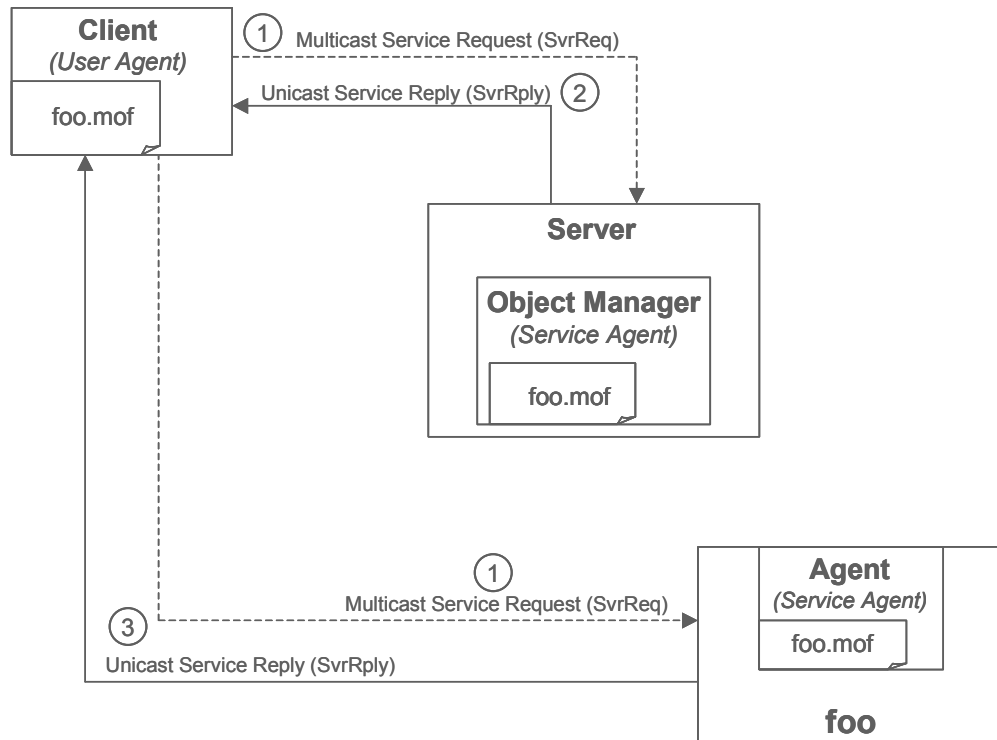


Figure 67: Service Agent Discovery without a Directory Agent

- 1: The User Agent (Client) sends a multicast Service Request specifying the characteristics of the service required.
- 2: The Object Manager that offers the service requested will respond with a unicast Service Reply Service URL containing its location information.
- 3: The Agent that offers the service requested will respond with a unicast Service Reply Service URL containing its location information.

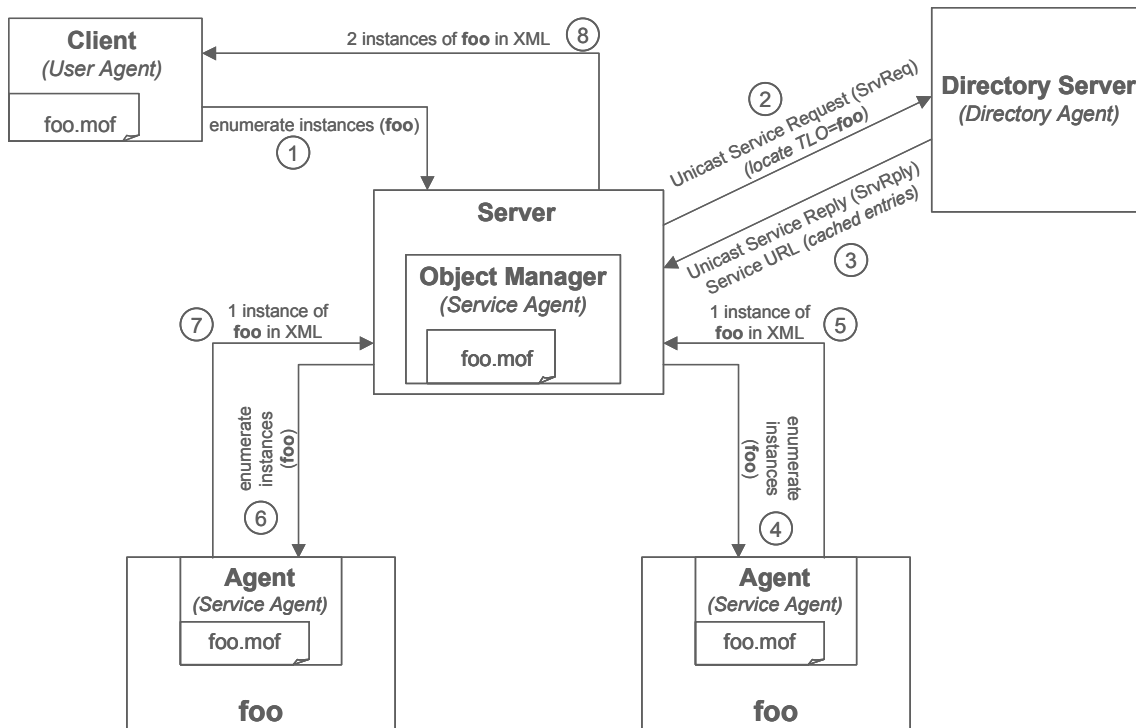


Figure 68: Service Agent Discovery Using a Directory Agent and Object Manager

- 1: The User Agent (Client) sends an 'Enumerate Instances (foo)' to a previously identified Service Agent (Object Manager).
- 2: The Service Agent (Object Manager) sends a unicast Service Request to the Directory Agent specifying the characteristics of the service required (TLO=foo).
- 3: The Directory Agent replies with a unicast Service Reply Service URL containing the location of each Service Agent (Agent) that has registered the requested service.
- 4, 6: The Service Agent (Object Manager) sends an 'Enumerate Instances (foo)' to each Service Agent (Agent) identified in #3.
- 5, 7: Each Service Agent (Agent) responds with one instance of 'foo' in XML.
- 8: The Service Agent (Object Manager) responds to the User Agent (Client) with two instances of 'foo' in XML.

5.7 Service Agents (SAs)

A Service Agent is an Agent, Lock Manager, or Object Manager process working on behalf of one or more services to advertise the services. See *Clause 2.3.2 Roles for Interface Constituents* and *Clause 7:Bluefin Roles* for further description of Agent, Lock Manager, and Object Manager.

Service Agents MUST accept multicast service requests and unicast service requests. SAs MAY accept other requests (Attribute and Service Type Requests). An SA MUST reply to appropriate SrvRqsts with SrvRply or SAAdvert messages. The SA MUST also register with all DAs as they are discovered (Figure 69).

5.7.1 Configuration

Service Agents (SAs) may be configured to use Directory Agents (DAs) via DHCP or statically. If a SA is not configured with either of these methods, it should actively discover DAs (see Figure 65).

5.7.2 Discovery of Directory Agents

Service Agents (SAs) that have been configured with Directory Agent (DA) information must unicast to communicate with those DAs. If a SA has not been configured with DA information, the SA should do active discovery of DAs using the Multicast Convergence Algorithm. SAs must passively detect DAs by listening for multicast DA Advertisements (DAAdverts). See Figure 65 for both active and passive DA discovery.

After a SA has discovered a DA, it must unicast to communicate with the DA. If the SA needs to register with a DA, a SA MUST wait a random time between 0 and CONFIG_REG_ACTIVE seconds before registering their services.

If a SA detects a DA it has never encountered, with a nonzero timestamp, the SA must register with the DA if they both are in the same scope (see Figure 69). SAs MUST examine the DAAdvert's timestamp to determine if the DA has had a stateless reboot since the SA last registered with it. If so, it registers with the DA. If the timestamp is set to 0, the DA is going down and no further messages should be sent to it. SAs MUST wait a random interval between 0 and CONFIG_REG_PASSIVE before beginning DA registration.

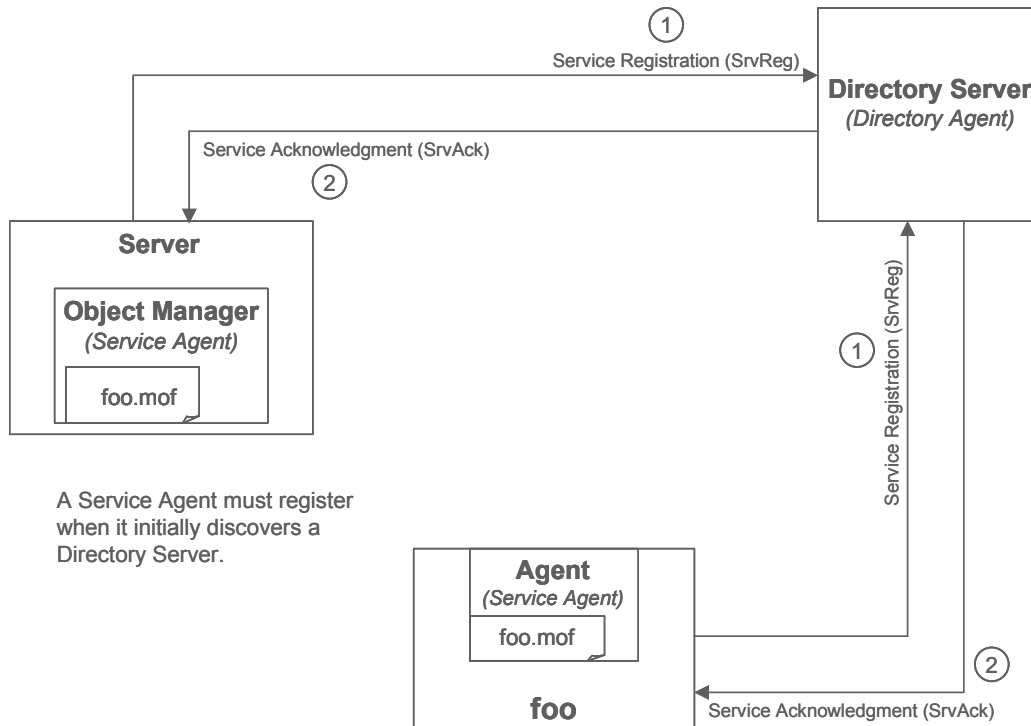


Figure 69: Service Agent Registration with a Directory Server

- 1: A Service Agent (Agent and/or Object Manager) registers its services with a Directory Agent by sending Service Registration (SrvReg) messages. A Service Agent must register in all the scopes it is configured to use.
- 2: The Directory Agent will respond with a Service Acknowledgment (SrvAck) messages to indicate success.

5.7.3 Scope

Services are grouped together using 'scopes'. These are strings that identify services, which are administratively identified. A scope could indicate a location, administrative grouping, proximity in a network topology or some other category. Service Agents are always assigned a scope string which may be a specific value or "DEFAULT".

5.7.4 Minimal Implementation

The SLP specifies rules for implementing an absolutely minimal, or "nano" Service Agent (SA). A nanoSA advertises a service type but no attributes allowing UAs to discover the service-by-service type and scope only. A nanoSA need not implement query handling since service requests for the service match if the query is empty. A nanoSA also need not implement TCP since no possibility for overflow exists.

5.8 Directory Agents (DAs)

Directory Agents **MUST** support the following SLP message types: Service Type Request (SrvRqst), Service Type Reply (SrvRply), Attribute Request (AttrRqst), Attribute Reply (AttrRply), Service Register (SrvReg), Service Deregister (SrvDereg), Service Acknowledge (SrvAck), DA Advertisement (DAAdvert), and SA Advertisement (SAAdvert). DAs **MUST** respond to AttributeRequest and Service Type Request messages.

DAs **MUST** accept unicast requests and multicast directory agent discovery service requests (for the service type "service:directory-agent"). DAs which receive a multicast SrvRqst for the directory agent discovery service **MUST** silently discard it if the <scope-list> is (a) not omitted and (b) does not include a scope they are configured to use. Otherwise the DA **MUST** respond with a DAAdvert.

When SLP SrvRqst, SrvTypeRqst, and AttrRqst messages are multicast, they contain a <PRList> of previous responders. Initially the <PRList> is empty. Any DA which sees its address in the <PRList> **MUST NOT** respond to the request. When these requests are unicast, the <PRList> is always empty.

DAs **MUST** flush service advertisements once their lifetime expires.

DAs **MUST** send unsolicited DAAdverts once per CONFIG_DA_BEAT. An unsolicited DAAdvert has an XID of 0.

5.8.1 Directory Agent (DA) Stateless Boot Timestamp

Directory Agent Advertisements (DAAdverts) **MUST** include DA Stateless Boot Timestamp. The Timestamp in the Authentication Block indicates the time at which all previous registrations were lost (i.e., the last stateless reboot). The Timestamp is set to 0 in a DAAdvert to notify User Agents (UAs) and Service Agents (SAs) that the DA is going down. DAs **MUST NOT** use equal or lesser Boot Timestamps to previous ones, if they go down and restart without service registration state. This would mislead SAs to not reregister with the DA.

5.8.2 Scope

By default, Directory Agents (DAs) are configured with the "DEFAULT" scope. Administrators may add other configured scopes, in order to support User Agents (UAs) and Service Agents (SAs) in non-default scopes. The default configuration **MUST NOT** be removed from the DA unless:

- There are other DAs which support the "DEFAULT" scope, or
- All UAs and SAs have been configured with non-default scopes.

5.8.3 Network Protocol Specifics

There can be only one Directory Agent per host. The Directory Agent will listen on port 427 (the port reserved for SLP). SLP Requests messages are multicast to the SLP Multicast address, which is 239.255.255.253. The default TTL to use for multicast is 255.

In isolated networks, broadcasts will work in place of multicast. To that end, SAs **SHOULD** and DAs **MUST** listen for broadcast Service Location messages at port 427. This allows UAs which do not support multicast the use of SLP on isolated networks.

If a SLP message does not fit into a UDP datagram it **MUST** be truncated to fit, and the OVERFLOW flag is set in the reply message. A UA which receives a truncated message **MAY** open a TCP connection with the DA or SA and retransmit the request, using the same XID. It **MAY** also attempt to make use of the truncated reply or reformulate a more restrictive request which will result in a smaller reply.

5.9 Service Agent Server (SA Server)

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) is required to listen for unicast request and specific multicast DA discovery service requests. And SAs and User Agents (UAs) that perform passive DA discovery must also 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 must listen for SLP messages. The SA Server will listen for incoming messages that request advertisement information and either answer each request or forward it to the appropriate SA. The SA Server will also perform 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.

5.9.1 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 objections 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.

5.9.2 SA Server (SAS) Clients

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 will listen 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.

5.9.2.1 SAS Client Requests – SA Server Responses

A SA Server will respond, 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 will discard 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.

5.9.3 SA Server Configuration

5.9.3.1 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 70 illustrates the various means of configuring a SA Server.

5.9.3.2 SLP Configuration File

If a SA Server will also function as a DA, the following DA configuration properties must be set:

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” will allow 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 will be included.

The remaining DA configuration property, net.slp.DAHeartBeat, with a default of 10,800 seconds, can be set as appropriate.

If a SA Server will not function as a DA, the following SA configuration property must be set:

Keyword	Data Type	Value
net.slp.SAAttributes	String	(SA-Server=true)

5.9.3.3 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 will contain the property name and `pcValue` will contain 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);
}
```

5.9.3.4 DHCP Configuration

If the Service Agent Server will also function 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.

5.9.3.5 Scope

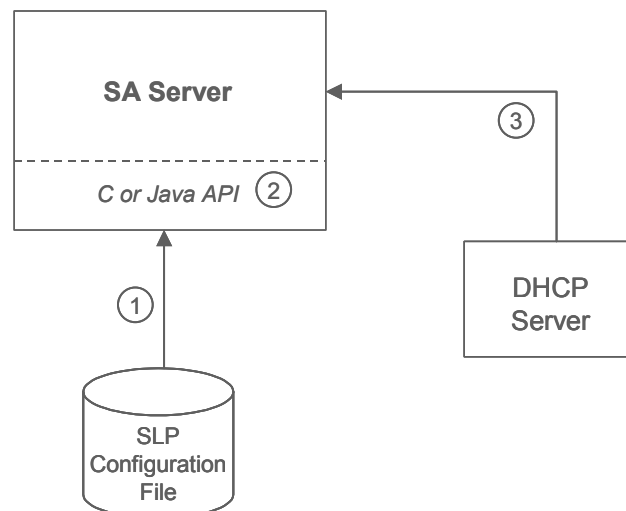
A Service Agent Server is configured with a minimum scope of `DEFAULT`. If a Service Agent Server is not functioning as a DA, `DEFAULT` will be the only scope configured. If a Service Agent Server is functioning as a DA, it may have additional scopes configured. Use of the `DEFAULT` scope will enable the associated SAS Clients (UAs, SAs and DA) to actively discover the Service Agent Server using a well-known value for scope.

Figure 70: 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.

5.9.4 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.



5.9.5 SAS Client Registration

Service Agents (SAs) that are SAS Clients must register and deregister with the local SA Server using the SrvReg/SrvDereg messages. The SA Server will respond with a Service Acknowledgement (SrvAck) message. The SA Server will 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 will also forward any SA registration to other DAs that have the same scope as the SA.

5.10 'Bluefin' Service Type Templates

5.10.1 'Bluefin' Abstract Service Template

Name of submitter: "Barry Nisly" <barry@prisa.com>
 Language of service template: en
 Security Considerations:
 See the security considerations of the concrete service types.
 Template Text:
 -----template begins here-----
 template-type=bluefin
 template-version=0.1
 template-description=
This is an abstract service type. The purpose of the bluefin service
type is to organize into a single category all Bluefin services.
 template-url-syntax=
 url-path= ; *Depends on the concrete service type.*

 service-hi-name=string

This is a human readable name of the service for purpose of displaying
in a human interface.

 service-hi-description=string O

This is a human readable description of the service for the purpose of
displaying in a human interface.

 service-id=string

This is a text rendering of unique identifier. It contains the same value
that appears in the "serviceid:" URI registered with the service.
 service-location-tcp=string M

The location of all services offered by the CIM Server over TCP transport.
Example: (service-location-tcp=http://example.com:8858,
https://example.com:8859,rmi://example.com:29340)

 role = string M L X
The Bluefin role.
 agent, object manager, lock manager
 profile = string M L X
The Bluefin profile associated with the service. (waiting on input from OMWG)
 block server, file server, host, ...
 -----template ends here-----

5.10.2 Bluefin' Concrete Service Template

Name of submitter: "Barry Nisly" <barry@prisa.com>

Language of service template: en

Security Considerations:

Template Text:

-----template begins here-----

template-type=bluefin:http

template-version=0.1

template-description=

This is a concrete service type based on the bluefin abstract service

type. This service type describes the Bluefin interface that uses

HTTP as the transport protocol.

template-url-syntax=

url-path = ; *Not used in this service template*

security = string O M L

none

The security protocol supported by the SLP agent.

none, ssl, password

-----template ends here-----

Clause 6: Lock Management

6.1 Introduction

The primary objective of lock management in Bluefin is to provide orderly access to shared resources by lock-aware Bluefin clients in a way that ensures expected behavior and achieves desired results during concurrent operations. Bluefin lock management will implement a degree of Isolation as described in terms of the ACID properties: Atomicity, Consistency, Isolation, and Durability. Bluefin lock management does **NOT** address nor provide the properties of Atomicity, Consistency, and Durability.

In the context of lock management, Bluefin defines an operation as a sequence of related agent actions initiated on behalf of a single Bluefin client. Even though Bluefin operations execute concurrently, it appears to each operation, *O*, that the other operations executed either before *O* or after *O*, but not both. This simply means that a Bluefin operation executing concurrently with other Bluefin operations under Bluefin Lock Management must behave exactly as if it were the only operation executing.

Most people are familiar with the locking semantics surrounding database operations, but the management of distributed resources differs substantially from that of updating records in a database, and these unique requirements need to be taken into account.

6.2 Terms

Lock Management Client: A XML-CIM Client that desires to lock one or more XML-CIM Servers in order to perform a protected operation.

Lock Management Server: A coordinating function for locking operations on multiple Lock Management Agents.

Lock Management Agent: A XML-CIM Server that can grant lock protection for some part of its Model.

6.3 Objectives

- Protect operations across multiple agents from multiple simultaneous non-cooperating clients.
- Protect invariants within an object model in a single agent.
- Provide for a simple, stateless locking paradigm with recovery for deadlock and error situations.
- Allow for a more sophisticated locking mechanism.
- Implementations of locking will be recommended, but not required.
- Provide for finer grain locking than just single agents.
- Need to address other management mechanisms and how they interact with out locking scheme.
- Do some minimal specification for the first release, but allow further extensions via SNIA and DMTF.
- Allow for standardizing this through the SNIA and DMTF.

6.3.1 Protected Operations

The operations that an agent needs to block when a lock is held on the resource are:

- GetInstance of the Instance that is locked

- DeleteInstance of the Instance that is locked
- ModifyInstance of the Instance that is locked
- Associator operations that involve locked Association Instance (implies a GetInstance)
- Enumerate Instances that includes the Instance that is locked
- Set Property on the Instance that is locked
- Extrinsic Method Call on the Instance that is locked
- Qualifier Operations that access the Instance that is locked
- Class Operations that would affect the Instance that is locked

6.3.2 Unprotected Operations

The operations that an agent can safely allow when a change lock is held on the resource are:

- GetProperty on the Instance that is locked
- Instance operations on non-locked Instances
- Class operations that don't affect locked Instance
- Qualifier operations on other Instances, Classes
- Query operations
- Associator operations that don't involve a locked Association Instance

6.3.3 Granularity of Locking

It is clear from the above that we would like to allow the granularity of locking to include the locking of individual properties and methods. This provides the maximum ability to have concurrent operations that do not interfere with each other to take place.

6.3.4 What is not covered

Locking does not address fully the needs of transactions, but does attempt to lay an architectural foundation for future work in this area. The locking model we are proposing is optimistic and requires participants to enforce well-known rules and cooperate to achieve the desired results. This locking model does not enforce locking semantics on unaware participants.

6.4 Lock Types

A single type of lock will satisfy management operations against resources. Management operations that only read properties do not interfere with each other and therefore no isolation is needed between multiple readers or a writer from a reader. Invariants can exist in a single instance or set of properties and methods, however, that need protection from multiple writers or a reader from a writer as seen in the above use cases. A change lock can handle the isolation for both of these cases. If a writer desires isolation from another writer, he first obtains the change lock that prevents any other writers from making changes to the instance or set of properties and methods during the update of the invariants. If a reader desires isolation from changes to invariants during the read of multiple, invariant properties, he first obtains the change lock that prevents any writers from modifying those properties. In the future, this change lock can be expanded to read and write locks if needed to provide better concurrency.

If there are dependencies between parts of a model (for example: `StorageExtents` in a `StoragePool`), the implementation MAY choose to consider the dependent parts to be locked when a request is granted for the antecedent parts, and reject non-locked client operations to those dependent parts. In most cases, locking a service may be sufficient to protect changes to the service's dependent parts. In the case where a particular object is represented in multiple Agents' namespaces, the lock must be obtained from the Agent that owns that object.

6.5 Lock Manager Reference Model

Lock Management involves three different roles as shown below in Figure 71: Lock Management Reference Model.

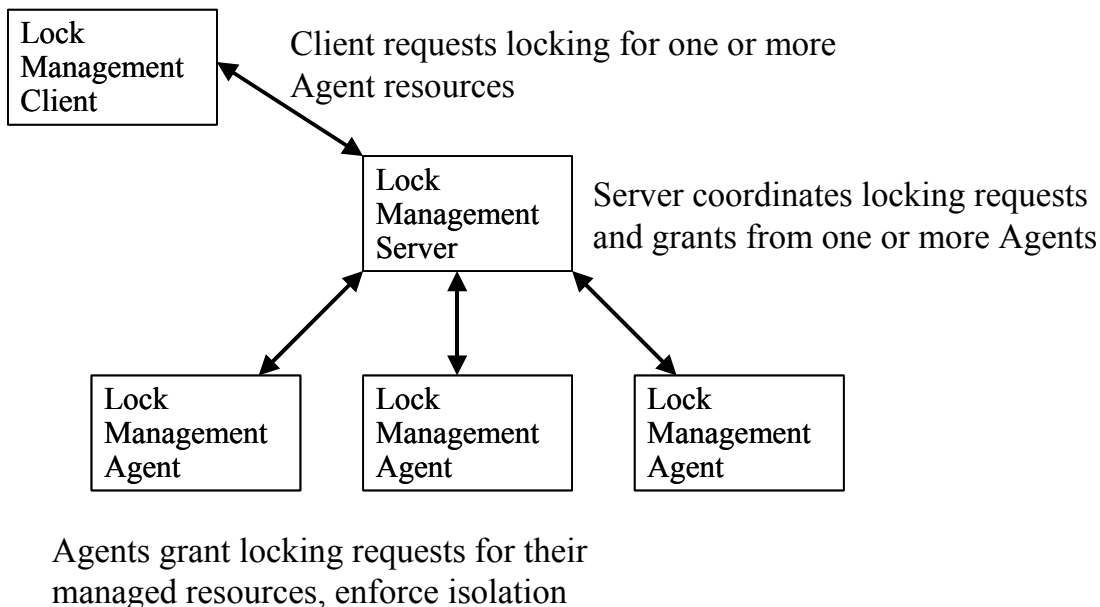


Figure 71: Lock Management Reference Model

The Lock Management (LM) Client is a reader or a writer of the resource that needs isolation from other Clients. The LM Client obtains the change lock, performs operations on one or more Agents, optionally extending its lock lease if it needs more time, and then releases the lock. The LM Agent grants requests for locks, extends lease times if necessary and protects operations from Clients that do not hold the lock. The LM Server allows locks to be held across multiple agents for atomic operations such as those needed for transactions. It enforces the rules for obtaining multiple locks and avoiding deadlock, coordinating the request, release, and renewal of locks. The minimal functions required for the LM Server can be extended by implementations to offer more efficiency and reliability. The intent is to provide a simple, easy to implement functionality that can be extended. Vendors are encouraged to add value in this area. A minimal lock management server is a stateless function, merely passing lock requests to Agents, receiving the results and passing them back to the Client; it keeps no memory of the locks that were granted and has no ability to queue lock requests. As a corollary to this, the minimum lock management server does not recognize a lock conflict, because it does not remember locks that it has previously granted.

6.5.1 Lock Management Server Operations

The operation messages that take place between the LM Client and the LM Server are described below:

Lock Request (AgentRequest [], LeaseDuration) – message from a LM Client requesting locks on multiple agents for desired lease duration. Each AgentRequest describes both the Agent, the Instances within that agent, and the properties/methods within each instance that need isolation.

Lock Grant (AgentResults [], LeaseTimeToLive) – message from the LM Server describing the actual instances that were locked and the remaining time on the lease. Each AgentResult describes the LM Agent that granted the request and the actual instances and properties/methods, which are locked (which may have a coarser granularity than requested). The LM Client will need to perform its operations before the LeaseTimeToLive elapses, or else renew the lease.

Lock Release (AgentResults []) – message from the LM Client that holds a lock informing the LM Server that its operations have completed and the resources are available for other LM Clients to lock.

Lock Refused (AgentResult) – message from the LM Server that indicates a failure to obtain a lock from one or more LM Agents. The AgentResult contains information from the first Agent that already had a requested instance or property/method that was locked.

A sequence diagram showing successful requesting and granting of locks is shown below in Figure 72:

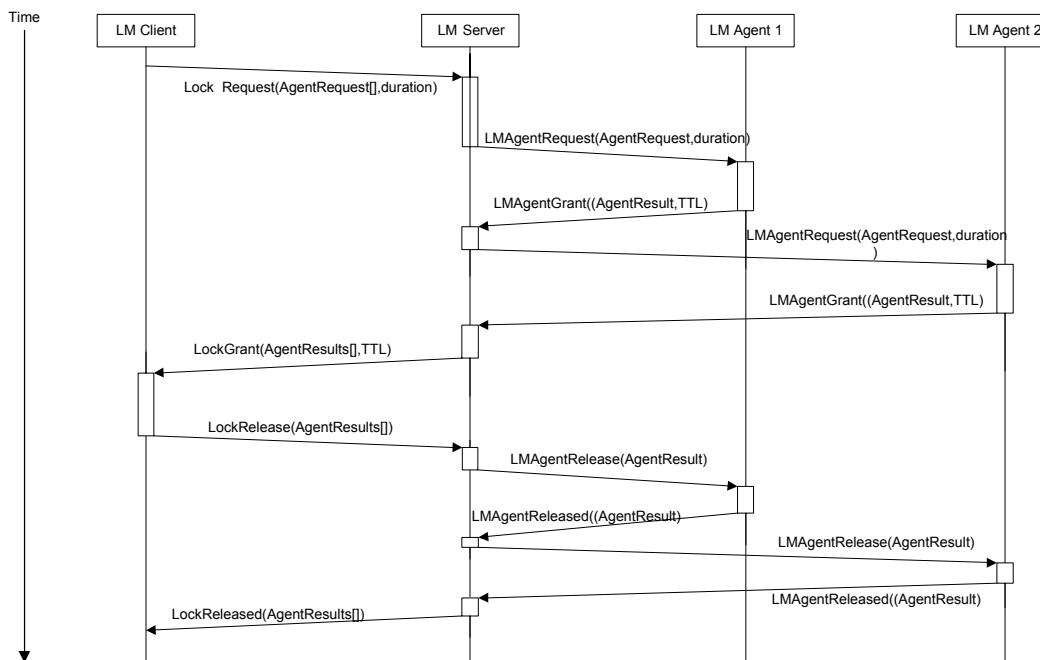


Figure 72: Lock Request Success Sequence Diagram

A sequence diagram showing unsuccessful requesting of locks is shown below in Figure 73:

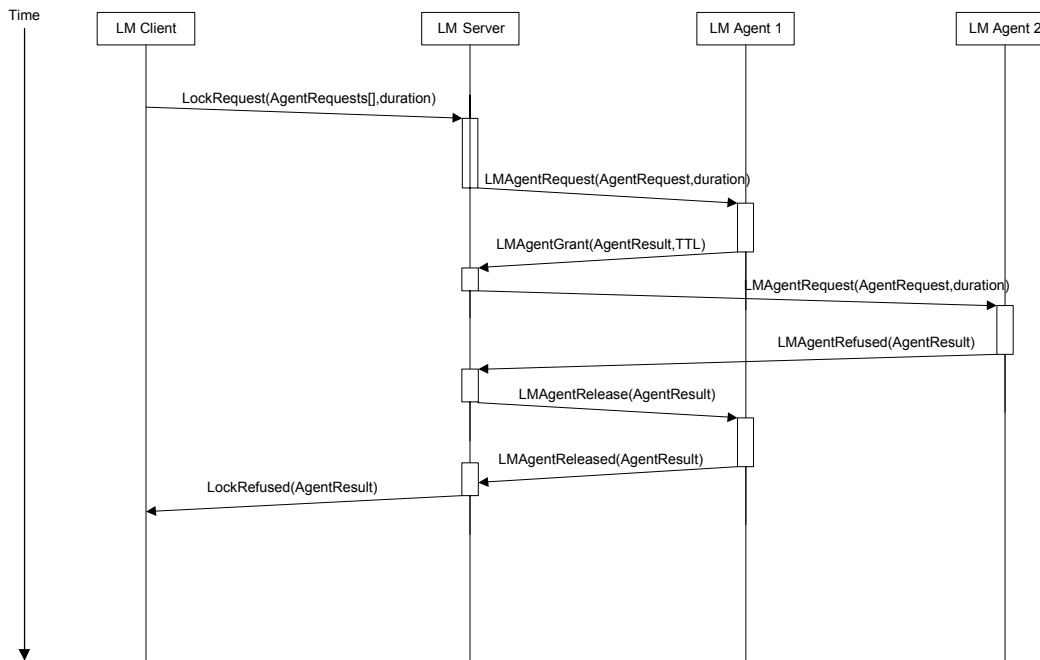


Figure 73: Unsuccessful Lock Request Sequence Diagram

6.5.2 Lock Management Agent Operations

The operation messages that take place between the LM Server and the LM Agent are similar to those between the LM Client and the LM Server, except that they contain pertaining to a single LM Agent instead of multiple:

LM Agent Request (AgentRequest, LeaseDuration) – message sent from LM Server to an LM Agent requesting locks on the instances specified by `CIMObjectPath[]` and associated properties and/or methods. The LM Agent is required to grant locks on all of the specified instances and properties and/or methods, or else refuse the entire request, returning the first `CIMObjectPath` and associated properties and/or methods that already had a lock.. The `LeaseDuration` time should be used for the lease time granted unless the Agent is configured with a smaller maximum lease time.

LM Agent Release (AgentResult) – message sent from the LM Server to an LM Agent indicating that the lock can be released and blocked operations can proceed.

LM Agent Renew (AgentResult, LeaseDuration) – message sent from LM Server to and LM Agent requesting that the lease time for the held lock (specified by `CIMObjectPath []` and associated properties and/or methods) be extended for another `LeaseDuration` milliseconds.

LM Agent Grant (AgentResult) – message sent from LM Agent to LM Server indicating that the lock request or lease renewal request was successful. The granted TTL may be less than the requested `LeaseDuration`. The Key is provided so that the LM Agent can allow certain operations from the Client that holds the lock, while blocking operations from other IP Clients that don't have the key

LM Agent Refuse (AgentResult) – message sent from LM Agent to LM Server indicating that the lock request or lease renewal request was unsuccessful. The `CIMObjectPath` and associated properties and/or methods indicate the first instance that already had a lock in the case of a lock request, or NULL if the lease renewal was unsuccessful.

6.6 Discovery

The Lock Management group that a Lock Management Server manages and one or more Lock Management Agents participate in will be identified with a SA attribute of LMGroup. That attribute will be a string value with a default value of “DefaultUnconfigured”. An administrator may create various Lock Managements groups by configuring the value of LMGroup as desired.

- Lock Management Client determines Lock Management group(s) required in the course of SA discovery of services, via LMGroup value for each LM Agent SA discovered.
- Lock Management Client then must discover the Lock Management Server with the appropriate value for the LMGroup of the LM Agents that are involved in the operation.

Configuring multiple lock management servers with the same LMGroup will cause unpredictable (and undesirable) results.

6.7 Deadlock Management

- Deadlock Avoidance: All locks must be acquired at once (should be acquired by the client before performing operations). If a lock request fails, all previously acquired locks are released after some retries.
- Deadlock Detection and Recovery: When a lock lease expires (detected), the lock is released (recovery)
- The number of lease renewals (configurable) should be limited with the default a small number (3).
- The maximum lease length is configurable with a reasonable default value (1,000 ms).

The deadlock rules above are implemented by a Lock Management Server on behalf of Clients, or by the Clients themselves in the absence of a Lock Management Server, and in the event of multiple LMGroups. If these rules are not followed, unpredictable results are possible.

6.8 Lock Leasing and Lease Extension

- The Lock Management Agent will control lease length by granting a Time To Live (TTL).
- The Lock Management Server will coordinate leasing.
 - Returns the smallest TTL from Lock Management Agents.
 - Lease renewals go through the Lock Management Server, then propagates to the Lock Management Agents.
- A Lock Management Agent may grant a lease renewal shorter than the duration requested.
 - Lock Management Agent may grant a shorter duration.
 - A different Lock Management Agent may reduce the total requested lease renewal as returned by the LM Server.
- When TTL expires, the Lock Management Agent will release the lock
- A subsequent release will not be considered an error (releasing a non-existent lock?)

The following diagram in Figure 74 shows the sequence of a successful lock lease renewal:

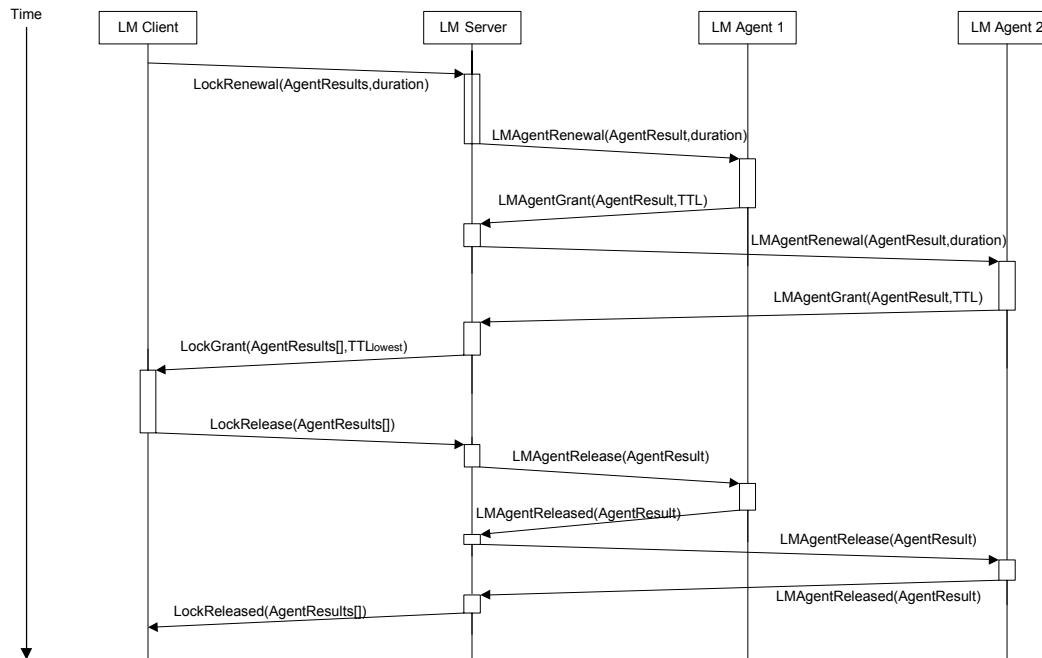


Figure 74: Lock Lease Renewal Success

The following diagram in Figure 75 illustrates the failure of a lock lease request:

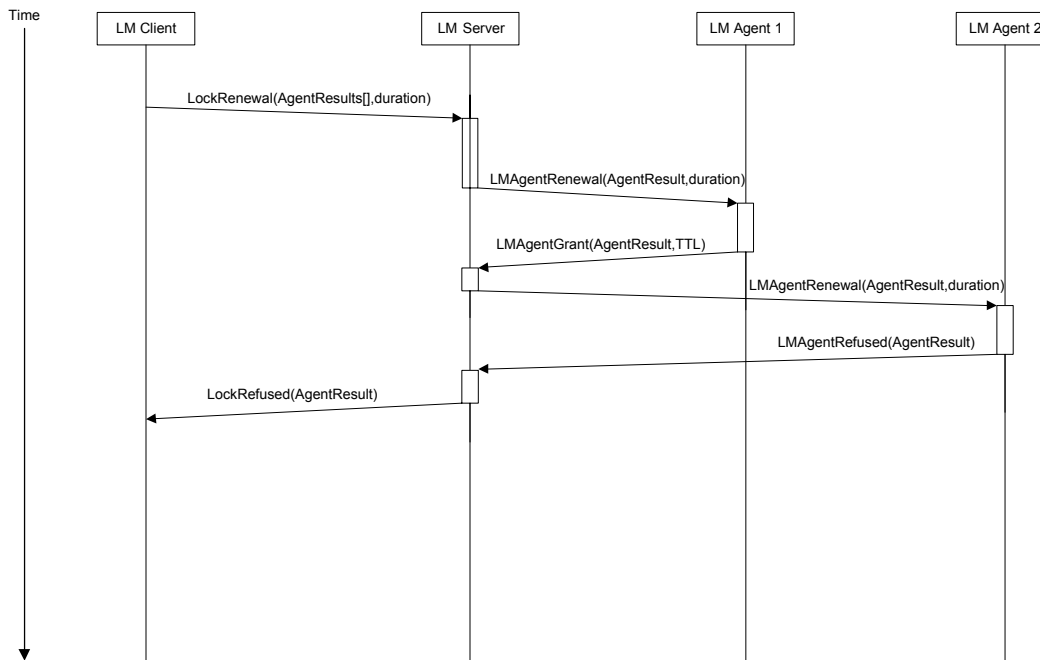


Figure 75: Lock Lease Renewal Failure

6.9 Lock Identification Token Considerations

Upon successfully granting a lock to a client, the Lock Management Agent will generate a unique **Identification Token** value that will be used by the granted client to perform operations. The value of this **Identification Token** is opaque to clients and lock managers, but must be faithfully produced in order to perform locked operations. Desirable properties of the **Identification Token** include:

- Reduce the potential for stale values being considered valid
- Each Agent controls its own value space (include example)

The **Identification Token** must be an integer in the range 0 to $2^{32}-1$. Since each agent generates its own **Identification Token** and does the corresponding validation of them, there is no problem that results from having two agents generate the same **Identification Token** value. The value of the **Identification Token** MUST NOT change on renewals,

The Lock **Identification Token** is passed as part of the context of the transport mechanism rather than as an explicit method parameter to intrinsic and extrinsic WBEM method calls. The CIM-XML transport definition is modified to accommodate this context using the HTTP protocol. Other transports will also need to define a means of transferring context values such as the lock **Identification Token**. Since all operations from this client for a given agent are locked by the same **Identification Token**, only one such token will be passed in this context.

6.10 Lock Management Implementations

Lock Management SHOULD be implemented by all appropriate Bluefin roles conforming to the first major version of this specification and MUST be implemented by all Bluefin roles conforming to the second major version of this specification. Care has been taken to allow roles that do not implement lock management to be able to interoperate with roles that do implement lock management. Obviously, existing CIM-XML Clients and CIM-XML Servers do not implement Lock Management as described herein and it is desirable to interoperate with these legacy systems during the (and in order to facilitate) adoption of Bluefin conforming implementations.

6.10.1 Lock Unaware Clients

A Lock Unaware Client is either a Bluefin V1.0 Client that does not implement Bluefin V1.0 lock management or a legacy (non Bluefin) Client.

Lock Unaware Clients will have no way of ensuring protection of invariants from other clients (of any type), but that is the current situation today without a lock management standard in place. Thus Lock Unaware Clients MUST take care to perform whatever steps are necessary to ensure that consistency is maintained, including multiple reads after doing a write (ensuring that the state was not partially updated). When a lock aware agent receives a request from a lock unaware client, it should treat the operation as being locked for the duration of the operation to protect lock aware clients of the agent. In other words, lock requests for an operation that is already being used by another client should not be granted until that operation completes. The set of relevant instances and properties/methods considered locked for this purpose is Agent model dependent. The lock unaware client's subsequent operation may be rejected if it is attempting to perform an operation on an object that is locked by a locking aware client.

6.10.2 Lock Unaware Agents/Object Managers

A Lock Unaware Agent or Object Manager is either a Bluefin V1.0 Agent or Object Manager that does not implement Bluefin V1.0 lock management or a legacy (non Bluefin) Agent or Object Manager.

Lock Unaware CIM-XML Servers (Object Managers and Agents) will not support the additional intrinsic methods for granting and releasing locks. These Object Managers and Agents MUST NOT include an LMGroup attribute in their service advertisement so that they will not be part of a Lock Manager's group. Clients that discover these Legacy roles will need to be coded to handle invariant operations as if they were Lock Unaware Clients (see Clause 6.10.1 Lock Unaware Clients).

6.11 Lock Management Client – Rules and Recommendations

A LM Client must follow the deadlock avoidance rules:

- If an LM Client attempts to obtain a lock while holding locks, and the attempt fails, the LM Client must not renew any leases on the locks that it holds and may retry the subsequent lock attempt up until the shortest lease on his current locks expire.
- Alternatively, he may release all locks at the first failed attempt.
- A Client discovers LM Agents and determines their LMGroup values. For agents within an LMGroup (all have same LMGroup value), if a Client finds an LMServer, he must use it for all locking operations. If an LM Server is not found for a group, an LM Client MAY issue requests to LM Agents directly as long as the above deadlock avoidance rules are followed.
- If an LM Client requests a lease time and receives a shorter time than it requested, it may try to extend the lease, it may release the lock, or it may attempt the operation in the time granted.

- A single Lock Management request may only include Agents that are in the same LMGroup.
- The LM Client should be counting down the TTL during its operations such that it knows when the lock has expired.

If a Bluefin client needs to do protected concurrent operations across multiple agents or object managers, it MUST be coded to support multiple LMGroups and perform locking across multiple Lock Managers. Just as the Lock Manager must do in a single LMGroup, the Client MUST obtain locks from all involved Lock Management Groups before proceeding with the cross agent operation. The same rules for obtaining locks on individual agents apply across multiple Lock Managers.

6.12 Lock Management Server – Rules and Recommendations

Only one LM Server can be active for a given Lock Management Group (set of Agents with the same LMGroup value). Conversely, an Agent belongs to only one Lock Management Group, determined by its LMGroup value. LMGroup values are administered to set up groups of LM Agents and their LM Server.

If an LM Server, during its discovery of LM Agents, finds another active LM Server with the same LMGroup value, it must not advertise itself. An administrator must only give one LM Server the same LMGroup value.

When an LM Server receives a request (renewals), it is responsible for sending the individual agent requests to each of the LM Agents involved in the LM Server request. It will have to examine the CIMObjectPath in the AgentRequest in order to determine the agent's host name or IP address.

- The LM Server needs to count down the TTLs it is receiving from the LM Agents as it is processing the request. This is done both in the request and renewal processing. All TTLs from the Agent Requests need to have valid time remaining when the LM Server sends the Lock Grant to the LM Client. The LM Server MAY retry lock requests to LM Agents that have timed out during the LM Server request or renewal processing.

6.12.1 Standard features

The minimal Lock Management Server would have the following characteristics and features:

- Accepts Lock Management Server requests and provides the proscribed responses
- Drives Lock Management Agent requests and processes their responses
- Is Stateless
- Can be subject to single points of failure
- Performs its functions only within its own LMGroup scope

6.12.2 Lock Manager Optional Proprietary Enhancements

- Distributed Lock Management Server
- Highly Available Lock Management Server
- Transactional Lock Management Server
- These enhancements are possible future enhancements to this specification, but may also be done by proprietary implementations as well.

6.13 Protocol Extensions – Methods

The following methods are proposed extensions to the Intrinsic Methods supported by Agents for the purposes of lock management.

AgentRequest – Contains one or more of:

- **CIMObjectPath** – a CIMObjectPath for each instance that is to be locked
- **ArrayOfNames** – an array of Property and/or Method Names for the instance specified by the CIMObjectPath (Usage is optional, based on the level of granularity needed)

AgentResults – Contains one or more of:

- **CIMObjectPath** – a CIMObjectPath for each instance that is locked
- **ArrayOfNames** – an array of Property and/or Method Names (optional)
- **IdentificationToken** – the Identification Token that is generated for the client to use in locked operations.
- **TTL** – the TTL that the Agent granted for this request

Lock Request (AgentRequest [], LeaseDuration) – message from a LM Client requesting locks on multiple agents for a desired lease duration. Each AgentRequest describes both the Agent and the Instances within that agent that need isolation.

Lock Grant (AgentResults [], LeaseTimeToLive) – message from the LM Server describing the actual instances that were locked and the remaining time on the lease. Each AgentResult describes the LM Agent that granted the request and the actual instances, which are locked (which may have a coarser granularity than requested). The LM Client will need to perform its operations before the LeaseTimeToLive elapses, or else renew the lease.

Lock Release (AgentResults []) – message from the LM Client that holds a lock informing the LM Server that its operations have completed and the resources are available for other LM Clients to lock.

Lock Refused (AgentResult) – message from the LM Server that indicates a failure to obtain a lock from one or more LM Agents. The AgentResult contains information from the first Agent that already had a requested instance locked.

LM Agent Request (AgentRequest, LeaseDuration) – message sent from LM Server to an LM Agent requesting locks on the instances specified by CIMObjectPath[]. The LM Agent is required to grant locks on all of the instances, or else refuse the entire request, returning the first CIMObjectPath that already had a lock. The LeaseDuration time should be used for the lease time granted unless the Agent is configured with a smaller maximum lease time.

LM Agent Release (AgentResult) – message sent from the LM Server to an LM Agent indicating that the lock can be released and blocked operations can proceed.

LM Agent Renew (AgentResult, LeaseDuration) – message sent from LM Server to and LM Agent requesting that the lease time for the held lock (specified by CIMObjectPath []) be extended for another LeaseDuration milliseconds.

LM Agent Grant (AgentResult) – message sent from LM Agent to LM Server indicating that the lock request or lease renewal request was successful. The granted TTL may be less than the requested LeaseDuration. The IdentificationToken is provided so that the LM Agent can allow certain operations from the Client that holds the lock, while blocking operations from other IP Clients that don't have the key

LM Agent Refuse (AgentResult) – message sent from LM Agent to LM Server indicating that the lock request or lease renewal request was unsuccessful. The CIMObjectPath indicates the first instance that already had a lock in the case of a lock request, or NULL if the lease renewal was unsuccessful.

Clause 7: Bluefin Roles

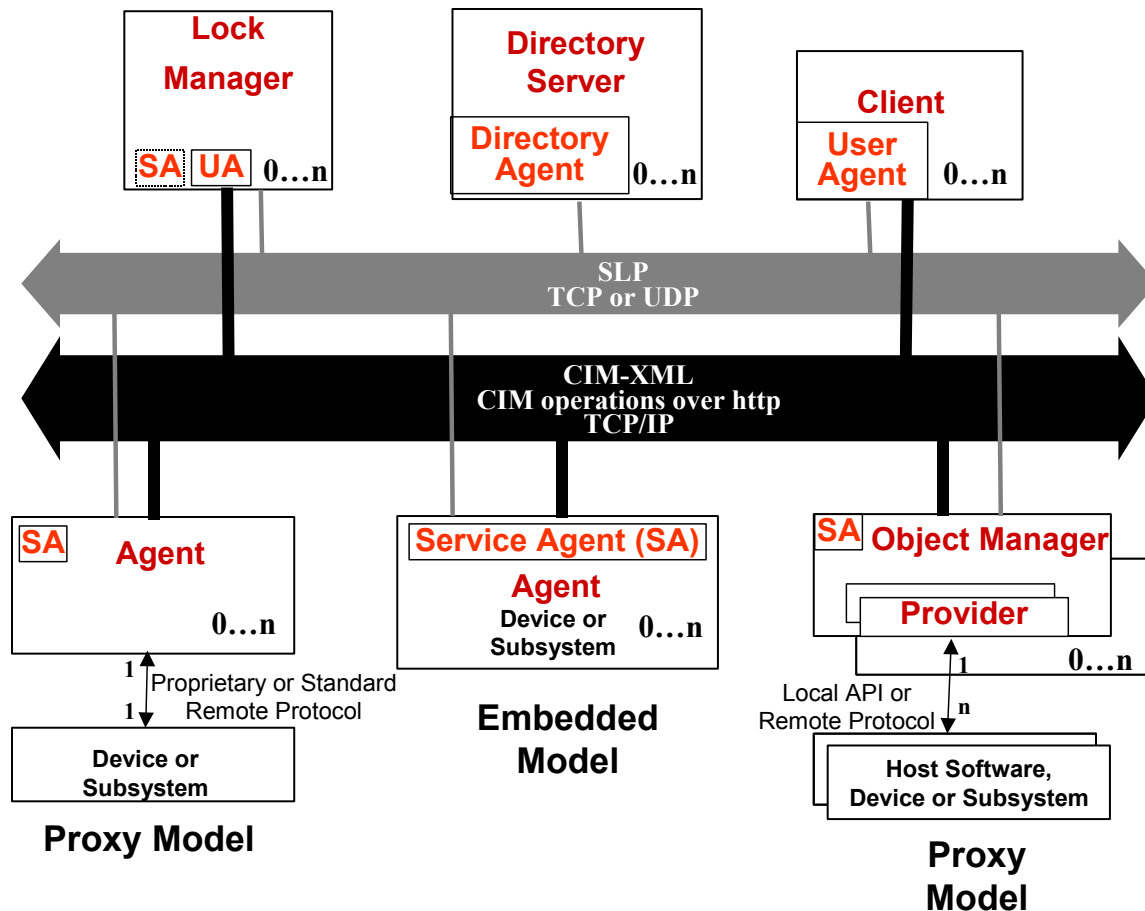


Figure 76: Complete Reference Model

7.1 Introduction

As shown in Figure 76 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:

- SLP Discovery Functions – the required discovery capabilities that the role must perform in the overall management system.
- Basic CIM-XML Operations – the management model operations that the role will perform.
- 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.

7.2 Client

The 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 will 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 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 also outside the scope of this specification, but is a candidate for follow-on work.

7.2.1 SLP Functions

The Client role is required to implement SLP User Agent (UA) functionality as specified in *Clause 5.6 User Agents (UA)*. The Client discovers all Agents within its configured scope that are required for its operations by querying for service specific attributes that match the criteria for those operations.

7.2.2 CIM-XML Protocol Functions

The Client role **MUST** implement CIM-Client functionality as specified by [CIM-XML] and **SHOULD** implement CIM-Listener functionality as specified by [CIM-XML].

7.2.3 Security Considerations

The Client role must implement security as specified in *Clause 3: Object Model*.

7.2.4 Lock Management Functions

The Client role **MUST** implement Lock Management Client functionality as specified in *Clause 6.11 Lock Management Client – Rules and Recommendations*. All Client operations that require isolation from other Clients **MUST** use Lock Management to protect those operations.

7.3 Agent

The intention of the Agent 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 Client and an Agent 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 agent can participate in a more complex management environment through the use of the standard mechanisms described here.

There are two basic implementation choices for the agent role as shown in Figure 76:

- Embedded Agent – the Agent functions are incorporated into the resource directly and do not involve separate installation steps to become operational.
- Proxy Agent – the Agent is hosted on a system separate from the resource and communicates with the resource via either a standard or proprietary remote protocol. This typically will involve an installation operation for the Agent 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 Agent role have purposely been scaled back from those of a typical Object Manager running on host with more significant resources. These required functions are described in the sections below.

7.3.1 SLP Functions

The Agent role is required to implement SLP Service Agent (SA) functionality as specified in Clause 3.7. Optionally, it SHOULD implement Service Agent Server functionality or use an existing SA Server if one exists. The Agent MUST advertise service specific attributes that allow the Client to locate it based on its profile, vendor and model as shown in the template below:

```

template-type=bluefin
template-version=0.1
template-description=
# This is an abstract service type. The purpose of the bluefin service
# type is to organize into a single category all Bluefin services.
template-url-syntax=
    url-path= ; Depends on the concrete service type.

service-hi-name=string

# This is a human readable name of the service for purpose of displaying
# in a human interface.

service-hi-description=string O

# This is a human readable description of the service for the purpose of
# displaying in a human interface.

service-id=string

# This is a text rendering of unique identifier. It contains the same value
# that appears in the "serviceid:" URI registered with the service.

service-location-tcp=string M

# The location of all services offered by the CIM Server over TCP transport.
# Example: (service-location-tcp=http://example.com:8858,
# https://example.com:8859,rmi://example.com:29340)

role = string M L X
# The Bluefin role.
agent, object manager, lock manager
profile = string M L X

```

The Bluefin profile associated with the service. (waiting on input from OMWG)
block server, file server, host, ...

7.3.2 CIM-XML Protocol Functions

The Agent role MUST implement CIM-Server functionality as specified by [CIM-XML].

7.3.2.1 Security Considerations

The Agent role must implement security as specified in Clause 3:.

7.3.2.2 Required Intrinsic Methods

An Agent is required to implement the following intrinsic methods as specified in [CIM-XML]:

- Get a CIM Class
- Get a CIM Instance
- Delete a CIM Instance
- Create a CIM Instance
- Modify a CIM Instance
- Enumerate subclasses of a CIM Class
- Enumerate subclass names of a CIM Class
- Enumerate instances of a CIM Class
- Enumerate instance names of a CIM Class
- Enumerate CIM Qualifier definitions
- Enumerate associators of a CIM Object
- Enumerate names of associators of a CIM Object
- Enumerate references to a CIM Object
- Enumerate names of references to a CIM Object
- Get a CIM Property value from a CIM Instance
- Set a CIM Property value from a CIM Instance
- Enumerate Qualifier declarations

Agents MAY implement other intrinsic methods as needed. Agents MUST also implement the intrinsic methods for locking specified here.

7.3.2.3 Required Model Support

The Agent MUST implement the Interop Schema as specified in [CIM-XML] and as detailed in the object model shown in Figure 77.

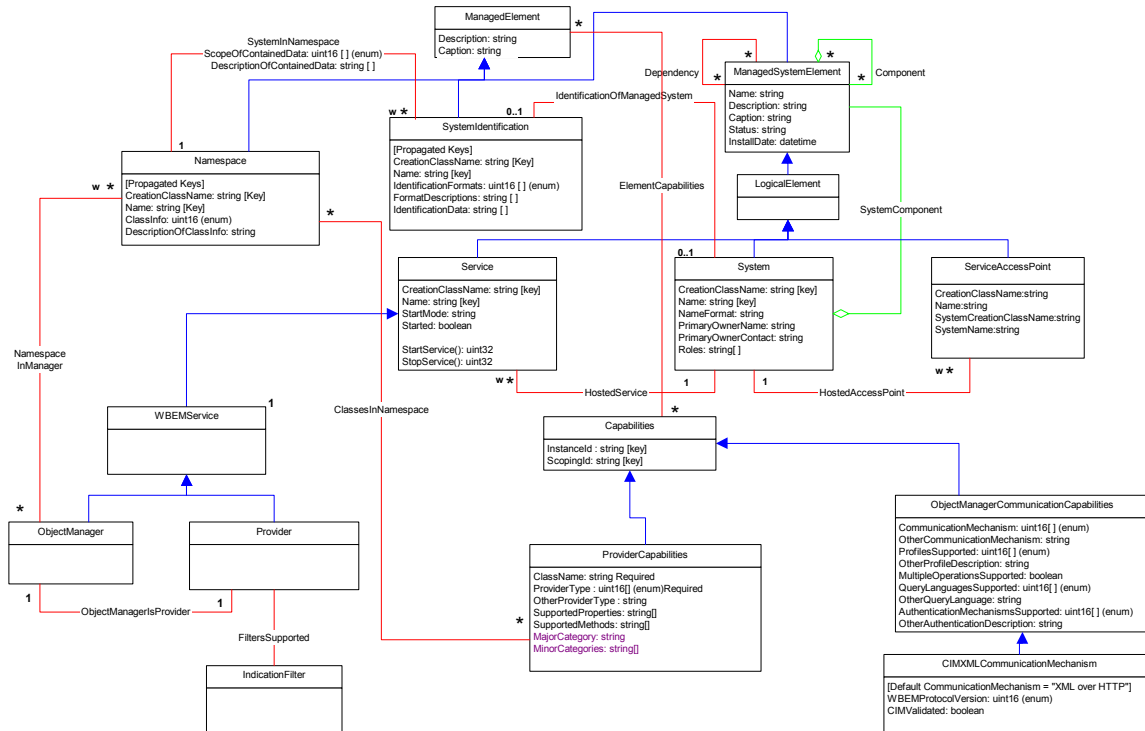


Figure 77: Interop Schema Object Model

7.3.3 Lock Management Functions

The Agent role **SHOULD** implement Lock Management Agent functionality as specified in *Clause 6.9 Lock Identification Token Considerations*.

7.4 Object Manager

The Object Manager 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 offloading these from the Client as well.

In addition, the Object Manager 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 Object Manager (see Clause 7.4).

An Object Manager 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 Object Managers (as many as one per host). Communication between Object Managers may be standardized in the future, but this capability is outside the scope of this specification. Object Managers may act as a point of aggregation for multiple Agents as described in 5.7.1 using existing standard mechanisms as specified here.

As Object Managers 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 than that of an Agent.

7.4.1 SLP Functions

The Object Manager role is required to implement SLP Service Agent (SA) functionality as specified in *Clause 5.7 Service Agents (SAs)*. The Object Manager **MUST** advertise service specific attributes that allow the Client to locate it based on its profile, vendor and model as shown in the template below:

```

template-type=bluefin
template-version=0.1
template-description=
# This is an abstract service type. The purpose of the bluefin service
# type is to organize into a single category all Bluefin services.
template-url-syntax=
    url-path= ; Depends on the concrete service type.

service-hi-name=string

# This is a human readable name of the service for purpose of displaying
# in a human interface.

service-hi-description=string O

# This is a human readable description of the service for the purpose of
# displaying in a human interface.

service-id=string

# This is a text rendering of unique identifier. It contains the same value
# that appears in the "serviceid:" URI registered with the service.

service-location-tcp=string M
# The location of all services offered by the CIM Server over TCP transport.
# Example: (service-location-tcp=http://example.com:8858,
# https://example.com:8859,rmi://example.com:29340)

role = string M L X
# The Bluefin role.
agent, object manager, lock manager
profile = string M L X
# The Bluefin profile associated with the service. (waiting on input from OMWG)
block server, file server, host, ...

```

7.4.2 CIM-XML Protocol Functions

The Object Manager role **MUST** implement CIM-Server functionality as specified by [CIM-XML].

7.4.2.1 Security Considerations

The Object Manager role must implement security as specified in Clause 4.

7.4.2.2 Required Intrinsic Methods

The Object Manager is required to implement the minimum profile as specified in [CIM-XML]. In addition, it **MUST** implement the intrinsic methods specified by 7.3.2.2.

7.4.2.3 Required Model Support

The Object Manager **MUST** implement the Interop Schema as specified in [CIM-XML] and as detailed in the object model shown in Figure 77.

7.4.3 Lock Management Functions

The Object Manager role **SHOULD** implement Lock Management Agent functionality as specified in *Clause 6.9 Lock Identification Token Considerations*.

7.4.4 Provider

A sub-role within an Object Manager 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 Object Manager as part of its operating system.

7.4.4.1 Required Model Support

The Provider **MUST** implement the Interop Schema as specified in [CIM-XML] and as detailed in the object model shown in Figure 77.

7.5 Lock Manager

The Lock Manager role is used to achieve locking across multiple instances of Agents and/or Object Managers such that a Client can perform concurrent operations with isolation from other Clients. The Lock Manager coordinates the acquisition of these locks, providing for the all or none semantics to avoid deadlock situations. A lock manager coordinates these operations across a group of Lock Management Agents (LM Agents) that can be Agents or Object Managers. The group is determined by the Lock Manager's participation in Discovery. Management Systems that do not have a Lock Manager **MAY** be restricted to performing unprotected operations unless the Client has implemented LMClient-LMAgent functionality.

7.5.1 SLP Functions

The Lock Manager role is required to implement SLP User Agent (UA) functionality as specified in *Clause 5.6 User Agents (UA)*. The Lock Manager discovers all Agents within its configured scope that are part of its LMGroup.

The Lock Manager role is required to implement SLP Service Agent (SA) functionality as specified in *Clause 5.7 Service Agents (SAs)*. The Object Manager **MUST** advertise service specific attributes that allow the Client to locate it based on its LMGroup:

```

template-type=bluefin
template-version=0.1
template-description=
# This is an abstract service type. The purpose of the bluefin service
# type is to organize into a single category all Bluefin services.
template-url-syntax=
  url-path= ; Depends on the concrete service type.

service-hi-name=string

# This is a human readable name of the service for purpose of displaying
# in a human interface.

```

```

service-hi-description=string O
# This is a human readable description of the service for the purpose of
# displaying in a human interface.

service-id=string
# This is a text rendering of unique identifier. It contains the same value
# that appears in the "serviceid:" URI registered with the service.

service-location-tcp=string M
# The location of all services offered by the CIM Server over TCP transport.
# Example: (service-location-tcp=http://example.com:8858,
# https://example.com:8859,rmi://example.com:29340)

role = string M L X
# The Bluefin role.
agent, object manager, lock manager
profile = string M L X
# The Bluefin profile associated with the service. (waiting on input from OMWG)
block server, file server, host, ...

```

7.5.2 Lock Management Functions

The Lock Manager role is required to implement the Lock Management Server functionality as specified in *Clause 6.12 Lock Management Server – Rules and Recommendations*.

7.6 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.

7.6.1 SLP Functions

The Directory Server role is required to implement SLP Directory Agent (DA) functionality as specified in *Clause 5.8 Directory Agents (DAs)*. The Directory registers all Agents, Object Managers and Lock Managers within its configured scope and allows queries for their respective service specific attributes.

7.7 Combined Roles on a Single System

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.

7.7.1 Object Manager as an Agent Aggregator

7.7.1.1 SLP Functions

The Object Manager role MAY implement SLP User Agent (UA) functionality as specified in *Clause 5.6 User Agents (UA)*. The Object Manager discovers all Agents within its configured scope that are aggregated by querying for service specific attributes that match the criteria for those aggregations.

7.7.1.2 CIM-XML Protocol Functions

The Object Manager role MAY implement CIM-Client functionality as specified by [CIM-XML] and MAY implement CIM-Listener functionality as specified by [CIM-XML]. An Object Manager MAY reflect instances and classes from the aggregated Agents (perhaps by delegating operations to the Agents), but is not required to do so. The Agent's Object Manager Model instances SHOULD be reflected in the advertised default namespace of the Object Manager. The hierarchy of Object Managers and Agents in a multi-level system needs to be reflected in the model such that it can be administrated.

7.7.1.3 Security Considerations

7.7.1.4 Lock Manager Functions

The Object Manager role MAY implement Lock Management Client functionality as specified in *Clause 6.11 Lock Management Client – Rules and Recommendations*. All Object Manager operations that require isolation from other Clients or Object Managers MUST use Lock Management to protect those operations.

Clause 8: Installation and Upgrade

8.1 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 deploy Clients, Agents and Object Managers. In order to supply a complete management solution, many management vendors will provide not only management client and object managers, but also other pieces of the management infrastructure (e.g., Lock Managers, Directory Servers, Object Managers, Databases, Messaging Servers, Application Servers and even Providers and Agents). Problems are possible when multiple vendors install/remove these infrastructure components in the same environment and conflicts 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, the management of constituents in a Bluefin environment should be easy to install, easy to upgrade, and easy to reconfigure. Mature products using Bluefin technology should experience seamless and nearly management free installation, upgrade, and reconfiguration.

This clause deals with the issues of Bluefin configuration management and recommends some steps that vendors should take to minimize the problem, leading to better customer satisfaction with the overall management solution.

8.2 Role of the Administrator

Ultimately, a vendor's installation software cannot make perfect decisions when conflicts arise, and 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, it is up to the administrator to resolve these conflicts and remove or disable the redundant infrastructure component(s).

Installation software can, 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 a conflict arise. The implications of this are that vendors will be 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.

8.3 Goals

8.3.1 Non-Disruptive Installation and De-installation

Clients, Agents, Proxy Agents, Lock Managers and Directory Servers MUST be capable of being installed and de-installed without disrupting the operation of other constituents in a Bluefin management environment. An Object Manager independent of its providers MUST be capable of being installed or de-installed from a Bluefin management environment without disrupting operations. 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 Bluefin interface constituents SHOULD NOT compromise the availability of mission critical applications.

8.3.2 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. The use of discovery and default configuration parameters throughout this specification is 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).

8.4 Installing Device Support

Manufacturers of storage hardware and software will typically install their product and the accompanying management support as an system. Bluefin software installed will typically fall into one of the following categories:

- Embedded Agent – the hardware device has an embedded Bluefin agent as an integrated component. No other installation of software is needed to enable management of the device.
- Proxy Agent – the hardware or software comes with an Agent that is installed on a host. The Proxy Agent will need to connect to the device and obtain a other unique identifying information.
- Provider - the hardware or software comes with a Provider that is installed into an Object Manager. The provider provides the management for one or more product instances and will need to either discover those instances or be explicitly configured to communicate with the device.

Conflicts are possible for Proxy Agents and Providers if multiple vendors attempt to install support for the same device. Also, when a device vendor needs to upgrade the Provider or Proxy Agent for the 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.

8.4.1 Installation

Installation software for devices needs to be able to find existing object managers 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 find existing agents/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 Bluefin Client during installation. This allows it to make the automated decisions that eliminate the need for an administrator to manually configure or adjust certain aspects of the management system.

The provider registration schema shows what device support is already installed and installation software SHOULD consult this schema before installing new software. If the installation software is upgrading device support from one scheme to another (for example from a proxy agent to a provider, or a provider to an embedded agent) the installation software needs to uninstall or disable the previous software support elements.

During installation, the installation software, acting as a Lock Aware Client may detect that some agents are Lock Unaware and needs to deal with (warn administrator) that both Lock Aware and Lock Unaware Agents/Object Managers could be the cause of inconsistent state in their network.

8.4.2 Discovery and Initialization of Device Support

Per the Bluefin Reference Model, vendors of Host Software, Devices, and Subsystems that are managed via a Proxy Agent or are managed through an Object Manager (with providers) are expected to provide a means for establishing a reliable connection between the Host Software, Device, or Subsystem and the ProxyAgent or Object Manager. As such, a special Client with administration/installation capability (as supplied by the vendor) is required to supply the IP address of a device/subsystem with related authentication credentials to a Proxy Agent or Object Manager designated to manage the device. This administrative Client may obtain the IP address of the device/subsystem via automated means (for example by probing through an in-band HBA, or looking at the object model that the HBA agent already provides) or via manual means (for example by requiring a system manager to manually input the IP address of the device/subsystem from documentation supplied by the vendor). Figure 78 below illustrates this requirement.

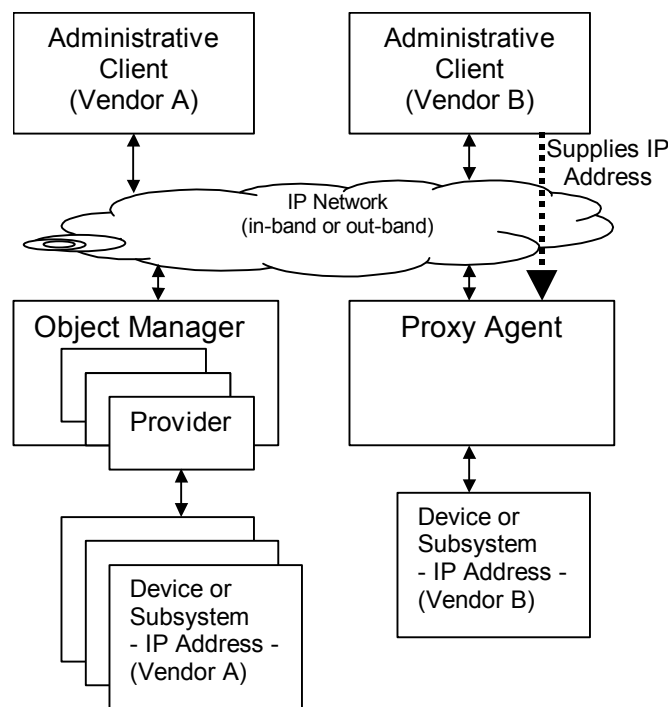


Figure 78: Configuration Administration

8.4.3 Removal/Update

During the removal of a device support software (agent, provider, object manager), the installation/removal software (if available) should automatically detect existing device support software in order to shutdown and remove these in a consistent manner. This detection process will need to be cognizant that Clients may be actively using this support. device and that thus, the device may need to be disabled for new management operations and administrated through an orderly shut-down procedure prior to de-installation . The implementation of shutdown procedures for components and any shutdown order dependency is outside the scope of this specification, but may need to be considered by implementers.

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. If the update includes installing a new provider, the installation software will need to use the provider installation/upgrade method that is supported by the existing Object Manager and will need to be coded for that (see *8.4.4 Reconfiguration*).

When a software update will involve 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.

8.4.4 Reconfiguration

When device support update requires an update of a provider, the device support installation software should configure the new provider with the same subscriptions that exist in the old provider before removing the old provider. This can be done via the instances of the subscriptions in the agent or object manager that currently exist.

8.4.5 Failure

Agents can become unavailable for several reasons. This includes powering off the device 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 shows up somewhere else, 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 device or its proxy becomes available again. In the case of a Proxy or Embedded agent, the agent (or its host) may go down, or the network to it could fail, but the device may still be available and that needs to be factored in to any 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.

8.5 Object Manager

8.5.1 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 existing object managers if one exists, by installing a provider for device support. If an object manager does not exist, or the device support does not work with the existing object manager (due to interface requirements, for example) it is recommended that the vendor supply a Proxy 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.

In band providers have a connection issue where zoning may alter the management path to the device from a provider or proxy agent. In this case, the device support may need to be installed on multiple hosts in the network and the vendor will need to provide some way to coordinate which provider or proxy agent is responsible for a particular device.

Vendors will typically install their providers in a unique namespace for isolation and qualification reasons. The installer must then discover (possibly via an SLP UA) the existing namespaces and insure that the one created for the new device is truly unique.

Installation of a management appliance still needs to be able to turn off built-in providers

Lock Aware Client need to deal with (warn administrator) both Lock Aware and Lock Unaware Agents/Object Managers could be the cause of inconsistent state in their network

8.5.2 Removal/Upgrade

In addition to the issues in 8.4.3, an Object Manager may be upgraded while keeping the same Providers as before. 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 it should be able to restore the previous configuration if possible.

8.5.3 Reconfiguration

See Clause 8.4.4 for issues that may also be applicable to Object Managers.

8.5.4 Failure

Temporary failure of an object manager (for example, host 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 additional components of the management system that the installation software needs to be aware of.

8.6 Client

8.6.1 Removal

When Client software is removed, the removal software should go in and remove any subscriptions for that client that exist in any agent or object manager. In addition, it should release any locks that are held in order to clean up the lock state as well.

8.6.2 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.

8.6.3 Failure

If possible, Clients should release locks before shutting down or upon unexpected failure.

8.7 Lock Manager

8.7.1 Installation

When installing a lock manager, the installation software **SHOULD** check for an existing lock manager for the configured LMGroup, if a Lock Manager is already present, don't install, If the installation software can't see a Lock Manager, the software may prompt the administrator to see if an existing (preferred) Lock Manager might be down (or even better – prompt only if a run time duplication is discovered).

8.7.2 Removal

During removal of a Lock Manager, the removal software **SHOULD** shutdown the Lock Manager to ensure there are no outstanding lock requests. A well designed Lock Manager **SHOULD** include a shutdown sequence that starts denying requests for new locks and advertises that it is going away upon the shutdown request and before the actual shutdown.

Without this clean up process, a lock client will not be able to use a lock manager to release locks and the locks will be unavailable until the agent times out their leases.

8.7.3 Reconfiguration

As a storage network grows, it may be desirable (for scalability reasons) to split the domain into two or more separate LMGroups. Vendors should advise their administrators on appropriate rules for this division (for example, not being able to do concurrent protected operations across groups).

It is recommended that vendors provide a means of determining when a Lock Manager is overloaded, causing unacceptable delays in processing lock requests. This information can be pivotal in ensuring timely processing of management operations within the storage network. Because of the complications involved in splitting a storage network into multiple LMGroup domains, a system administrator will desire to scale a single Lock Manager as far as possible.

8.8 Directory Server

8.8.1 Installation

The installation of more than one directory server 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 directory servers should be deployed in the management system. Customers may have already done this for network or system management reasons already.

8.8.2 Removal/Failure

SLP Clients already handle failure and removal of DAs as per the specification (See **Clause 5**).

8.9 Management Domains

The set of agents, object managers and a lock manager that are configured with the same LMGroup value may be considered a management domain for purposes of administration. The use of SLP scope is independent of the use of LMGroups for these purposes. Clients should not depend on any relationship between LMGroups and SLP Scopes.

8.9.1 Initial Configuration

Vendors should recommend that administrators of their products use the same LMGroup value for all agents and object managers in the same storage network (might include multiple fabrics). Vendors should also recommend to administrators of their products that all agents and object managers be on the same IP subnet or on connected subnets where the intervening router is configured to allow multicast packets between the subnets (this is to allow SLP discovery messages to flow to the entire management system).

8.9.2 Reconfiguration

Vendors of lock managers **SHOULD** consider producing software that will easily reconfigure (merge or split) a lock management domain to ease the burden of this task. Splitting or merging an LMGroup should involve bringing the old LockManager(s) down, reconfiguring the agents and object managers with their new LMGroup value (meanwhile the clients are going directly to the agents for lock requests) and then starting both of the new Lock Manager(s).

Appendix A: Glossary

Introduction

This Glossary provides a common set of definitions and terminology for usage within Bluefin documents, in particular the Bluefin Specification.

Where appropriate the source of a glossary entry is identified. Where a Bluefin definition conflicts with or is substantially different from an accepted definition from another source, both entries will be included.

All links are internal to this document. At this time internal hyperlinks are to the alphabetic section rather than to specific entries. No external links are included.

New Additions/Modifications:

This dictionary represents an attempt to arrive at a common body of terminology for the technologies it represents. The reader should recognize that in this rapidly evolving field, new terminology is constantly being introduced, and common usage is shifting. It is a living document, updated as necessary to reflect a consensus on common usage.

A

Access Control

SOURCE [SNIA]

The granting or withholding of a service or access to a resource to a requestor based on the identity of the principal for which the requestor is acting.

Address masking

CONTEXT [Storage System]

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](#).

Addressable Unit

CONTEXT [Storage System]

storage addressable unit (e.g. [LUN](#), [Virtual Disk](#), [Logical Disk](#), [Logical Volume](#), [Volume Set](#)).

Agent

SOURCE (Bluefin)

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.

Aggregation

SOURCE(CIM V2.2 Specification, Appendix E Glossary)

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.

ANSI:

CONTEXT [Standards] SOURCE[SNIA]

Acronym for American National Standards Institute.

API:

CONTEXT [Management] SOURCE[SNIA]

Acronym for Application Programming Interface. An interface used by an application program to request services. Abbreviated API. The term API is usually used to denote interfaces between applications and the software components that comprise the operating environment (e.g., operating system, file system, volume manager, device drivers, etc.)

Array

CONTEXT [Storage System]

A storage array, i.e., a [disk array](#) or [tape array](#).

Array Configuration

CONTEXT [Storage System]

1. Assignment of the disks and operating parameters for a disk array. Disk array configuration includes designating the array's member disks or extents and the order in which they are to be used, as well as setting parameters such as stripe depth, RAID model, cache allowance, spare disk assignments, etc. cf. [LUN Mapping](#)
2. The arrangement of disks and operating parameters that results from such an assignment.

Asymmetric Virtualization Appliance:

CONTEXT [Storage System] SOURCE[SNIA]

Synonym for an appliance that provides out-of-band virtualization. Out-of-band virtualization is the preferred term.

ATM:

CONTEXT [Network] SOURCE[SNIA]

Acronym for Asynchronous Transfer Mode.

B**Block Virtualization:**

CONTEXT [Storage System] SOURCE[SNIA]

The act of applying virtualization (q.v.), to one or more block based (storage) services for the purpose of providing a new aggregated, higher level, richer, simpler, secure etc. block service to clients. cf. file virtualization. Block virtualization functions can be nested. A disk drive, RAID system or volume manager all perform some form of block address to (different) block address mapping or aggregation.

C**Cardinality**

SOURCE (DMTF)

The number of values that may apply to an attribute for a given entity. Refer [UML Standards](#).

CIM:

CONTEXT [Management] SOURCE[SNIA]

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.

Abbreviated CIM. 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.

Client

SOURCE (LBP-WG)

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.

Completion Semantics

SOURCE (LBPWG)

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:

1. 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, polling are examples of mechanisms available to the caller in this regard.
2. 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.

Consumer

CONTEXT [Storage System]

A host, identified by [HBA WWN](#) or other identifier, that is allowed access to a storage [addressable unit](#)

Control Software

CONTEXT [Storage System]

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 disk](#)s 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.

Concurrency Control Protocol

SOURCE (LBP-WG)

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

1. Lock ordering: Transactions are ordered according to the order of arrival of their operations at the resource(s).
2. Optimistic ordering: Transactions proceed until they are ready to commit, whereupon a check is made to see whether they have performed conflicting operations.
3. Timestamp ordering: Transactions are ordered according to the time they were initiated.

Cooperating Clients

SOURCE (LBP-WG)

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

D

Data Invariant

SOURCE (LBP-WG)

A data invariant is the name given to the consistency-state of shared data. A data invariant must always be TRUE. When the data invariant is violated, the invariant must be protected via mutual exclusion. For example, suppose I have 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 I append 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

DES:

CONTEXT [Security] SOURCE[SNIA]

Acronym for Data Encryption Standard.

Device

CONTEXT [TBD]

a storage system that is addressable from the SAN.

DHCP:

CONTEXT [Network] SOURCE[SNIA]

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. Abbreviated DHCP. DHCP greatly simplifies the administration of large networks, and networks in which nodes frequently join and depart.

Directory

SOURCE (FC-GS-3)

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

Directory Agent (DA):

CONTEXT [SLP] SOURCE[Bluefin]

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.

Discovery

CONTEXT [Management]

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.

Disk Array

CONTEXT [Storage System]

A set of disks from one or more commonly accessible disk subsystems, combined with a body of control software. The control software presents the disks' storage capacity to hosts as one or more virtual disks. Control software is often called firmware or microcode when it runs in a disk controller. Control software that runs in a host computer is usually called a volume manager.

DLT:

CONTEXT [Tape] SOURCE[SNIA]

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

DRM:

CONTEXT [Management] SOURCE[SNIA]

Disk Resource Management. A work group in SNIA that is defining CIM models for storage devices, including switches, HBAs, Disk subsystems, Tape and storage appliances.

DMTF:**CONTEXT [Management] SOURCE[SNIA]**

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. Abbreviated DMTF. The DMTF has a web site at www.dmtf.org.

E**Enclosure****CONTEXT [Storage System]**

A box or cabinet.

Enumerate**CONTEXT [CIM] SOURCE[CIM]**

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.

Extent**CONTEXT [Storage Device] [Storage System] SOURCE[CIM]**

1. A set of consecutively addressed FBA disk blocks that is allocated to consecutive addresses of a single file.
2. A set of consecutively located tracks on a CKD disk that is allocated to a single file.
3. 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](#).

Extrinsic Method**CONTEXT [CIM]**

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

F**Fabric****CONTEXT [SAN] SOURCE (FC-GS-3)**

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

FC-GS-3

SOURCE (www.T11.org)

Fibre Channel - Generic Services 3 .. Abbreviation FC-GS-3 or GS-3

NCITS Project Number 1356-D T11.3 Group

FIPS:

CONTEXT [Security] SOURCE[SNIA]

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.

G**Grammar**

SOURCE (LBP-WG)

A formal definition of the syntactic structure of a language (see syntax), normally given in terms of production rules which 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 which 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.

GS-3

SOURCE (www.T11.org)

Refer [FC-GS-3](#)

H**Hard Zone**

SOURCE(Bluefin)

A [Zone](#) consisting of Zone Members, which are permitted to communicate with one another via the Fabric. Hard Zones are enforced by the [Fabric](#), which prohibits communication among members not in the same Zone. Note that [well-known addresses](#) are implicitly included in every Zone.

HBA

CONTEXT [TBD]

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

Host

CONTEXT [TBD]

A computer running an O/S.

HTTP

SOURCE (LBP-WG)

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

Hub

CONTEXT [Bluefin]

interconnect element that supports a ring topology.

I**Inheritance Relationship**

SOURCE (DMTF)

Refer [UML Standards](#).

Interconnect Element

CONTEXT [Bluefin]

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

Interface Definition Language (IDL)

SOURCE (LBP-WG)

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).

Intrinsic Method

CONTEXT [CIM]

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](#)

J**JBOD:**

CONTEXT [Storage System] SOURCE[SNIA]

Acronym for "Just a Bunch Of Disks." Originally used to mean a collection of disks without the coordinated control provided by control software; today the term JBOD most often refers to a cabinet of disks whether or not RAID functionality is present. cf. disk array.

K

L

LAN:

CONTEXT [Network]

Acronym for Local Area Network.

Language-Binding

SOURCE (LBP-WG)

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.

Lock Manager:

CONTEXT [Locking] SOURCE [Bluefin]

Short name for Lock Management Server.

Logical Disk

CONTEXT [Storage System]

A set of consecutively addressed FBA disk blocks that is part of a single virtual disk to physical disk mapping. Logical disks are used in some array implementations as constituents of logical volumes or partitions. Logical disks are normally not visible to the host environment, except during array configuration operations. *cf.* extent, virtual disk

Logical Unit (LU)

CONTEXT [SCSI]

The entity within a SCSI target that executes I/O commands. SCSI I/O commands are sent to a target and executed by a logical unit within that target. A SCSI physical disk typically has a single logical unit. Tape drives and array controllers may incorporate multiple logical units to which I/O commands can be addressed. Each logical unit exported by an array controller corresponds to a virtual disk. *cf.* LUN, target, target ID

Logical Unit Number (LUN)

CONTEXT [SCSI]

The SCSI identifier of a logical unit within a target.

Logical Volume

CONTEXT [Storage System]

A virtual disk made up of logical disks. Also called a virtual disk, or volume set.

LTO:

CONTEXT [Tape]

Acronym for Linear Tape Open.

LUN Mapping

CONTEXT [Storage System]

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 \(LU\)](#) is then be 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 independent of any knowledge of the intended use of the LUN. Only LUNs that are exposed via a [port](#) are available for access.

LUN Masking

CONTEXT [Storage System]

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](#)

M**MAN:**

CONTEXT [Network] SOURCE [SNIA]

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). Abbreviated MAN. 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.

Managed Object Format

CONTEXT [Management]

The syntax and formal description of the objects and associations in the CIM schemas. Abbreviated as MOF. MOF can also be translated to XML using a Document Type Definition published by the DMTF.

Mapping

CONTEXT [Storage System]

Conversion between two data addressing spaces. For example, mapping refers to the conversion between physical disk block addresses and the block addresses of the virtual disks presented to operating environments by [control software](#).

Marshalling

SOURCE (LBP-WG)

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

MD5:

CONTEXT [Security] SOURCE [SNIA]

A specific message-digest algorithm producing a 128-bit digest which is used as authentication data by an authentication service.

Method

SOURCE (LBP-WG)

The name of [one or more] operations[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.).

1. Methods are specified in an IDL.
2. Methods are declared in source header files of a programming language (.h files, Java Interface files, etc.,).
3. Methods are defined (or implemented) in source implementation files (.cpp. java class files, etc.,).

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

MOF

CONTEXT [Management]

Acronym for Managed Object Format.

Monitoring

CONTEXT [Bluefin]

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](#).

N

NAA:

CONTEXT [Standards] SOURCE [SNIA]

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).

NDMP:**CONTEXT [Backup] SOURCE [SNIA]**

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. Abbreviated NDMP.

An open standard protocol for network-based backup of NAS devices. Abbreviated NDMP. 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>.

N_Port**CONTEXT [SAN]**

Refer to [Port](#). Node

CONTEXT [SAN] SOURCE (FC-GS-3)

A collection of Ports. A Fiber channel device with a group of [ports](#).

SOURCE (SNIA)

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](#).

Non-cooperating clients**SOURCE (LBP-WG)**

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.

O**Operation****SOURCE (LBP-WG)**

An action executed within the body of a method (AKA procedure, function, or subroutine). Operations are distinct from methods (see [Method](#)).

Out-of-Band**CONTEXT [Fibre Channel] SOURCE [SNIA]**

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

P

Partition

CONTEXT [Storage System]

A subdivision of the capacity of a physical or [virtual disk](#). Partitions are consecutively numbered ranges of blocks that are recognized by MS-DOS, Windows, and most UNIX operating systems.

Synonym for the type of [extent](#) used to configure arrays.

A contiguously addressed range of logical blocks on a physical media that is identifiable by an operating system via the partition's type and subtype fields. A partition's type and subtype fields are recorded on the physical media and hence make the partition self-identifying.

PKI:

CONTEXT [Security] SOURCE [SNIA]

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

Platform

SOURCE [GS3]

Collection of Nodes.

Port

CONTEXT [SAN]

Connection point for links.

SOURCE [FC-GS-3]

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.

Port Mapping

CONTEXT [Storage System]

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](#)

Profile:

CONTEXT [Standards] SOURCE [SNIA]

A proper subset of a standard that supports interoperability across a set of products or in a specific application. Profiles exist for FCP (FCSI and PLDA), IP, and other areas. A profile is a vertical slice through a standard containing physical, logical and behavioral elements required for interoperability.

Protocol

SOURCE (LBP-WG)

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.

Provider

SOURCE (DMTF)

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.

Q

R

RAID:

CONTEXT [Storage System] SOURCE [SNIA]

An Acronym for Redundant Array of Independent Disks, a family of techniques for managing multiple disks to deliver desirable cost, data availability, and performance characteristics to host environments.

Relationship

SOURCE (DMTF)

Refer [UML Standards](#).

Required Reference

SOURCE (DMTF)

Refer [UML Standards](#).

S

SAN

CONTEXT [Fibre Channel] [Network] [Storage System]

1. Acronym for storage area network. (This is the normal usage in SNIA documents.)
2. Acronym for Server Area Network which connects one or more servers.
3. Acronym for System Area Network for an interconnected set of system elements.
4. A group of fabrics that have common leaf elements.

Scope:

CONTEXT [SLP] SOURCE [Bluefin]

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

SCSI:

CONTEXT [Standards] SOURCE [SNIA]

Acronym for Small Computer System Interface.

Semantics

SOURCE (LBP-WG)

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.

Server

SOURCE (LBP-WG)

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.

Service Agent (SA):

CONTEXT [TBD] SOURCE [Bluefin]

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.

Service Agent Server (SAServer):

CONTEXT [TBD] SOURCE [Bluefin]

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.

SES:

CONTEXT [SCSI] SOURCE [SNIA]

Acronym for SCSI Enclosure Services. An ANSI X3T10 standard for management of environmental factors such as temperature, power, voltage, etc. Abbreviated SES.

SLP:

CONTEXT [SLP, Discovery]

Acronym for Service Location Protocol.

Small Computer Storage Interface (SCSI)

CONTEXT [SCSI]

A collection of ANSI standards and proposed standards which define I/O buses primarily intended for connecting storage subsystems or devices to hosts through [host bus adapters](#). Originally intended primarily for use with small (desktop and desk-side workstation) computers, SCSI has been extended to serve most computing needs, and is arguably the most widely implemented I/O bus in use today.

SNIA:

CONTEXT [Standards] SOURCE [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.

SNMP:

CONTEXT [Networking, Management] SOURCE [SNIA]

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.

SNMP Trap:

CONTEXT [Management] SOURCE [SNIA]

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

Soft Zone

SOURCE (FC-GS-3)

A [Zone](#) consisting of Zone Members which 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.

Spare:

CONTEXT [Storage System] SOURCE [SNIA]

An object reserved for the purpose of substitution for a like object in case of that object's failure.

SPI:

CONTEXT [SCSI] SOURCE [SNIA]

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

SRM:

CONTEXT [Management] SOURCE [SNIA]

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.

SSL:

CONTEXT [Security] SOURCE [SNIA]

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)

SSP:

CONTEXT [Business]

Acronym for Storage Service Provider.

Switch:

CONTEXT [TBD]

Fibre channel interconnect element that supports a mesh topology.

Symmetric Virtualization Appliance:

CONTEXT [Storage System] SOURCE [SNIA]

Synonym for an appliance that provides in-band virtualization. In-band virtualization appliance is the preferred term.

Synchronous

SOURCE (LBP-WG)

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

Syntax

SOURCE (LBP-WG)

(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.

T

Tape:

Tape Drive:

CONTEXT [Storage System] SOURCE [SNIA]

A storage device that writes data sequentially in the order in which it is delivered, and reads data in the order in which it is stored on the media. Unlike disks, tapes use implicit data addressing. cf. disk

Tape Array

CONTEXT [Storage System]

A collection of tapes from one or more commonly accessible storage subsystems, combined with a body of control software.

Target

CONTEXT [SCSI]

The system component that receives a SCSI I/O command. cf., [LUN](#), [target ID](#)

Target Id

CONTEXT [SCSI]

The [SCSI](#) bus address of a target device or controller.

TCP/IP:

CONTEXT [Network] SOURCE [SNIA]

Shorthand for the suite of protocols that includes TCP, IP, UDP, and ICMP. This is the basic set of communication protocols used on the Internet.

TLS:

CONTEXT [Security]

Acronym for Transport Layer Security.

Transfer Syntax

SOURCE (LBP-WG)

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

T10:

CONTEXT [SCSI] SOURCE [SNIA]

The American National Standards Institute T10 technical committee, the standards organization responsible for SCSI standards for communication between computers and storage subsystems and devices.

T11:**CONTEXT [Fibre Channel] SOURCE [SNIA]**

The American National Standards Institute T11 technical committee, the standards organization responsible for Fibre Channel and certain other standards for moving electronic data into and out of computers and intelligent storage subsystems and devices.

U**UDP:****CONTEXT [Network] SOURCE [SNIA]**

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.

UML Standards**SOURCE (DMTF)**

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 **MUST** 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.

Universal Markup Language (UML)

CONTEXT [DMTF]

Refer to [UML Standards](#)

URL:

CONTEXT [Networking]

Uniform Resource Locator.

User Agent (UA):

CONTEXT [SLP] SOURCE [Bluefin]

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.

V

VAR:

CONTEXT [Business]

Value Added Remarketeer.

Virtualization System:

CONTEXT [Storage System] SOURCE [SNIA]

The act of integrating one or more (back end) services or functions with additional (front end) functionality for the purpose of providing useful abstractions. Typically virtualization hides some of the back end complexity, or adds or integrates new functionality with existing back end services. Examples of virtualization are the aggregation of multiple instances of a service into one virtualized service, or to add security to an otherwise insecure service. Virtualization can be nested or applied to multiple layers of a system.

Virtual Disk:

CONTEXT [Storage System]

A set of disk blocks presented to an operating environment as a range of consecutively numbered logical blocks with disk-like storage and I/O semantics. The virtual disk is the disk array object that most closely resembles a physical disk from the operating environment's viewpoint. *cf.* logical disk

Volume Set:

CONTEXT [Storage System]

Synonym for virtual disk.

W

WAN:

CONTEXT [Network] SOURCE [SNIA]

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

Weak Reference

SOURCE (DMTF)

Refer [UML Standards](#).

WBEM:

CONTEXT [Management] SOURCE [SNIA]

Acronym for Web Based Enterprise Management. Web-Based Enterprise Management is an initiative in the DMTF. Abbreviated WBEM. 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.

Well-known Address

SOURCE (FC-GS-3)

An address identifier defined in FC-PH to access a Service. A well-known address shall not be subject to [Zone](#) restrictions; i.e., a well-known address is always accessible, irrespective of the current Active Zone Set.

World Wide Name

CONTEXT [Fibre Channel] SOURCE (SNIA)

1. A 64-bit unsigned Name_Identifier which is worldwide unique. *cf.* Fibre Channel Name
2. A unique 48 or 64 bit number assigned by a recognized naming authority (often via block assignment to a manufacturer) that identifies a connection or a set of connections to the network. Abbreviated WWN. A WWN is assigned for the life of a connection (device). Most networking technologies (e.g., Ethernet, FDDI, etc.) use a world wide name convention.

WWN

CONTEXT [Fibre Channel] SOURCE [SNIA]

Acronym for [World Wide Name](#). A 64-bit unsigned Name_Identifier which is worldwide unique. *cf.* Fibre Channel Name A unique 48 or 64 bit number assigned by a recognized naming authority (often via block assignment to a manufacturer) that identifies a connection or a set of connections to the network. Abbreviated WWN. A WWN is assigned for the life of a connection (device). Most networking technologies (e.g., Ethernet, FDDI, etc.) use a world wide name convention.

W3C:

CONTEXT [Networking]

World Wide Web Consortium.

X

XML:

CONTEXT [Standards] SOURCE [SNIA]

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

XML-CIM Listener:

SOURCE [CIM Operations over HTTP Specification, Version 1.1c]

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

XML-CIM Server

SOURCE(DMTF)

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

Y

Z

Zone

CONTEXT [SAN]

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

SOURCE [FC-GS-3]

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.

Zone Definition

SOURCE [FC-GS-3]

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

Zone Member

SOURCE [FC-GS-3]

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.

Zone Member Definition

SOURCE [FC-GS-3]

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

- 1) a port on a Switch, (specifically by Domain_ID and port number); or,
- 2) the device's N_Port_Name; or,
- 3) the device's address identifier; or,
- 4) the device's Node_Name.

Zone Set**SOURCE (FC-GS-3)**

One or more Zones which 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.

Appendix B: Bibliography

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

- CIM Tutorial
(<http://www.dmtf.org/education/cimtutorial/index.php>)
- CIM UML Diagrams and MOFs
(http://dmf.org/standards/standard_cim.php)
- CIM System / Device Working Group Modeling Storage
(http://www.dmtf.org/var/release/Whitepapers/CIM_Device23_storage_wp.PDF)

Appendix C: Detailed Class Derivations

The following section provides a derivation for all classes found in the “Required Classes” sections of the object model defined in Clause 3. Each derivation includes the properties within each class which are important to the Bluefin object model, listing them under the super class within which they are defined. Concrete class names within the derivation are listed in **bold**.

C.1 **ActiveConnection**

The *ActiveConnection* association class is used to indicate that two *ProtocolEndpoints* are communicating or have the ability to communicate. Note that *ActiveConnection* was subclassing from *SAPSAPDependency* which is deprecated for CIM 2.6.

Property/Method	Type	Qualifier/Parameter	Description/Notes
Dependency			
ActiveConnection			
Antecedent	REF	Key	ProtocolEndpoint reference
Dependent	REF	Key	ProtocolEndpoint reference

Table 14: ActiveConnection Association Derivation

C.2 **AdminDomain**

(As defined by CIM)

This is a special grouping of *ManagedSystemElements*. The grouping is viewed as a single entity, reflecting that all of its components are administered similarly - either by the same user, group of users or policy. It serves as an aggregation point to associate one or more of the following elements: network devices, such as routers and switches, servers, and other resources that can be accessed by end systems. This grouping of devices plays an essential role in ensuring that the same administrative policy and actions are applied to all of the devices in the grouping. The specific behavior and/or semantics of the *AdminDomain* can be identified through its aggregated and associated entities.

The *System* class and its subclasses provide the scope for numerous types of managed objects. As such, these classes must have the ability to create unique keys. This attribute is used by the *System* class and its subclasses to define a unique Name, independent of the specific discovery protocol used. Use of the heuristic is optional, but recommended.

AdminDomain is a part of the Core model, which has frequently been used in the Networks Model to group together various network resources that must be administered the same way, perhaps using the same policies. Viewed in this light, its principal subclass is *AutonomousSystem*.

Property/Method	Type	Qualifier/Parameter	Description/Notes
ManagedElement			
ManagedSystemElement			
LogicalElement			
System			
Name	string		FC-GS Fabric Name
AdminDomain			

Property/Method	Type	Qualifier/Parameter	Description/Notes
NameFormat	string		

Table 15: AdminDomain Derivation

C.3 **AlertIndication**

(As defined by CIM)

A concrete superclass for CIM Alert notifications. An AlertIndication is a specialized type of Indication that contains information about the severity, cause, recommended actions and other data of a real world event. This event and its data may or may not be modeled in the CIM class hierarchy.

Property/Method	Type	Qualifier/Parameter	Description/Notes
Indication			
IndicationTime	datetime		Date and Time of Indication creation
ProcessIndication			
AlertIndication			
Description	string		A short description of the Indication
AlertingManagedElement	string		This property contains the full path 'object name' of the instance encoded as a string parameter
AlertType	uint16	Required	Primary classification of the Indication.
OtherAlertType	string		
PerceivedSeverity	uint16	Required	The severity of the AlertIndication from the notifier's point of view
OtherSeverity	string		
ProbableCause	uint16	Required	The probable cause of the situation which resulted in the AlertIndication

C.4 **AllocatedFromStoragePool**

AllocatedFromStoragePool is an association describing how LogicalElements are allocated from underlying StoragePools. These elements would be subclasses of StorageExtents or StoragePools.

Property/Method	Type	Qualifier/Parameter	Description/Notes
Dependency			
AllocatedFromStoragePool			
Antecedent	REF	Key	StoragePool reference
Dependent	REF	Key	LogicalElement reference
SpaceConsumed	Unit64		Space Consumed from this Pool. Units("Bytes").

Table 16: AllocatedFromStoragePool Derivation

C.5 **AssociatedStorageConfigurationJob**

AssociatedStorageConfigurationJob links the job to the object being modified. It should 'point' to the SourcePool until the new object is instantiated at which point it should point to the pool or volume being created. Note that one of these associations must exist for each job.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
Dependency			
UnitAccess			
Antecedent	REF	Key, override	StorageConfigurationJob reference
Dependent	REF	Key, override	LogicalElement reference

Table 17: AssociatedStorageConfigurationJob Association Derivation

C.6 **BasedOn**

(As defined by CIM)

BasedOn is an association describing how StorageExtents can be assembled from lower level Extents. For example, ProtectedSpaceExtents are parts of PhysicalExtents, while VolumeSets are assembled from one or more Physical or ProtectedSpaceExtents. As another example, CacheMemory can be defined independently and realized in a PhysicalElement or can be 'based on' Volatile or NonVolatileStorageExtents.

(As refined by Bluefin)

BasedOn is the association that defines storage extent virtualization – carving one large extent into multiple smaller extents, or combining extents to form a larger one.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Association] Dependency			
[Association] BasedOn			
Antecedent	REF	Override	StorageExtent Reference
Dependent	REF	Override	StorageExtent Reference
StartingAddress	unit64		where in lower level storage, the higher level Extent begins
EndingAddress	unit64		where in lower level storage, the higher level Extent ends.
OrderIndex	unit16		indicates the order to the BasedOn associations that describes how a higher level StorageExtent is assembled

Table 18: BasedOn Derivation

C.7 **ChangerDevice**

(As defined by CIM)

ChangerDevices represent hardware that moves PhysicalMedia within a System, such as a StorageLibrary.

Property/ Method	Type	Qualifier/ Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>MediaTransferDevice</i>			

Property/ Method	Type	Qualifier/ Parameter	Notes
<i>ChangerDevice</i>			
MaxTransitTime	uint32		
MediaFlipSupported	boolean		
AuditInProgress	boolean		
AuditsPerformed	boolean		

Table 19: ChangerDevice Derivation

C.8 Chassis

(As defined by CIM)

The Chassis class represents the PhysicalElements that enclose other Elements and provide definable functionality, such as a desktop, processing node, UPS, disk or tape storage, or a combination of these.

Property/ Method	Type	Qualifier/ Parameter	Notes
NumberOfPowerCords	uint16		
CurrentRequiredOrProduced	sint16		
HeatGeneration	uint16		
ChassisTypes	uint16[]		
TypeDescriptions	string[]		

Table 20: Chassis Derivation

C.9 Component

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>Component</i>			
GroupComponent	System	Key	System
PartComponent	LogicalDevice	Key	Port

Table 21: Component

C.10 ComponentCS

A ComputerSystem can aggregate another ComputerSystem. This association can be used to model MPP Systems with workstation frontends, an I2O subsystem embedded in a UnitaryComputerSystem, or a System that splits functionality between two processors, potentially running different Operating Systems.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Aggregation] Component			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Aggregation] SystemComponent			
[Aggregation] ComponentCS			
GroupComponent	REF	Override	ComputerSystem Reference
PartComponent	REF	Override	ComputerSystem Reference

Table 22: ComponentCS Aggregation Derivation

C.11 ComputerSystem

(As defined by CIM)

A class derived from System that is a special collection of ManagedSystemElements. This collection provides compute capabilities and serves as aggregation point to associate one or more of the following elements: FileSystem, OperatingSystem, Processor and Memory (Volatile and/or NonVolatile Storage).

(As refined by Bluefin)

This class represents a host, storage array, or fabric interconnect element (e.g. switch). The property OtherIdentifyingInfo is only required in the profile of the fabric and switch which contains the Domain ID.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
InstanceName	string		
ManagedSystemElement			
OperationalStatus	uint16		
LogicalElement			
System			
CreationClassName	string	MaxLen(256), Key	Name of Class
Name	string	MaxLen(256), Key	
ComputerSystem			
NameFormat	string	(Override "NameFormat")	Note – Need CR to add "T11Platform" and "NodeWWN"
OtherIdentifyingInfo	string[]		For the switch/fabric profile this property is required and should contain the DomainID with IdentifyingDescription containing "DomainID"
IdentifyingDescription	string[]		
Dedicated	int16[]	"Blockserver"	Defines System Type, REQUIRED
OtherDedicatedDescription	string		

Table 23: ComputerSystem Derivation

C.12 ComputerSystemPackage

(As defined by CIM)

Similar to the way that LogicalDevices are 'Realized' by PhysicalElements, ComputerSystems are realized in one or more PhysicalPackages. The ComputerSystemPackage association explicitly defines this relationship.

Property/Method	Type	Qualifier/Parameter	Notes/Description
<i>[Association] Dependency</i>			
[Association] ComputerSystemPackage			
Antecedent	REF	Override	PhysicalPackage Reference
Dependent	REF	Override	UnitaryComputerSystem

Table 24: ComputerSystemPackage Derivation

C.13 ConcreteIdentity

(As defined by CIM)

ConcreteIdentity associates two elements representing different aspects of the same underlying entity. It is defined as a Terminal, concrete subclass of LogicalIdentity, to be used in place of many specific subclasses of LogicalIdentity that add no semantics - "i.e., that do not clarify the type of identity, update cardinalities, or add/remove qualifiers. Note that a Terminal class cannot be subclassed. This is done to limit the use of ConcreteIdentity - as a concrete form of a general identity relationship. Specific semantics continue to be defined as subclasses of the abstract LogicalIdentity class.

(As refined by Bluefin)

Property/Method	Type	Qualifier/Parameter	Description/Notes
[Association] LogicalIdentity			
[Association] ConcreteIdentity			
SystemElement	REF	Override	LogicalDevice Reference
SameElement	REF	Override	LogicalDevice Reference

Table 25: ConcreteIdentity Derivation

C.14 Configuration

The Configuration object allows the grouping of sets of parameters (defined in Setting objects) and dependencies for one or more ManagedSystemElements. The Configuration object represents a certain behavior, or a desired functional state for the ManagedSystemElements. The desired functional state is typically driven by external requirements such as time or location. For example, to connect to a Mail System from 'home', a dependency on a modem exists, but a dependency on a network adapter exists at 'work'. Settings for the pertinent LogicalDevices (in this example, POTSModem and NetworkAdapter) can be defined and aggregated by the Configuration. Therefore, two 'Connect to Mail' Configurations may be defined grouping the relevant dependencies and Setting objects.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Configuration			
Name	Maxlen(256)	Key	The label by which the Configuration object is known.

Table 26: Configuration Class Derivation

C.15 ConfigurationCapacity

(As defined by CIM)

ConfigurationCapacity provides information on the minimum and maximum numbers of power supplies, fans, disk drives, etc. that can be connected to or placed on/into a **PhysicalElement** and the number that must be connected/added/removed at a time). The **PhysicalElement** whose configuration is described is identified using the **ElementCapacity** association, inherited from **PhysicalCapacity**. The object whose capacities are indicated (i.e., the power supply or fan) is identified in the **ObjectType** property of this class. Since the same min/max configurations can apply to multiple instances, this class is not defined as 'weak'.

Examples of the use of the **ConfigurationCapacity** class are to describe that a 'control unit' **Chassis** may be connected to (at most) 4 other I/O **Chassis**, or to describe what a **StorageLibrary**'s cabinet may contain. Continuing the latter example, a particular **StorageLibrary**'s cabinet might hold a minimum of 3 and a maximum of 9 **TapeDrives**, and a minimum of 88 and a maximum of 264 **StorageMediaLocations** ("Slots"). This information would be described in two instances of **ConfigurationCapacity**, both associated to the **StorageLibrary**'s **PhysicalPackage**. This class does NOT represent the tradeoffs that are likely to be required of one resource for another. It simply represents capacities. In the case of the **StorageLibrary**, there may be only 2 valid configurations - 9 **TapeDrives** with 88 Slots, or 3 **TapeDrives** with 264 Slots. This class only conveys that 'up to' 9 Drives and 'up to' 264 slots may be available and are supported.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>PhysicalCapacity</i>			
<i>ConfigurationCapacity</i>			
Name	string	Key	
ObjectType	Uint16	Key	"Other", "Processors", "Power Supplies", ... see MOF
OtherTypeDe scription	string	MaxLen (64)	
MinimumCap acity	Uint64		
MaximumCap acity	Uint64		
Increment	Uint32		

Table 27: ConfigurationCapacity Derivation

C.16 ControlledBy

(As defined by CIM)

The **ControlledBy** relationship indicates which **Devices** are commanded by or accessed through the **Controller LogicalDevice**.

(As refined by Bluefin)

Property/ Method	type	Qualifier/ Parameter	Description/Notes
[Association] Dependency			
[Association] DeviceConnection			
NegotiatedSpeed	unit64		
NegotiatedDataWidth	unit32		
[Association] ControlledBy			
Antecedent	REF	Override	Controller Reference
Dependent	REF	Override	LogicalDevice Reference
AccessState	unit16		
TimeOfDeviceReset	datetime		
NumberOfHardResets	unit32		
NumberOfSoftResets	unit32		

Table 28: ControlledBy Derivation

C.17 Controller

(As defined by CIM)

Controller is a superclass for grouping the miscellaneous control-related Devices that exist. Examples of Controllers are SCSIControllers, USBControllers, SerialControllers, etc. The Controller class is an abstraction for Devices with a single protocol stack, which exist primarily for communication to, and control or reset of downstream (ControlledBy) Devices.

Property/ Method	Type	Qualifier/Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
Controller			
TimeOfLastReset	datetime		
ProtocolSupported	uint16	Valuemap	See MOF
MaxNumberControlled	Uint32		
ProtocolDescription	string		

Table 29: Controller Derivation

C.18 Dependency

(As defined by CIM)

Dependency is a generic association used to establish dependency relationships between ManagedElements.

Property/Method	Type	Qualifier/Parameter	Description/Notes
Dependency			
Antecedent	REF	Key	ComputerSystem
Dependent	REF	Key	LogicalPortGroup

Table 30: Dependency Derivation

C.19 **DependencyContext**

(As defined by CIM)

This relationship associates a Dependency with one or more Configuration objects. For example, a ComputerSystem's dependencies could change based on the site/network to which the System is attached.

Property/Method	Type	Qualifier/Parameter	Description/Notes
[Aggregate] DependencyContext			
Context	REF	Aggregate, Key	Configuration Reference
Dependency	REF	Key	Dependency Reference

Table 31: DependencyContext Derivation

C.20 **DeviceSAPImplementation**

(As defined by CIM)

An association between a Service and how it is implemented. The cardinality of this association is many-to-many. A Service may be provided by more than one LogicalDevice, operating in conjunction. And, any Device may provide more than one Service. When multiple Devices are associated with a single Service, it is assumed that these elements operate in conjunction to provide the Service. If different implementations of a Service exist, each of these implementations would result in individual instantiations of the Service object. These individual instantiations would then have associations to the unique implementations.

(As refined by Bluefin)

A FCPort is associated to the ProtocolEndpoint by DeviceSAPImplementation and "joins" the System and Device model to the Network model.

Property/Method	Type	Qualifier/Parameter	Description/Notes
Dependency			
DeviceSAPImplementation			
Antecedent	REF	Key	FCPort reference
Dependent	REF	Key	ProtocolEndpoint reference

C.21 DeviceServicesLocation

(As defined by CIM)

Within an automated StorageLibrary, Media should be accessible to the various robotics and MediaTransferDevices (Pickers, Changers, InterLibraryPort s, etc.). The Library may be serviced by different TransferDevices, each responsible for a subset of the Library's StorageMediaLocations. The DeviceServicesLocation association indicates that the Transfer Device handles Media stored in the referenced Location. For example, LibraryPort 'A' may only service Media from Slots 1-10, while LibraryPort 'B' covers Slots 11-33. This detail is conveyed by this association.

Property/Method	Type	Qualifier or Parameter	Description/Notes
<i>[Association] Dependency</i>			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
<i>[Association] DeviceServicesLocation</i>			
Antecedent	REF	Key, Override	MediaTransferDevice Reference
Dependent	REF	Key, Override	StorageMediaLocation Reference
Inaccessible	boolean		

Table 32: DeviceServicesLocation Derivation

C.22 DeviceSoftware

(As defined by CIM)

(As refined by Bluefin)

The Purpose property of the *DeviceSoftware* association class indicates whether the software component identifies firmware, option ROM, or one or more device drivers. Logic which resides in boot ROM may load drivers from option ROM.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>[Association] Dependency</i>			
<i>[Association] DeviceSoftware</i>			
Antecedent	REF	Key, Override	SoftwareElement Reference
Dependent	REF	Key, Override	LogicalDevice Reference
Purpose	uint16 (enum)	"Unknown", "Other", "Driver", "Configuration Software", "Application Software", "Instrumentation", "Firmware", "BIOS", "Boot ROM"	
PurposeDescription	string		Freeform text to describe Purpose
LoadedOnDevice	boolean		True if "burned into" or located on device
UpgradeableOnDevice	boolean		True if software can be updated

Table 33: DeviceSoftware Derivation

C.23 DiskDrive

(As defined by CIM)

Capabilities and management of a DiskDrive, a subtype of MediaAccessDevice.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
SystemCreationClassName	string	MaxLen(256)	The scoping System's CreationClassName.
SystemName	string	MaxLen(256)	The scoping System's Name.
CreationClassName	string	MaxLen(256)	The name of the concrete subclass
DeviceID	string	MaxLen(64)	
OtherIdentifyingInfo	String[]		
IdentifyingDescriptions	String[]		
<i>MediaAccessDevice</i>			
<i>DiskDrive</i>			

Table 34: DiskDrive Derivation

C.24 ElementCapabilities

(As defined by CIM)

ElementCapabilities represents the association between ManagedElements and their Capabilities. Note that the cardinality of the ManagedElement reference is Min(1), Max(1). This cardinality mandates the instantiation of the ElementCapabilities association for the referenced instance of Capabilities. ElementCapabilities describes the existence requirements and context for the referenced instance of ManagedElement. Specifically, the ManagedElement MUST exist and provides the context for the Capabilities.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ElementCapabilities</i>			
<i>HostedCapabilities</i>			
ManagedElement	ManagedElement		AdminDomain
Capabilities	Capabilities		ZoneCapabilities

Table 35: ElementCapabilities Derivation

C.25 ElementCapacity

(As defined by CIM)

ElementCapacity associates a PhysicalCapacity object with one or more PhysicalElements. It serves to associate a description of min/max hardware requirements or capabilities (stored as a kind of PhysicalCapacity), with the PhysicalElements being described.

Property/ Method	Type	Qualifier or Parameter	Description/Notes
<i>[Association] Dependency</i>			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
<i>[Association] ElementCapacity</i>			
Antecedent	REF	Key, Override	PhysicalCapacity Reference
Dependent	REF	Key, Override	PhysicalElement Reference

Table 36: ElementCapacity Derivation

C.26 ElementConfiguration

This association relates a Configuration object to one or more ManagedSystemElements. The Configuration object represents a certain behavior, or a desired functional state for the associated ManagedSystemElements.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>Dependency</i>			
<i>ElementConfiguration</i>			
Element	REF	Key	ManagedSystemElement reference
Configuration	REF	Key	Configuration reference

Table 37: ElementConfiguration Association Derivation

C.27 ElementSetting

ElementSetting represents the association between ManagedSystemElements and the Setting classes defined for them.

Property/ Method	Type	Qualifier or Parameter	Notes
<i>[Association] Dependency</i>			
<i>[Association] ElementSetting</i>			
Element	REF	Key	ManagedElement Reference
Setting	REF		Setting Reference

Table 38: ElementSetting Association Derivation

C.28 ElementStatistics

(As defined by CIM)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>Statistics</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ElementStatistics</i>			
Stats	REF	Key, Weak	DeviceStatisticalInformation
Element	REF	Key, Min(1), Max(1)	FCPort

Table 39: DeviceStatistics Derivation

C.29 **ExecutingStorageConfigurationJob**

ExecutingStorageConfigurationJob is an association between the StorageConfigurationService and an executing StorageConfigurationJob. The cardinality of this association is 1-to-many and is weak with respect to the service. Each service may have many executing Configuration Jobs.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>Dependency</i>			
<i>ExecutingStorageConfigurationJob</i>			
Antecedent	REF	Key, override	Service reference
Dependent	REF	Key, override	StorageConfigurationJob reference

Table 40: ExecutingStorageConfigurationJob Derivation

C.30 **ExtentRedundancyComponent**

(As defined by CIM)

Describes the StorageExtents participating in a StorageRedundancyGroup.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>[Aggregation] Component</i>			
<i>[Aggregation] RedundancyComponent</i>			
<i>[Aggregation] ExtentRedundancyComponent</i>			
GroupComponent	REF	Override	StorageRedundancyGroup Reference
PartComponent	REF	Override	StorageExtent Reference

Table 41: ExtentRedundancyComponent Derivation

C.31 **ExtraCapacityGroup**

(As defined by CIM)

A class derived from RedundancyGroup indicating that the aggregated elements have more capacity or capability than is needed. An example of this type of redundancy is the installation of N+1 power supplies or fans in a system.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			

Property/Method	Type	Qualifier/Parameter	Description/Notes
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>RedundancyGroup</i>			
CreationClassName	string	MaxLen(256), Key	The name of the concrete subclass
Name	string	MaxLen(256), override, Key	
RedundancyStatus	UInt16		
ExtraCapacityGroup			
MinNumberNeeded	UInt32		the smallest number of elements that must be operational in order to have redundancy.
LoadBalancedGroup	Bool		indicating whether load balancing is supported

Table 42: ExtraCapacityGroup Derivation

C.32 FCPort

(As defined by CIM)

Capabilities and management of a Fibre Channel Port Device.

(As refined by Bluefin)

The FCPort class represents information about a single FC port. The *SystemDevice* association is used to associate a *FibreChannelAdapter* with its owning *System*.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
DeviceID	String	Key, MaxLen (64)	
<i>NetworkAdapter</i>			
PermanentAddress	String	MaxLen (64)	Port WWN (InfiniBand: Port GUID)
NetworkAddresses[]	String	MaxLen (64), ArrayType ("Indexed")	FCID (InfiniBand: LIDs) FC-FS Address Identifier
Speed	uint64	Units ("Bits per Second")	Current bandwidth estimate FCFS Port Operating Speed
MaxSpeed	uint64		FC-FS Port Speed Capabilities
SupportedMaximumTransmissionUnit	uint64		
ActiveMaximumTransmissionUnit	uint64		
Port			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
InstanceName	String		
OtherPortType	uint16		
PortNumber	uint16		
OtherLinkTechnology	uint16		
<i>FCPort</i>			
PortType	uint16		FC-GS Port.Type
LinkTechnology	uint16		FC
SupportedFC4Types	uint16		FC-GS FC4-TYPEs
ActiveFC4Types	uint16		FC-GS FC4-TYPEs
SupportedCOS	uint16		FC-GS Class Of Service
ActiveCOS	uint16		FC-GS Class Of Service

Table 43: FCPort Derivation

C.33 **FCPortStatistics**

(As defined by CIM)

FCPortStatistics is the statistics for the FCPort.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>StatisticalInformation</i>			
<i>DeviceStatisticalInformation</i>			
CreationClassName	string	key	
SystemName	string	key	
DeviceCreationClassName	string	key	
DeviceID	string	key	
Name	string	key	
<i>NetworkAdapterStatistics</i>			
BytesTransmitted	uint64		FA MIB 3.0 ConnUnitPortStatCountTxElements
BytesReceived	uint64		FA MIB 3.0 connUnitPortStatCountRxElements
PacketsTransmitted	uint64		FA MIB 3.0 connUnitPortStatCountTxObjects
PacketsReceived	uint64		FA MIB 3.0 connUnitPortStatCountRxObjects
<i>FCPortStatistics</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
LIPCount	uint64		FA MIB 3.0 connUnitPortStatCountNumberLinkResets
NOSCount	uint64		FA MIB 3.0 ConnUnitPortStatCountNumberOfflineSequences
ErrorFrames	uint64		
DumpedFrames	uint64		
LinkFailures	uint64		FA MIB 3.0 connUnitPortStatCountLinkFailures
LossOfSyncCounter	uint64		FAMIB30 connUnitPortStatCountLossofSynchronization
LossOfSignalCounter	uint64		FA MIB 3.0 connUnitPortStatCountLossofSignal
PrimitiveSeqProtocolErCount	uint64		FA MIB 3.0 connUnitPortStatCountPrimitiveSequenceProtocolErrors
CRCError	uint64		FA MIB 3.0 connUnitPortStatCountInvalidCRC
InvalidTransmissionWords	uint64		FA MIB 3.0 connUnitPortStatCountInvalidTxWords
FramesTooShort	uint64		FA MIB 3.0 connUnitPortStatCountFramesTruncated
FramesTooLong	uint64		FA MIB 3.0 connUnitPortStatCountFramesTooLong
AddressErrors	uint64		FA MIB 3.0 connUnitPortStatCountAddressErrors
BufferCreditNotProvided	uint64		FA MIB 3.0 connUnitPortStatCountInputBuffersFull
DelimiterErrors	uint64		FA MIB 3.0 connUnitPortStatCountDelimiterErrors
EncodingDisparity	uint64		FA MIB 3.0 connUnitPortStatCountEncodingDisparityErrors
LinkResetsReceived	uint64		FA MIB 3.0 connUnitPortStatCountRxLinkResets
LinkResetsTransmitted	uint64		FA MIB 3.0 connUnitPortStatCountTxLinkResets
MulticastFramesReceived	uint64		FA MIB 3.0 connUnitPortStatCountRxMulticastObjects
MulticastFramesTransmitted	uint64		FA MIB 3.0 connUnitPortStatCountTxMulticastObjects

Table 44: FCPortStatistics Derivation

C.34 FRU

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			

Property/Method	Type	Qualifier/Parameter	Description/Notes
Caption	string	MaxLen (64)	
Description	string		
FRU			
FRUNumber	string	Key, MaxLen (64)	
IdentifyingNumber	string	Key, MaxLen (64)	e.g., SW serial number or HW die number
Name	string	MaxLen (256)	
Vendor	string	Key, MaxLen (256)	
RevisionLevel	string	MaxLen (64)	

Table 45: FRU Derivation

C.35 ForwardingService

The *ForwardingService* represents the functions used in forwarding network traffic. Its instances act on packets received from one or more *ProtocolEndpoints* or *Services*, and drop (discard), or send those packets to one or more other *ProtocolEndpoints* or *Services*. The explicit Endpoints being forwarded between, are described using the *ForwardsAmong* association (or one of its subclasses).

Property/Method	Type	Qualifier or Parameter	Req	Notes
<i>ManagedElement</i>				
<i>ManagedSystemElement</i>				
<i>LogicalElement</i>				
<i>Service</i>				
SystemCreationClassName	string	MaxLen (256),Key,Propagated		
SystemName;	string	MaxLen (256),Key,Propagated		
CreationClassName	string	MaxLen (256),Key		
Name	string	MaxLen (256),Key,override		
StartMode	String	MaxLen (10)		See MOF
Started	boolean			
StartService()	uint32			
StopService()	uint32			
<i>NetworkingService</i>				
Keywords[]	string			
ServiceURL	string			
StartupConditions []	string			
StartupParameters []	string			
<i>ForwardingService</i>				

Property/Method	Type	Qualifier or Parameter	Req	Notes
ProtocolType	unit16			type of protocol (other=1)
OtherProtocolType	String			Valid only if ProtocolType = 1

Table 46: ForwardingService Derivation

C.36 **ForwardsAmong**

The *ForwardsAmong* association class represents the dependency that exists between the ProtocolEndpoints that are used to forward data and the ForwardingService that is performing the forwarding of data.

Property/Method	Type	Qualifier/Parameter	Req	Description/Notes
<i>Dependency</i>				
<i>ServiceSAPDependency</i>				
ForwardAmong				
Antecedent	REF	Key		ProtocolEndpoint reference
Dependent	REF	Key		ForwardingService reference

Table 47: ForwardAmong Association Derivation

C.37 **HostedAccessPoint**

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>Dependency</i>			
HostedAccessPoint			
Antecedent	REF	Override	System Reference
Dependent	REF	Override	RemoteServiceAccessPoint Reference

Table 48: HostedAccessPoint Derivation

C.38 **HostedCollection**

(As defined by 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.

(As refined by Bluefin)

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>Dependency</i>			
HostedCollection			
Antecedent	REF	Key, Min(1), Max(1)	ComputerSystem
Dependent	REF	Key, weak	LogicalPortGroup

Table 49: HostedCollection Inheritance

C.39 HostedService

(As defined by CIM)

HostedService is an association between a Service and the System on which the functionality resides. The cardinality of this association is 1-to-many. A System may host many Services. 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. The model does not represent Services hosted across multiple systems. This is modeled as an ApplicationSystem that acts as an aggregation point for Services, that are each located on a single host.

(As refined by Bluefin)

Property/Method	Type	Qualifier/Parameter	Description/Notes
[Association] Dependency			
[Association] HostedService			
Antecedent	REF	Override	System Reference
Dependent	REF		Service Reference

Table 50: HostedService Derivation

C.40 HostedStoragePool

SystemStoragePool is a specialization of SystemComponent association that establishes that the StoragePool is defined in the context of the System.

Property/Method	Type	Qualifier/Parameter	Description/Notes
[Aggregation] Component			
[Aggregation] SystemComponent			
[Aggregation] HostedStoragePool			
GroupComponent	REF	Override	System Reference
Partcomponent	REF	Override	StoragePool Reference

Table 51: HostedStoragePool Derivation

C.41 IndicationFilter

IndicationFilter defines the criteria for generating an Indication and what data should be returned in the Indication. It is derived from ManagedElement to allow modeling the dependency of the filter on a specific service.

Property/Method	Type	Qualifier/Parameter	Description/Notes
ManagedElement			
IndicationFilter			

Property/Method	Type	Qualifier/Parameter	Description/Notes
SystemCreationClassName	string	KEY	The Filter is defined in the context of a System, where it is hosted or to which it applies.
SystemName	string	KEY	
CreationClassName	string	KEY	Indicates the name of the class or the subclass used in the creation of an instance.
Name	string	KEY	The name of the filter.
SourceNameSpace	String		The path to a local namespace where the Indications originate. If NULL, the namespace of the Filter registration is assumed.
Query	String		A query expression that defines the condition(s) under which Indications will be generated.
QueryLanguage	string		

Table 52: IndicationFilter Derivation

C.42 *IndicationHandler*

IndicationHandler is an abstract superclass describing how an Indication is to be processed/delivered/'handled'. This may define a destination and protocol for delivering Indications, or it may define a process to invoke. This class is derived from *ManagedElement* to allow modeling the dependency of the Handler on a specific service.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>IndicationHandler</i>			
SystemCreationClassName	string	KEY	The Filter is defined in the context of a System, where it is hosted or to which it applies
SystemName	string	KEY	
CreationClassName	string	KEY	Indicates the name of the class used in the creation of an instance.
Name	string	KEY	The name of the filter.
Owner	String		The name of the entity that created and/or maintains this Handler.

Table 53: IndicationHandler Derivation

C.43 *IndicationHandlerCIM-XML*

IndicationHandlerCIM-XML describes the destination for Indications to be delivered via HTTP, using a CIM-XML representation.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
IndicationHandler			
SystemCreationClassName	string	KEY	The Filter is defined in the context of a System, where it is hosted or to which it applies
SystemName	string	KEY	
CreationClassName	string	KEY	Indicates the name of the class or the subclass used in the creation of an instance.
Name	string	KEY	The name of the filter.
Owner	String		The name of the entity that created and/or maintains this Handler.
IndicationHandlerCIM-XML			
Destination	string	Required	The destination URL to which HTTP/CIM-XML Indication messages are to be delivered. The scheme prefix is implied and is not required, but must be 'http:' if specified.

Table 54: IndicationHandlerCIM-XML Derivation

C.44 IndicationSubscription

(As defined by CIM)

IndicationSubscription describes a flow of Indications. The flow is specified by the referenced Filter, and directed to the referenced destination or process in the Handler.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Association] IndicationSubscription			
Filter	REF	Key	IndicationFilter Reference
Handler	REF	Key	IndicationHandler Reference

Table 55: IndicationSubscription Association Derivation

C.45 InstalledSoftwareElement

(As defined by CIM)

The InstalledSoftwareElement association allows one to identify the Computer System a particular Software element is installed on.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Association] Dependency			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
[Association] InstalledSoftwareElement			

Property/Method	Type	Qualifier/Parameter	Description/Notes
Antecedent	REF	Override	SoftwareElement Reference
Dependent	REF	Override	ComputerSystem Reference

Table 56: InstalledSoftwareElement Derivation

C.46 InstCreation

(As defined by CIM)

CIM_InstCreation notifies when an instance is created.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>Indication</i>			
IndicationTime	datetime		Date and Time of Indication creation
<i>InstIndication</i>			
SourceInstance	string	EmbeddedObject, Required	Copy if Instance that changed to regenerate the Indication.
<i>InstCreation</i>			

Table 57: InstCreation Derivation

C.47 InstDeletion

(As defined by CIM)

CIM_InstDeletion notifies when an instance is deleted.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>Indication</i>			
IndicationTime	datetime		Date and Time of Indication creation
<i>InstIndication</i>			
SourceInstance	string	EmbeddedObject, Required	Copy if Instance that changed to regenerate the Indication.
<i>InstDeletion</i>			

Table 58: InstDeletion Derivation

C.48 InstModification

(As defined by CIM)

CIM_InstModification notifies when an instance is modified.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>Indication</i>			
IndicationTime	datetime		Date and Time of Indication creation
<i>InstIndication</i>			
SourceInstance	string	EmbeddedObject, Required	Copy if Instance that changed to regenerate the Indication.
<i>InstModification</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
PreviousInstance	string	EmbeddedObject, Required	A copy of the 'previous' instance whose change generated the Indication

Table 59: InstModification Derivation

C.49 InterLibraryPort

(As defined by CIM)

InterLibraryPort s represent hardware that transports Physical Media between connected StorageLibraries. The LibraryExchange association identifies the connected Libraries, by identifying the connected InterLibraryPort s.

Property/ Method	Type	Qualifier or Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>MediaTransferDevice</i>			
<i>InterLibraryPort</i>			
LastAccessed	datetime		
ImportCount	Uint64	counter	
ExportCount	Uint64	counter	
Direction	Uint16 enum	"Unknown", "Import", "Export", "Both Import and Export"	

Table 60: InterLibraryPort Derivation

C.50 LibraryExchange

(As defined by CIM)

LibraryExchange indicates that two StorageLibraries are connected through their InterLibraryPort s.

Property/ Method	Type	Qualifier or Parameter	Description/Notes
<i>[Association] Dependency</i>			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
<i>[Association] LibraryExchange</i>			
Antecedent	REF	Override	InterLibraryPort Reference
Dependent	REF	Override	InterLibraryPort Reference

Table 61: LibraryExchange Derivation

C.51 LibraryPackage

(As defined by CIM)

Similar to the way that LogicalDevices are 'Realized' by PhysicalElements, a StorageLibrary can be realized in one or more PhysicalPackage s. The LibraryPackage association explicitly defines this relationship.

Property/Method	Type	Qualifier or Parameter	Notes
<i>[Association] Dependency</i>			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
<i>[Association] LibraryPackage</i>			
Antecedent	REF	Override	PhysicalPackage Reference
Dependent	REF	Override	StorageLibrary Reference

Table 62: LibraryPackage Derivation

C.52 LimitedAccessPort

(As defined by CIM)

LimitedAccessPort s represent hardware that transports Physical Media into or out of a System, such as a StorageLibrary. They are identified as 'limited' since these Ports do not provide access to ALL the PhysicalMedia or StorageMediaLocations in a Library, but only to a subset.

Property/Method	Type	Qualifier or Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>MediaTransferDevice</i>			
<i>LimitedAccessPort</i>			
Locked	boolean		
Extended	boolean		
ExtendTimeout	Uint32	Units ("Seconds")	
LastExtended	datetime		
ImportCount	Uint64	counter	
ExportCount	Uint64	counter	
Direction	Uint16 enum	"Unknown", "Import", "Export", "Both Import and Export"	

Table 63: LimitedAccessPort Derivation

C.53 LogicalDevice

(As defined by CIM)

An abstraction or emulation of a hardware entity, that may or may not be Realized in physical hardware. Any characteristics of a LogicalDevice that are used to manage its operation or configuration are contained in, or associated with, the LogicalDevice object. Examples of the operational properties of a Printer would be paper sizes supported, or detected errors. Examples of the configuration properties of a Sensor Device would be threshold settings. Various configurations could exist for a LogicalDevice. These configurations could be contained in Setting objects and associated with the LogicalDevice.

(As used in Bluefin Profiles)

LogicalDevice is often used to represent an example of any concrete subclass of LogicalDevice (such as StorageVolume, TapeDrive, ...).

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
SystemCreationClassName	string	MaxLen(256), Key	The scoping System's CreationClassName.
SystemName	string	MaxLen(256), Key	The scoping System's Name.
CreationClassName	string	MaxLen(256), Key	The name of the concrete subclass
DeviceID	string	MaxLen(64), Key	unique identifying information
PowerManagementSupported	boolean		
PowerManagementCapabilities	Int16[]		
Availability	Int16		
StatusInfo	Int16		
LastErrorCode	UInt32		
ErrorDescription	string		
ErrorCleared	boolean		
OtherIdentifyingInfo	String[]		
PowerOnHours	UInt64		
TotalPowerOnHours	UInt64		
IdentifyingDescriptions	String[]		
AdditionalAvailability	UInt16[]		
MaxQuiesceTime	UInt64		

Table 64: LogicalDevice Derivation

C.54 LogicalModule

(As defined by CIM)

LogicalModule is the logical device corresponding to a line card/blade in a device. For example, a line card in a switch is an instance of LogicalModule, associated with the switch itself. A logical module is not necessarily independently managed.")

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
DeviceID	string	Key, MaxLen (64)	
<i>LogicalModule</i>			
ModuleNumber	Uint16		

Table 65: LogicalModule Derivation

C.55 LogicalNetwork

(As defined by CIM)

A LogicalNetwork groups together a set of ProtocolEndpoints of a given type which are able to communicate with each other directly. It is used for describing the characteristics of the grouping and/or its associated medium. A LogicalNetwork represents the ability to send and/or receive data over a network.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
InstanceName	String		Node Symbolic Name
<i>Collection</i>			
CollectionMSEs			
CollectionID	String	key	Node WWN FC-GS InterconnectElement.Name, REQUIRED
<i>LogicalNetwork</i>			
SystemCreationClassName	string	propagated, key	REQUIRED
SystemName	string	propagated, key	REQUIRED

Table 66: LogicalNetwork Derivation

C.56 LogicalPortGroup

(As defined by CIM)

A collection of one or more ports logically grouped for administrative purposes. This class is created for specific ease of query when a Port is associated to more than one SystemSpecificCollection. In FibreChannel, this is the case (e.g. Node, Zone, ZoneSet).

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
InstanceName	String		Node Symbolic Name
<i>Collection</i>			
<i>SpecificCollection</i>			
SystemCreationClassName	string	propagated, key	REQUIRED
SystemName	string	propagated, key	REQUIRED
InstanceID	String	Key	Node WWN FC-GS InterconnectElement.Name, REQUIRED
<i>LogicalPortGroup</i>			

Table 67: LogicalPortGroup Derivation

C.57 MemberOfCollection

(As defined by CIM)

MemberOfCollection is an aggregation used to establish membership of ManagedElements in a Collection.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>MemberOfCollection</i>			
Collection	Collection	Key	LogicalPortGroup
Member	ManagedElement	Key	FCPort

Table 68: MemberOfCollection Inheritance

C.58 ModulePort

(As defined by CIM)

ModulePort associates ports with their hosting modules.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>CIM_SystemComponent</i>			
<i>CIM_ModulePort</i>			
GroupComponent	REF	Key	LogicalModule
PartComponent	REF	Key	FCPort

Table 69: ModulePort Derivation

C.59 ObjectManager

(As defined by CIM)

"A type of WBEMService that defines the capabilities of the CIM Server in which this ObjectManager class resides. Details related to communicating with the ObjectManager, and the Manager's basic capabilities, are accessed through the associated ObjectManagerCommunicationCapabilities class available through the CommMechanismForManager association.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	
Description	string		
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>SoftwareElement</i>			
Name	string	Key, MaxLen (256), Override	
Version	string	Key, Maxlen (64)	<Major>.<Minor>.<Revision> or <Major><Minor><letter><revision>
SoftwareElementState	uint16	Key, Values {}	"Deployable", "Installable", "Executable", "Running"
SoftwareElementID	string	Key, MaxLen (256)	
TargetOperatingSystem	uint16	Key, Values {}	"WINNT", "Windows 2000", "Solaris", ...
OtherTargetOS	string		If TargetOperatingSystem = "Other"
Manufacturer	string	Maxlen (256)	
BuildNumber	string	Maxlen (64)	
SerialNumber	string	Maxlen (64)	
CodeSet	string	Maxlen (64)	
IdentificationCode	string	Maxlen (64)	Manufacturer's SKU or part number
LanguageEdition	string	Maxlen (32)	Use language codes defined in ISO 639
<i>ObjectManager</i>			

Table 70: ObjectManager Derivation

C.60 PackgedComponent

(As defined by CIM)

A Component is typically contained by a **PhysicalPackage** , such as a **Chassis** or **Card**. The **PackagedComponent** association makes this relationship explicit. In the first sentence, the word, typically, is used. This is because a Component may be removed from, or not yet inserted into, its containing Package (i.e., the **Removable** Boolean is **TRUE**). Therefore, a Component may not always be associated with a **Container**.

Property/ Method	Type	Qualifier or Parameter	Notes
<i>[Association] Dependency</i>			
<i>[Association] PackagedComponent</i>			
Antecedent	REF	Override	PhysicalPackage Reference
Dependent	REF	Override	PhysicalComponent Reference

Table 9: PackagedComponent Derivation

C.61 **PhysicalConnector**

(As defined by CIM)

The **PhysicalConnector** class represents any **PhysicalElement** that is used to connect to other Elements. Any object that can be used to connect and transmit signals or power between two or more **PhysicalElements** is a descendant (or member) of this class. For example, Slots and D-shell connectors are types of **PhysicalConnectors**.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>PhysicalElement</i>			
Manufacturer			
Model			
SerialNumber			
Version			
<i>PhysicalConnector</i>			
ConnectorType			FC-GS Port.Module Type
OtherTypeDescription			

Table 71: PhysicalConnector Derivation

C.62 **PhysicalMedia**

(As defined by CIM)

The **PhysicalMedia** class represents any type of documentation or storage medium, such as tapes, CDROMs, etc. This class is typically used to locate and manage Removable Media (versus Media sealed with the **MediaAccessDevice**, as a single Package, as is the case with hard disks). However, 'sealed' Media can also be modeled using this class, where the Media would then be associated with the **PhysicalPackage** using the **PackagedComponent** relationship.

(As refined by Bluefin)

Provides properties of media. For disks, few properties are used, but it does provide associations for the 1 or more extents that reside on the media.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
Caption	String	MaxLen(64)	Short (one line) description
Description	String		Longer description
ManagedSystemElement			
Name	String	MaxLen(256)	
Status	String	MaxLen(10)	
PhysicalElements			
Tag	String	MaxLen(256), Key	An arbitrary string that uniquely identifies the Physical Element
CreationClassName	String	MaxLen(256). Key	The name of the concrete subclass
Manufacturer	String	MaxLen(256)	
Model	String	MaxLen(64)	
SKU	String	MaxLen(64)	
SerialNumber	String	MaxLen(64)	
Version	String	MaxLen(64)	
Partnumber	String	MaxLen(256)	
OtherIdentifyingInfo	String		
PoweredOn	Boolean		
ManufactureDate	datetime		
PhysicalComponet			
Removable	Boolean		
Replaceable	Boolean		
HotSwappable	Boolean		
PhysicalMedia			
Capacity	UInt64		
MediaType	UInt16		
MediaDescription	string		
WriteProtectOn	boolean		
CleanerMedia	boolean		
MediaSize	Real32		
MaxMounts	UInt64		
MountCount	UInt64		
DualSided	boolean		
PhysicalLabels	String[]		
LabelStates	UInt16[]		
LabelFormats	UInt16[]		
TimeOfLastMount	datetime		
TotalMountTime	Unit64		

Table 72: PhysicalMedia Derivation

C.63 PhysicalMediaInLocation

(As defined by CIM)

Within a `StorageLibrary`, all Media should be accounted for, and be present in some Storage Location. This relationship is made explicit by the `PhysicalMediaInLocation` association. In addition, one can determine if a Location is empty or full based on whether this association exists for the `StorageMediaLocation`.

Property/Method	Type	Qualifier or Parameter	Notes
<i>[Association] Dependency</i>			
<i>[Association] PhysicalMediaInLocation</i>			
Antecedent	REF	Override	StorageMediaLocation Reference
Dependent	REF	Override	PhysicalMedia Reference
Orientation	Uint16	"Unknown", "Side 0", "Side 1", "Side A", "Side B", "Not Applicable"	

Table 9: `PhysicalMediaInLocation` Derivation

C.64 **PhysicalPackage**

(As defined by CIM)

The `PhysicalPackage` class represents `PhysicalElements` that contain or host other components. Examples are a Rack enclosure or an adapter Card.

(As refined by Bluefin)

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>PhysicalElement</i>			
Manufacturer			FC-GS InterconnectElement.Information List.Vendor Name
Model			FC-GS InterconnectElement.Information List.Model Name/Number
SerialNumber			
Version			FC-GS InterconnectElement.Information List.Release Code
PartNumber			
<i>PhysicalPackage</i>			

Table 73: `PhysicalPackage` Derivation

C.65 **PhysicalTape**

(As defined by CIM)

The `PhysicalTape` class represents additional data for a Tape Media. Information on the tape length and whether it must be unloaded from BOT are properties of this class.

Property/Method	Type	Qualifier or Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>MediaAccessDevice</i>			
<i>PhysicalTape</i>			
TapeLength	Real32	Units ("Feet")	
UnloadAnywhere	boolean		

Table 10: PhysicalTape Derivation

C.66 PortImplementsEndpoint

The *PortImplementsEndpoint* association class associates a LogicalPort with one or more ProtocolEndpoints that are implemented 'on it'.

Property/Method	Type	Qualifier/Parameter	Req	Description/Notes
<i>Dependency</i>				
<i>ServiceSAPDependency</i>				
<i>ForwardAmong</i>				
Antecedent	REF	Key		LogicalPort reference
Dependent	REF	Key		ProtocolEndpoint reference

Table 74: PortImplementsEndpoint Association Derivation

C.67 Product

(As defined by CIM)

Product is a concrete class that is a collection of PhysicalElements, SoftwareFeatures and/or other Products, acquired as a unit. Acquisition implies an agreement between supplier and consumer which may have implications to Product licensing, support and warranty. Non-commercial (e.g., in-house developed Products) should also be identified as an instance of Product.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Product</i>			
IdentifyingNumber	string	Key, MaxLen (64)	e.g., SW s/n, HW die, RYO project number
Name	string	Key, MaxLen (256)	Commonly used product name FC-GS InterconnectElement.Model Name/Number
Vendor	string	Key, MaxLen (256)	FC-GS InterconnectElement.Vendor Name
Version	string	Key, MaxLen (64)	FC-GS InterconnectElement.Release Code

Table 75: Product Derivation

C.68 ProductPhysicalElements

(As defined by CIM)

Indicates the PhysicalElements that make up a Product.

Property/Method	Type	Qualifier/Parameter	Notes/Description
<i>[Association] Aggregation</i>			
Antecedent	REF	Key	ManagedElement Reference
Dependent	REF	Key	ManagedElement Reference
<i>[Association] ProductPhysicalElements</i>			
Antecedent	REF	Override	Product Reference
Dependent	REF	Override	PhysicalElement Reference

Table 76: ProductPhysicalElements Derivation

C.69 ProtocolEndpoint

A *ProtocolEndpoint* represents a communication service used to link ports in *LogicalNetworks*. *LogicalNetworks* collect a set of *ProtocolEndpoints* of a given type that can send or receive data over a network using the *InLogicalNetwork* association. Each *ProtocolEndpoint* is associated with the *FibrePort* using the *DeviceSAPImplementation* class. Although a suitable specialization, *PortImplementsEndpoint*, specialization is defined, it is preferable to use its superclass to accommodate the possible coalescence of *FibrePort* and *FibreChannelAdapter*.

A Fibre Channel Adapter port that supports simultaneous use of multiple upper layer protocols uses a separate *ProtocolEndpoint* class to represent communication using each protocol. Refer to "Upper Layer Protocol Support" for additional information.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>ServiceAccessPoint</i>			
Name	string	MaxLen (256)	
CreationClassName	string	Key, MaxLen (256)	
SystemCreationClassName	string	Key, MaxLen (256)	
SystemName	string	Key, MaxLen (256)	
<i>ProtocolEndpoint</i>			
NameFormat	string	MaxLen (256)	heuristic that ensures unique name
ProtocolType	string	MaxLen (64), ValueMap { } Values { }	"IPv4", "Fibre Channel", "InfiniBand", ...
OtherTypeDescription	string	MaxLen (64)	used when ProtocolType = "Other"

Table 77: ProtocolEndpoint Derivation

C.70 ProvidesServiceToElement

(As defined by CIM)

ProvidesServiceToElement is used to describe that ManagedElements may be dependent on the functionality of one or more Services. An example is that a Processor and an Enclosure (PhysicalElement) are dependent on AlertOnLAN Services to signal an incomplete or erroneous boot, and hardware-related errors.

(As refined by Bluefin)

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>[Association] Dependency</i>			
<i>[Association] ProvidesServiceToElements</i>			
Antecedent	REF	Override	Service Reference
Dependant	REF	Override	ManagedElement Reference

Table 78: ProvidesServiceToElements Derivation

C.71 Realizes

(As defined by CIM)

Realizes is the association that defines the mapping between a Logical Device and the physical component that implements the Device.

Property/Method	Type	Qualifier/Parameter	Notes/Description
<i>[Association] Dependency</i>			
<i>[Association] Realizes</i>			
Antecedent	REF	Override	PhysicalElement Reference
Dependent	REF	Override	LogicalDevice Reference

Table 79: Realizes Derivation

C.72 RedundancyComponent

(As defined by CIM)

The RedundancyComponent association indicates that this set of fans' or 'these physical extents' participate in a single RedundancyGroup.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>[Aggregation] Component</i>			
PartComponent	REF	KEY	ManagedElement Reference
<i>[Aggregation] RedundancyComponent</i>			
GroupComponent	REF	Override	RedundancyGroup Reference

Table 80: RedundancyComponent Derivation

C.73 RedundancyGroup

(As defined by CIM)

A class derived from LogicalElement that is a special collection of ManagedSystemElements. This collection indicates that the aggregated components together provide redundancy. All elements aggregated in a RedundancyGroup should be instantiations of the same object class.

(As refined by Bluefin)

Indicates that underlying extents in BasedOn associations provide redundancy. Absence of this class indicates lack of redundancy (for example, concatenation or RAID 0).

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
RedundancyGroup			
CreationClassName	string	MaxLen(256)	The name of the concrete subclass
Name	string	MaxLen(256),override	
RedundancyStatus	Uint16		

Table 81: RedundancyGroup Derivation

C.74 RemoteServiceAccessPoint

(As defined by CIM)

RemoteServiceAccessPoint describes access and/or addressing information for a remote connection, that is known to a 'local' network element. This information is scoped/contained by the 'local' network element, since this is the context in which it is 'remote'.

Why the remote access point is relevant and information on its use are described by subclassing RemoteServiceAccessPoint, or by associating to it.")

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>ServiceAccessPoint</i>			
SystemName	string	key	
SystemCreationClassName	string	key	
CreationClassName	string	key	
Name	string	key	
RemoteServiceAccessPoint			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
AccessInfo	string		FCGS InterconnectElement.Management Addresss
InfoFormat	uint16		
OtherInfoFormatDescription	string		

Table 82: RemoteServiceAccessPoint

C.75 *SCSIController*

(As defined by CIM)

Capabilities and management of the SCSIController.

(As refined by Bluefin)

Typically, each FCPort has a 1-1 association to a SCSIController which has the SCSI (as opposed to the FC) aspects of the port.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
SystemCreationClassName	string	MaxLen(256), Key	The scoping System's CreationClassName.
SystemName	string	MaxLen(256), Key	The scoping System's Name.
CreationClassName	string	MaxLen(256), Key	The name of the concrete subclass
DeviceID	string	MaxLen(64), Key	unique identifying information
PowerManagementSupported	boolean		
PowerManagementCapabilities	Int16[]		
Availability	Int16		
StatusInfo	Int16		
LastErrorCode	UInt32		
ErrorDescription	string		
ErrorCleared	boolean		
OtherIdentifyingInfo	String[]		
PowerOnHours	UInt64		
TotalPowerOnHours	UInt64		
IdentifyingDescriptions	String[]		
AdditionalAvailability	UInt16[]		
MaxQuiesceTime	UInt64		
<i>Controller</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
TimeOfLastReset	datetime		
ProtocolSupported	Uint16		
MaxNumberControlled	Uint32		
ProtocolDescription	string		
SCSIController			
Name	string	MaxLen (256), Override, Key	
Version	string	Maxlen (64), Key	<Major>.<Minor>.<Revision> or <Major> <Minor> <letter> <revision>
ProtectionManagement	Int16		
MaxDataWidth	Uint32		
MaxTransferRate	Uint64		
ControllerTimeouts	Uint32		
SignalCapabilities	Uint16		

Table 83: SCSIController Derivation, General Case

Property/ Method	Type	Qualifier/ Parameter	Notes/Description
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	
Description	string		
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>Controller</i>			
SCSIController			
Name	string	Key, MaxLen (256), Override	
Version	string	Key, Maxlen (64)	<Major>.<Minor>.<Revision> or <Major> <Minor> <letter> <revision>

Table 84: SCSIController Derivation, Alternate Case

C.76 **SCSIInterface** (As defined by CIM)

SCSIInterface is a **ControlledBy** relationship indicating which Devices are accessed through a SCSIController and the characteristics of this access.

(As refined by Bluefin)

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Association] Dependency			
[Association] DeviceConnection			
NegotiatedSpeed	Unit64		
NegotiatedDataWidth	Unit32		
[Association] ControlledBy			
Dependent	REF	Override	LogicalDevice Reference
AccessState	Unit16		
TimeOfDeviceReset	datetime		
NumberOfHardResets	Unit32		
NumberOfSoftResets	Unit32		
[Assosiation] SCSIInterface			
Antecedent	REF	Override	SCSIController Reference
SCSITimeouts	Unit32		
SCSIRetries	Unit32		
InitiatorId	Unit32		
TargetId	Unit32		
TargetLUN	Unit64		
SCSIReservation	Unit16		
SCSISignal	Unit16		
MaxQueueDepth	Unit32		
QueueDepthLimit	Unit32		

Table 85: SCSIInterface Derivation, General Case

C.77 SC SILUN

(As defined by CIM)

The association indicates a relationship between a Storage Volume exposed as a LUN through a slave SCSI Controller. A new relationship is required in order to distinguish

between the 'Host' connection (represented by SCSIInterface) and the 'Target' connection (represented by SC SILUN).

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Association] Dependency			
[Association] DeviceConnection			
NegotiatedSpeed	Unit64		
NegotiatedDataWidth	Unit32		
[Association] ControlledBy			
Dependent	REF	Override	LogicalDevice Reference
AccessState	Unit16		
TimeOfDeviceReset	datetime		
NumberOfHardResets	Unit32		
NumberOfSoftResets	Unit32		
[Assosiation] SC SILUN			
DeviceNumber	Unit64		

Table 86: SC SILUN Derivation

C.78 Service

(As defined by CIM)

A Service is a Logical Element that contains the information necessary to represent and manage the functionality provided by a Device and/or SoftwareFeature. A Service is a general-purpose object to configure and manage the implementation of functionality. It is not the functionality itself.

Property/Method	Type	Qualifier/Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
Service			
SystemCreationClassName	string	MaxLen (256),Key,Propagated	
SystemName;	string	MaxLen (256),Key,Propagated	
CreationClassName	string	MaxLen (256),Key	
Name	string	MaxLen (256),Key,override	
StartMode	String	MaxLen (10)	See MOF
Started	boolean		
StartService()	uint32		
StopService()	uint32		

Table 87: Service Derivation

C.79 ServiceAccessBySAP

(As defined by CIM)

ServiceAccessBySAP is an association that identifies the access points for a Service. For example, a printer may be accessed by Netware, Macintosh or Windows ServiceAccessPoints, potentially hosted on different Systems.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>[Association] Dependency</i>			
[Association] ServiceAccessBySAP			
Antecedent	REF	Override	System Reference
Dependant	REF	Override	ServiceAccessPoint Reference

Table 88: ServiceAccessBySAP Derivation

C.80 *Setting*

The *Setting* class represents configuration-related and operational parameters for one or more *ManagedSystemElement*(s). A *ManagedSystemElement* may have multiple *Setting* objects associated with it. The current operational values for an *Element*'s parameters are reflected by properties in the *Element* itself or by properties in its associations. "

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Setting</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
SettingID	Maxlen(256)	Key	The identifier by which the Setting object is known.
VerifyOKToApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The VerifyOKToApplyToMSE method is used to verify that this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The ApplyToMSE method performs the actual application of the Setting to the referenced ManagedSystemElement.
VerifyOKToApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The VerifyOKToApplyToCollection method is used to verify that this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The ApplyToCollection method performs the application of the Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyToMSE method against each of the Elements aggregated by the Collection.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyIncreme	Uint32	[IN]	The ApplyIncrementalChangeToMSE

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ntalChangeToMSE()		CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	method performs the actual application of a subset of the properties in the Setting to the referenced ManagedSystemElement.
VerifyOKToApplyIncrementalChangeToCollection ()	Uint32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The VerifyOKToApplyIncrementalChangeToCollection method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyIncrementalChangeToCollection()	Uint32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The ApplyIncrementalChangeToCollection method performs the application of a subset of the properties in this Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyIncrementalChangeToMSE method against each of the Elements aggregated by the Collection.

Table 89: Setting Derivation

C.81 **SettingContext**

This relationship associates **Configuration** objects with **Setting** objects. For example, a **NetworkAdapter**'s Settings could change based on the site/network to which its hosting **ComputerSystem** is attached. In this case, the **ComputerSystem** would have two different **Configuration** objects, corresponding to the differences in network configuration for the two network segments. **Configuration A** would aggregate a **Setting** object for the **NetworkAdapter** when operating on segment "ANet", whereas **Configuration B** would aggregate a different **NetworkAdapter** Setting object, specific to segment "BNet". Note that many Settings of the computer are independent of the network **Configuration**. For example, both **Configurations A** and **B** would aggregate the same **Setting** object for the **ComputerSystem**'s **MonitorResolution**.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Aggregation]SettingContext			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
Context	REF	Override	Configuration Reference
Setting	REF	Override	Setting Reference

Table 90: SettingContext Aggregation Derivation

C.82 **SharedSecretService**

(As defined by CIM)

SharedSecretService is a service which ascertains whether messages received are from the Principal with whom a secret is shared. Examples include a login service that proves identity on the basis of knowledge of the shared secret, or a transport integrity service (like Kerberos provides) that includes a message authenticity code that proves each message in the message stream came from someone who knows the shared secret session key.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>Service</i>			
Name	string	Key	
<i>SecurityService</i>			
<i>AuthenticationService</i>			
<i>CredentialService</i>			
<i>LocalCredentialManagementService</i>			
<i>SharedSecretService</i>			

Table 91: SharedSecretService Derivation

C.83 **SharedSecret**

(As defined by CIM)

SharedSecret is the secret shared between a Users Access and a particular SharedSecret security service. Secrets may be in the form of a password used for initial authentication, or as with a session key, used as part of a message authentication code to verify that a message originated by the principal with whom the secret is shared.

It is important to note that SharedSecret is not just the password, but rather is the password used with a particular security service.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Credential</i>			
<i>SharedSecret</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
SystemCreationClassName	string	Propagated, Key	REQUIRED
SystemName	string	Propagated, Key	REQUIRED
ServiceCreationClassName	string	Propogated, Key, MaxLen (256)	REQUIRED
ServiceName	string	Propogated, Key, MaxLen (256)	REQUIRED
RemoteID	string	Key, MaxLen (256)	the name by which the user is known at the remote secret key authentication service
Secret	String		Secret known by the User Access
Algorithm	string		transformation algorithm, if any, used to protect passwords before use in the protocol
Protocol	String		the protocol with which the SharedSecret is used

Table 92: SharedSecret Derivation

C.84 **SoftwareElement**

Multiple components of firmware and software are used to provide Fibre Channel Adapter capabilities. Asset and state information about each component is provided by a separate *SoftwareElement* class. The type of each *SoftwareElement* is indicated by the *DeviceSoftware* association class. Each *SoftwareElement* may be associated with multiple *FibreChannelAdapters* by instantiating a separate *DeviceSoftware* class for each association.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
SoftwareElement			
Name	string	Key, MaxLen (256), Override	
Version	string	Key, Maxlen (64)	<Major>.<Minor>.<Revision> or <Major><Minor><letter><revision>
SoftwareElementState	uint16	Key, Values {}	"Deployable", "Installable", "Executable", "Running"
SoftwareElementID	string	Key, MaxLen (256)	
TargetOperatingSystem	uint16	Key, Values {}	"WINNT", "Windows 2000", "Solaris", ...
OtherTargetOS	string		If TargetOperatingSystem = "Other"
Manufacturer	string	Maxlen (256)	

Table 93: SoftwareElement Derivation

C.85 **SpareGroup** (As defined by CIM)

A class derived from RedundancyGroup indicating that one or more of the aggregated elements can be spared. (The actual Spares are defined using the ActsAsSpare association.) An example is the use of redundant NICs in a ComputerSystem - where one NIC is primary and the other is Spare. The 'primary' NIC would be a member of the SpareGroup (associated using the RedundancyComponent class) and the 'other' NIC would be associated using the ActsAsSpare relationship.

(As refined by Bluefin)

Represents a collection of spare disks in a disk array.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
RedundancyGroup			
CreationClassName	string	MaxLen(256), Key	The name of the concrete subclass
Name	string	MaxLen(256), override, Key	
RedundancyStatus	Uint16		
SpareGroup			
Failover()	uint32	FailoverFrom, FailoverTo	

Table 94: SpareGroup Derivation

C.86 StorageAccessService

A service providing interfaces to device-based access control (AKA LUN Masking) and setting a Logical Unit number (AKA LUN masking). Methods are defined for exposing LogicalDevices to initiators one at a time, or by manipulating groups of initiator IDs, target controllers, and logical devices. The group commands model behavior in high-end disk arrays that allow an admin to define groups to avoid an explosion of associations.

Property/Method	Type	Qualifier or Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>Service</i>			

Property/ Method		Type	Qualifier or Parameter	Notes
SystemCreationClassName	string	MaxLen (256),Key,Propagated		
SystemName;	string	MaxLen (256),Key,Propagated		
CreationClassName	string	MaxLen (256),Key		
Name	string	MaxLen (256),Key,override		
StartMode	String	MaxLen (10)		See MOF
Started	boolean			
StartService()	uint32			
StopService()	uint32			
StorageAccessService				
Expose()		[in,Description ("An instance of a LogicalDevice that will be exposed to an Initiator")] CIM_LogicalDevice REF Device, [in,Description ("An instance of a target Controller through which the Device (above) will be exposed to an Initiator. A null means all controllers are exposed.")] CIM_Controller REF Target, [in,Description ("The ID of the initiator being granted access. The format of this ID is defined in InitiatorIDFormat.")] string InitiatorID, [in, Description ("The format of the InitiatorID."),ValueMap {"1", "2", "3"}, Values {"PortWWN", "NodeWWN", "Hostname"}] uint16 InitiatorIDFormat, [in, Description ("Access Mode granted"),ValueMap {"1", "2", "3"},Values {"ReadWrite", "ReadOnly", "DefaultAccess"}] uint16 AccessMode		Expose a LogicalDevice to an Initiator via a target Controller. This method modifies the access control in the associated storage system and creates UnitAccess associations that model the granted access.
MapExpose()	Uint32	[in,Description ("The instance of a LogicalDevice that will be exposed to an Initiator")] CIM_LogicalDevice REF Device, [in,Description ("The Unit Number to assign to the Initiator. ")]uint64 Number,		Expose a LogicalDevice to an Initiator via a target Controller and assign a Unit Number (such as a SCSI LUN). This method modifies the access control in the associated storage system and creates a UnitAccess association that models the granted access.

Property/ Method	Type	Qualifier or Parameter	Notes
		[in,Description ("The instance of a Target Controller through which the Device (above) will be exposed to an Initiator. A null means all controller will be mapped and exposed.")] CIM_Controller REF Target, [in,Description ("The ID of the initiator being grated access. The format of this ID is defined in InitiatorIDFormat.")] string InitiatorID, [in, Description ("The format of the IntiatorID."),ValueMap {"1", "2", "3"}, Values {"PortWWN", "NodeWWN", "Hostname"}] uint16 InitiatorIDFormat, [Description("Access Mode granted"), ValueMap {"1", "2", "3"},Values {"ReadWrite", "ReadOnly", "DefaultAccess"}] uint16 AccessMode	
Map()	Uint32	[in,Description ("An instance of a LogicalDevice that will be assigned a unit number for the specified controller. ")] CIM_LogicalDevice REF Device, [in,Description ("The Unit Number to assign. ")]uint64 Number, [in,Description ("An instance of a target Controller through which the Device (above) will be exposed to Initiators. A null means apply to all controllers")] CIM_Controller REF Target, [Description("Access Mode granted"),ValueMap {"1", "2", "3"},Values {"ReadWrite", "ReadOnly", "DefaultAccess"}] uint16 AccessMode	Specify the Unit address assigned to all initiators for a Logical Device.
Deny()	Uint32	[in,Description ("An instance of a LogicalDevice that will be exposed to an Initiator")] CIM_LogicalDevice REF Device, [in,Description ("An instance of	Deny Access from an initiator to a LogicalDevice.

Property/ Method	Type	Qualifier or Parameter	Notes
		<p>a Controller through which the Device (above) will be Denied access from an Initiator. A null means all controllers."}] CIM_Controller REF Controller,</p> <p>[in,Description ("The ID of the initiator being denied access. The format of this ID is defined in InitiatorIDFormat.")] string InitiatorID,</p> <p>[in, Description ("The format of the InitiatorID."), ValueMap {"1", "2", "3"}, Values {"PortWWN", "NodeWWN", "Hostname"}] uint16 InitiatorIDFormat</p>	
UnExpose()	Uint32	<p>[in,Description ("An instance of a LogicalDevice that will be exposed to an Initiator")] CIM_LogicalDevice REF Device,</p> <p>[in,Description ("An instance of a target Controller through which the Device (above) will be unexposed from an Initiator. A null means all controllers are exposed.")] CIM_Controller REF Target,</p> <p>[in,Description ("The ID of the initiator being denied access. The format of this ID is defined in InitiatorIDFormat.")] string InitiatorID,</p> <p>[in, Description ("The format of the InitiatorID."),ValueMap {"1", "2", "3"},Values {"PortWWN", "NodeWWN", "Hostname"}] uint16 InitiatorIDFormat</p>	Invoke hardware mechanism to undo the results of an Expose method and remove the UnitAccess association.
CreateInitiatorID()	Uint32	<p>[in, Description ("The initiator Name")]string Name,</p> <p>[in, Description ("The initiator name format"),ValueMap {"1", "2", "3"},Values {"PortWWN", "NodeWWN", "Hostname"}] uint16 NameFormat,</p>	Create an InitiatorID object.

Property/ Method		Type	Qualifier or Parameter	Notes
			[out, Description ("The new InitiatorID.")] CIM_InitiatorID REF InitiatorID	
RemoveInitiatorID()	UInt32		[in, Description ("The InitiatorID instance to remove.")] CIM_InitiatorID REF InitiatorID	Remove an InitiatorID instance.
CreateCollection()	UInt32		[in, Description ("The type of collection to create"), ValueMap {"1", "2", "3"}, Values {"LogicalDevices", "Controllers", "InitiatorIDs"}] uint16 DesiredCollectionType, [in, Description ("A meaningful name for the collection")] string CollectionName, [out, Description ("The new CIM_Collection.")] CIM_Collection REF Collection	Create a collection of LogicalDevices, Controllers, or InitiatorIDs. The resulting collection is used in the GroupExpose method.
RemoveCollection()	UInt32		[in, Description ("The collection to be removed.")] CIM_Collection REF Collection	Remove a collection.
GroupExpose()	UInt32		[in, Description ("An instance of a Collection of LogicalDevice that will be exposed to an Initiator. A null means all devices.")] CIM_Collection REF DeviceGroup, [in, Description ("An instance of a Collection of target Controller through which the Devices in the DeviceGroup (above) will be exposed to Initiators. A null means all controllers.")] CIM_Collection REF TargetGroup, [in, Description ("An instance of a collection of IDs of initiators being granted access.")] CIM_Collection REF InitiatorIDGroup, [Description ("Access Mode granted"), ValueMap {"1", "2", "3"}, Values {"ReadWrite", "ReadOnly", "DefaultAccess"}] uint16 AccessMode	Expose a group of LogicalDevices to a group of Initiators via a group of target Controllers. This method modifies the access control in the associated storage system and creates UnitAccess associations that model the granted access.
GroupDeny()	UInt32		[in, Description ("An instance of a Collection of LogicalDevice	Deny access from a group of LogicalDevices to a group of

Property/ Method	Type	Qualifier or Parameter	Notes
		that will be exposed to an Initiator")] CIM_Collection REF DeviceGroup, [in,Description ("An instance of a Collection of target Controller through which the Devices in the DeviceGroup (above) will be exposed to Initiators.")] CIM_Collection REF TargetGroup, [in,Description ("An instance of a collection of IDs of initiators being granted access.")] CIM_Collection REF InitiatorIDGroup	Initiators via a group of target Controllers. This method modifies the access control in the associated storage system and creates the GroupAccess association with Deny AccessMode.
GroupUnexpose()	Uint32	[in, Description ("The GroupAccess association to remove.")] CIM_GroupAccess REF GroupAccess	Invoke hardware mechanism to undo the results of a GroupExpose method and remove the GroupAccess instance.
AddMemberToGroup()	Uint32	[in,Description ("The Collection receiving a new member.")] CIM_Collection REF Group, [in,Description ("The member to be added to the Group.")] CIM_ManagedSystemElement REF Member	Add a target, or InitiatorID to a collection.
AddDeviceToGroup()	Uint32	[in,Description ("The Collection receiving a new member.")] CIM_Collection REF Group, [in,Description ("The device to be added to the Group.")] CIM_LogicalDevice REF Device, [in,Description ("The unit number to assign to the device. A device with this number must not already exist in the Group.")] uint64 Number	Add a device to a collection, assigning the ID common to initiators associated to this DeviceGroup.
RemoveMemberFromGroup()	Uint32	[in,Description ("The Collection.")] CIM_Collection REF Group, [in,Description ("The member to be removed from the Group.")] CIM_ManagedSystemElement REF Member	Remove a target controller, Logical device, or InitiatorID from a collection.

Property/ Method	Type	Qualifier or Parameter	Notes
SystemDefaultAccessMode	Uint16		Default Access Mode for this system."),ValueMap {"1", "2", "3"},Values {"ReadWrite", "ReadOnly", "NoAccess"}
SetSystemDefaultAccessMode()	Uint32	[in,Description ("The Access Mode to set."),ValueMap {"1", "2", "3"}, Values {"ReadWrite", "ReadOnly", "NoAccess"}] uint16 AccessMode	Method to set the System DefaultAccessMode.
AvailableMethods[]	string		A list of the methods (above) implemented by this service.

Table 95: StorageAccessService Derivation

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
Caption	string	Maxlen (64)	Short (one line) description
Description	string		Longer description
ManagedSystemElement			
InstallDate	datetime		
Status	string	Maxlen (10)	
LogicalElement			
Service			
SystemCreationClassName	string	Maxlen(256), Key	The scoping System's CreationClassName.
SystemName	string	Maxlen(256), Key	The scoping System's Name.
CreationClassName	string	Maxlen(256), Key	The name of the concrete subclass
Name	string	Maxlen(256), Override, Key	
StartMode	string	Maxlen(10)	
Started	boolean		
StartService()	uint32		
StopService()	unit32		
StorageAccessService			

Table 96: StorageAccessService Alternate Derivation

C.87 StorageCapabilities

The UnitAccess relationship indicates which Devices are exposed through the Target Controller to initiators.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
Capabilities			
StorageCapabilities			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
InstanceType	Uint16		Enumeration indicating the type of instance this Storage Capabilities applies to.
NoSinglePoint OfFailure	boolean		Is No Single Point of Value supported? Possible values are false = can't, true = can do
NoSinglePoint OfFailureDefa ult	boolean		What is the default value for No Single Point of Value? Possible values are false = can't, true = can do
DataRedunda ncyMax	Uint16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter describes the maximum supported value. Possible values are 1 to n
DataRedunda ncyMin	Uint16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter describes the minimum supported value. Possible values are 1 to n
DataRedunda ncyDefault	uint16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter describes the default value. Possible values are 1 to n
SpindleRedun dancyMax	Uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the maximum supported value. Possible values are 0 to n
SpindleRedun dancyMin	Uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the minimum supported value. Possible values are 0 to n
SpindleRedun dancyDefault	Uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
			would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the default value. Possible values are 0 to n
DeltaReservationMin	uint16		Delta reservation is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations. This parameter sets the lower limit
DeltaReservationMax	uint16		Delta reservation is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations. This parameter sets the upper limit
DeltaReservationDefault	uint16		Delta reservation is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations. This parameter sets the default value

Table 97: StorageCapabilities Derivation

C.88 **StorageConfigurationJob**

StorageConfigurationJob is a job that has been scheduled/started by a 'Start' method within the StorageConfigurationService. It contains parameters showing the status of the method execution. The instance must be deleted, once the job completes. Note that Instance Indications can be used to track execution. The status the job should be shown as 'starting' until the instance being created exists. Status will then change to 'OK'. When the job finishes an InstanceIndication may be delivered with Status set to 'stopping' or 'error'

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
ManagedSystemElement			
LogicalElement			
Job			

Property/Method	Type	Qualifier/Parameter	Description/Notes
JobStatus	string		A free form string representing the Job's status.
TimeSubmitted	datetime		Time that the Job was submitted.
StartTime	datetime		Time that the Job was begun.
ElapsedTime	datetime		Length of time that the Job has been executing.
UntilTime	datetime		Time after which the Job is invalid or should be stopped.
Notify	string		User to be notified upon Job completion or failure.
Owner	string		User that submitted the Job.
Priority	uint32		Indicates the urgency or importance of execution of a Job.
<i>StorageConfigurationJob</i>			
SystemCreationClassName	String MaxLen(256)	Key	The scoping Services's SystemCreationClassName.
SystemName	String MaxLen(256)	Key	The scoping services's SystemName.
ServiceCreationClassName	String MaxLen(256)	Key	The scoping Services's CreationClassName.
ServiceName	String	Key	The scoping services's Name.
CreationClassName	String MaxLen(256)	Key	CreationClassName indicates the name of the class or the subclass used in the creation of an instance. When used with the other key properties of this class, this property allows all instances of this class and its subclasses to be uniquely identified.
InstanceID	String	Key	Unique identifies the job within the scoping system
MethodName	String		The name of the method invoked.
PercentComplete	UInt16		Percentage of job complete
ErrorCode	UInt16		Vendor error code. Will be set to 0 if the job completed without error.
ErrorDescription	string		Free form string containing error description.

Table 98: StorageConfigurationJob Class Derivation

C.89 StorageConfigurationService

This service allows the active management of a Storage Server. It allows jobs to be started for the creation, modification and deletion of Storage Objects (storagePools and Storage Volumes)

Property/Method	Type	Qualifier or Parameter	Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			

Property/ Method	Type	Qualifier or Parameter	Notes
<i>LogicalElement</i>			
<i>Service</i>			

Property/ Method		Type	Qualifier or Parameter	Notes
SystemCreationClassName	string	MaxLen (256),Key,Propagated		
SystemName;	string	MaxLen (256),Key,Propagated		
CreationClassName	string	MaxLen (256),Key		
Name	string	MaxLen (256),Key,override		
StartMode	String	MaxLen (10)		See MOF
Started	boolean			
StartService()	uint32			
StopService()	uint32			
StorageConfigurationService				
KillJob()	Unit32	[IN,MappingStrings{"CIM_StorageConfigurationJob.InstanceID"}, Description("Identifier for storage job")] string InstanceID		Kill the StorageConfigurationJob with the specified instance ID, any underlying processes and remove any 'dangling' associations. Note that the job to be killed must be linked to this Service via ExecutingStorageConfigurationJob
CreateStoragePool()	Unit32	[Out, in(false),Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, in,Description ("The an instance of StorageCapabilities to be maintained by the created StoragePool. If set to a null value, the default configuration from the source pool will be used")] CIM_StorageCapabilities ref Goal, [in,out, Description("Size of pool")] uint64 Size, [in, Description ("Array of strings containing representations of references to input CIM_StoragePool instances.")] string InPool[], [in, Description ("Array of strings containing representationsof references to CIM_StorageExtent instances")] string Extent[], [out, in(false), Description		Start a job to create a StoragePool. This function takes a size to attempt to achieve and returns the size achieved. Space is taken from either or both of input StoragePools and Extents. Capabilities attributes that the Pool must support are passed in by the CapabilitiesGoal. if the requested size cannot be created, no action will be taken, the Return Value will be 0x1002 and the output value, size will be set to the nearest possible size. If 0 is returned, no StorageConfigurationJob instance is created.

Property/ Method	Type	Qualifier or Parameter	Notes
		("Handle to pool being created")) CIM_StoragePool ref OutPool	
CreateStorageVolume()	Uint32	[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [in, Description("The definition for the StorageSetting to be maintained by the created StorageVolume. If set to a null value, the default configuration from the source pool will be used")] CIM_StorageSetting ref Goal, [in, out, Description("Size of volume")] uint64 Size, [in, Description("Pool to create volume from")] CIM_StoragePool ref InPool, [out, in(false) Description("Handle to volume being created")] CIM_StorageVolume ref OutVolume	Start a job to create a StorageVolume. This function takes a size to attempt to achieve and returns the size achieved. Space is taken from the input StoragePool. The desired settings of the Volume created are specified by Goal. If the requested size cannot be created, no action will be taken, the Return Value will be 0x1001 (Size and the output value, size will be set to the nearest possible size. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started to create the StorageVolume
DeleteStoragePool()	Uint32	[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [in, Description("Handle to pool to delete")] CIM_StoragePool ref Pool	Start a job to delete a StoragePool. The freed space is returned to the source StoragePool.If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started to delete the StoragePool
DeleteStorageVolume()	Uint32	[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [in, Description("Handle to volume to delete")] CIM_StorageVolume ref Volume	Start a job to delete a StoragePool. The freed space is returned to the source StoragePool.If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started to delete the StorageVolume
ModifyStoragePool()	Uint32	[Out, in(false), Description("Handle to job (may be null if job completed)")]	Start a job to modify a StoragePool. This function takes a size to attempt to achieve (+ or -) and returns the size

Property/ Method	Type	Qualifier or Parameter	Notes
		<p>CIM_StorageConfigurationJob ref Job,</p> <p>[in, Description ("The definition for the StorageCapabilities to be maintained by the modified StoragePool.")]</p> <p>CIM_StorageCapabilities ref CapabilitiesGoal,</p> <p>[in,out, Description("Size of pool")] uint64 size,</p> <p>[in, Description ("Array of strings containing representations of references to input pools")] string InPool[],</p> <p>[in, Description ("Array of strings containing representations of references to source extents")] string Extent[],</p> <p>[in, Description ("Handle to pool being modified")]</p> <p>CIM_StoragePool ref OutPool</p>	<p>achieved. Space is taken from either or both of input StoragePools and Extents or space is freed back to the source pool. Capabilities attributes that the Pool must support are passed in via the class instance specified by CapabilitiesGoal. If the requested size cannot be created, no action will be taken, the Return Value will be 0x1001 and the output value 'size' will be set to the nearest possible size. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started to modify the StoragePool</p>
ModifyStorageVolume()	Uint32	<p>[Out, in(false), Description("Handle to job (may be null if job completed)")]</p> <p>CIM_StorageConfigurationJob ref Job,</p> <p>[in, Description ("The definition for the StorageSetting to be maintained by the created StorageVolume.")]</p> <p>CIM_StorageSetting ref Goal,</p> <p>[in,out] uint64 Size,</p> <p>[in] CIM_StorageVolume ref Volume</p>	<p>Start a job to modify a StorageVolume. This function takes a size to attempt to achieve (+ or -) and returns the size achieved. Space is taken from the input StoragePool or space is freed back to the StoragePool. The desired settings of the modified Volume are specified by Goal. If the requested size cannot be created, no action will be taken, the Return Value will be 0x1001 and the output value 'size' will be set to the nearest possible size. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started to modify the StorageVolume</p>
CreateReplica()	Uint32	<p>[Out, in(false), Description("Handle to job (may be null if job completed)")]</p> <p>CIM_StorageConfigurationJob ref Job,</p> <p>[In, Required, Description("Source Storage Object")]</p>	<p>Start a job to create a new Storage Object which is a Replica of the Source storage object. Note that based on, CopyType this function can be used to: instantiate the Replica, and to create an ongoing association between Source and Replica. If 0 is returned, no StorageConfigurationJob instance is</p>

Property/ Method	Type	Qualifier or Parameter	Notes
		<p>CIM_LogicalElement ref SourceElement,</p> <p>[Out, in(false), Description("Handle to created replica.")] CIM_LogicalElement ref TargetElement,</p> <p>[In, Description("The definition for the StorageSetting to be maintained by the Target StorageVolume.")] CIM_StorageSetting ref TargetSettingGoal,</p> <p>[In, Description("New storage for the TargetElement, will be drawn from TargetPool if specified, otherwise allocation is implementation specific.")] CIM_StoragePool ref TargetPool,</p> <p>[In, Description("CopyType describes the type of copy that will be made Values are: Async: create and maintain an asynchronous copy of the source. Sync: create and maintain a synchronised copy of the source. UnSyncAssoc: create an unsynchronised copy and maintain an association to the source. UnSyncUnAssoc: create unassociated copy of the source element. "), Values { "Async", "Sync", "UnSyncAssoc", "UnSyncUnAssoc", "DMTF Reserved", "Vendor Specific"}, ValueMap { "0", "1", "2", "3", "..", "0x8000.." }] uint16 CopyType</p>	<p>created. If 0x1000 a StorageConfigurationJob will be started.</p>
DetachReplica ()	UInt32	<p>[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job,</p> <p>[In,Description("Target StorageExtent")]</p>	<p>Starts a job to 'forget' the synchronization association between two storage objects. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started.</p>

Property/ Method		Type	Qualifier or Parameter	Notes
			CIM_LogicalElement ref ReplicaElement	
FractureReplica()	Uint32		[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [In,Description("Target StorageExtent, LogicalFile, or FileSystem. ")] CIM_LogicalElement ref ReplicaElement	Start a job to suspend the synchronization between two storage objects. The association is remembered and typically changes are remembered to allow a fast resynchronization. This is typically used during a backup cycle to allow one of the objects to be copied while the other remains in production. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started.
ReSyncReplica()	Uint32		[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [In,Description("StorageSynchronized association describing association")] CIM_StorageSynchronized ref Synchronization	Start a job to re-establish the synchronization of a replica. If CopyJob is Sync or Async, this will negate the action of the StartFractureReplica method. If CopyJob is UnsyncAssoc, this will update the replica to reflect the latest contents of the source. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started.
RestoreFromReplica()	Uint32		[Out, in(false), Description("Handle to job (may be null if job completed)")] CIM_StorageConfigurationJob ref Job, [In,Description("StorageSynchronized association describing association")] CIM_StorageSynchronized ref Synchronization	Start a job to renew the contents of the original Storage Object from a replica. If 0 is returned, no StorageConfigurationJob instance is created. If 0x1000 a StorageConfigurationJob will be started.

Table 99: StorageConfigurationService Derivation

C.90 StorageExtent

(As defined by CIM)

StorageExtent describes the capabilities and management of the various media that exist to store data and allow data retrieval. This superclass could be used to represent the various components of RAID (Hardware or Software) or as a raw logical extent on top of physical media.

(As refined by Bluefin)

StorageExtent represents a (logically) contiguous unit of disk storage. StorageExtent is a superclass of StorageVolume. When used a concrete class, it generally means storage which is internal to the Disk Array and not available to external entities.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
SystemCreationClassName	string	MaxLen(256), Key	The scoping System's CreationClassName.
SystemName	string	MaxLen(256), Key	The scoping System's Name.
CreationClassName	string	MaxLen(256), Key	The name of the concrete subclass
DeviceID	string	MaxLen(64), Key	unique identifying information
PowerManagementSupported	boolean		
PowerManagementCapabilities	Int16[]		
Availability	Int16		primary availability and status of the Device
StatusInfo	Int16		
LastErrorCode	UInt32		captures the last error code reported
ErrorDescription	string		
ErrorCleared	boolean		error reported in LastErrorCode is now cleared
OtherIdentifyingInfo	String[]		captures additional data, beyond DeviceID information, that could be used to identify a LogicalDevice
PowerOnHours	UInt64		
TotalPowerOnHours	UInt64		
IdentifyingDescriptions	String[]		free-form strings providing explanations and details behind the entries in the OtherIdentifyingInfo
AdditionalAvailability	UInt16[]		
MaxQuiesceTime	UInt64		
<i>StorageExtent</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
DataOrganization	Uint16		"Other", "Unknown", "Fixed Block", "Variable Block", or "Count Key Data"
Purpose	String		
Access	Uint16		readable, writable, ...
ErrorMethodology	String		type of error detection and correction
BlockSize	Uint64		
NumberOfBlocks	Uint64		
ConsumableBlocks	Uint64		maximum number of blocks, of size BlockSize, which are available for consumption when layering StorageExtents using the BasedOn association
IsBasedOnUnderlyingRedundancy	boolean		
SequentialAccess	boolean		FALSE for disk storage

Table 100: StorageExtent Derivation

C.91 **StorageMediaLocation**

(As defined by CIM)

StorageMediaLocation is a PhysicalElement where PhysicalMedia may be placed. This class describes an entity that holds Media and is not just a 'place' (as is conveyed by the Location object). This class is typically used in the context of a StorageLibrary. Examples of StorageMediaLocations are MediaAccessDevices, InterLibraryPorts or 'slots' in a Library's panel.

Property/ Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>PhysicalElements</i>			
<i>PhysicalPackage</i>			
StorageMediaLocation			

Property/ Method	Type	Qualifier/Parameter	Description/Notes
LocationType	Uint16 enum	"Unknown", "Other", "Slot", "Magazine", "MediaAccessDevice", "InterLibrary Port", "Limited Access Port", "Door", "Shelf", "Vault"	
LocationCoordinates	string		
MediaTypesSupported	Uint16[] enum	ArrayType ("Indexed")	See MOF for list of media types
MediaSizesSupported	Real32[]	ArrayType ("Indexed"), Units ("Inches")	
MediaCapacity	Uint32		
TypeDescriptions	String[]	ArrayType ("Indexed")	

Table 101: **StorageMediaLocation** Derivation**C.92 StorageLibrary**

(As defined by CIM)

A **StorageLibrary** is a collection of **ManagedSystemElements** that operate together to provide cartridge library capabilities. This object serves as an aggregation point to group the following elements: **MediaTransferDevices**, a **LabelReader**, a library **Door**, **MediaAccessDevices**, and other Library components.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
ManagedElement			
InstanceName	string		
ManagedSystemElement			
OperationalStatus	uint16		
LogicalElement			
System			
CreationClassName	String	MaxLen(256), Key	Name of Class
Name	String	MaxLen(256), Key	
StorageLibrary			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
Capabilities	UInt16[]	"Unknown", "Other", "Automatic Cleaning", "Manual Operation", "Front Panel Lockable"	
Overfilled	boolean		
AuditNeeded	boolean		
AuditInProgress	boolean		
MaxAuditTime	UInt64	Units ("Seconds")	
Automated	boolean		
RoboticsEnabled	boolean		
EnableRobotics([IN] boolean Enable)	UInt32		

Table 102: StorageLibrary Derivation

C.93 StoragePool

A pool of Storage that is managed by a particular System. StoragePools may consist of component StoragePools or StorageExtents. StorageExtents/StoragePools that belong to the StoragePool have a Component relationship to the StoragePool. StorageExtents/StoragePools that are components of a pool have their available space aggregated into the pool. StoragePools and StorageVolumes may be created from StoragePools. This is indicated by the AllocatedFromStoragePool association StoragePool is weak to a system associated by SystemStoragePool.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
StoragePool			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
SystemCreationClass sName	String	MaxLen(256),Key	The scoping System's CreationClassName.
SystemName	String	key	The scoping System's Name.
CreationClassName	String	MaxLen(256),Key	A unique name in the context of the System that identifies this pool.
PoolStatus[]	Uint16	Building", "Ready", "De graded", "DMTF Reserved", "Vendor Specific (ValueMap)	Indication of current status of storage pool over and above that provided by Operational Status.
GetSupportedSizes()	Uint32	[in, Description ("Goal to qualify volume size by")] CIM_StorageSetting ref Goal, [out, in(false), Description ("List of support sizes for Volume and Pool creation/modification"), Units ("Bytes")] uint64 Sizes[]	For pools that support discrete sizes for volume or pool creation, this method can be used to retrieve a list of supported sizes. Note that different pool implementations may support either or both the GetSupportedSizes and GetSupportedSizeRanges at different times depending on Pool configuration. Also note that the advertized size may change after the call due to requests from other clients. If the pool currently only supports a range of sizes, then the return value will be set to 1

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
GetSupportedSizeRange()	Uint32	[in, Description ("Goal to qualify volume size by")]CIM_StorageSetting ref Goal, [out, in(false), Description ("The minimum size of a volume"),Units ("Bytes")]uint64 MinimumVolumeSize, [out, in(false), Description ("The minimum size of a volume"),Units ("Bytes")]uint64 MaximumVolumeSize, [out, in(false), Description ("A volume size must be a multiple of this value"),Units ("Bytes")]uint64 VolumeSizeDivisor	For pools that support a range of sizes for volume or pool creation, this method can be used to retrieve the supported range. Note that different pool implementations may support either or both the GetSupportedSizes and GetSupportedSizeRanges at different times depending on Pool configuration. Also note that the advertized size may change after the call due to requests from other clients. If the pool currently only supports discrete sizes, then the return value will be set to 1
TotalAvailableSpace	Uint64		Raw available space in pool.

Table 103: StoragePool Class Derivation

C.94 **StoragePoolComponent**

From MOF

StoragePoolComponent is a specialization of the Component association that establishes 'part of' relationships between a StoragePool and the StorageExtents of which it is composed.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Aggregation] Component			
[Aggregation] StoragePoolComponent			
Pool	REF	Override	StoragePool Reference
Component	REF	Override	Component Reference

Table 104: StoragePoolComponent Derivation

C.95 **StorageRedundancyGroup**

(As defined by CIM)

A class derived from RedundancyGroup containing mass storage-related redundancy information. StorageRedundancyGroups are used to protect user data. They are made up of one or more PhysicalExtents, or one or more AggregatePExtents. StorageRedundancyGroups may overlap. However, the underlying Extents within the overlap should not contain any check data.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
RedundancyGroup			
CreationClassName	string	MaxLen(256)	The name of the concrete subclass
Name	string	MaxLen(256),override	
RedundancyStatus	UInt16		
StorageRedundancyGroup			
TypeOfAlgorithm	UInt16		The TypeOfAlgorithm specifies the algorithm used for data redundancy and reconstruction.
StorageRedundancy	UInt16		provides additional information on the state of the RedundancyGroup, beyond the RedundancyStatus property. Information like "Reconfig In Progress" (value =1) or "Redundancy Disabled" can be specified using this property.
IsStriped	Boolean		
IsConcatenated	boolean		

Table 105: StorageRedundancyGroup Derivation

C.96 **StorageSetting**

StorageSetting is roughly equivalent to a Service Level Agreement (SLA) when associated by ElementSetting to a StorageVolume. StorageSetting is a Service Level Objective (SLO) when used in a StartCreateStorageVolume method. It defines a series of properties with Maximum and Minimum values that the object should stay between.

Property/Method	Type	Qualifier/Parameter	Description/Notes
<i>ManagedElement</i>			
Setting			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
VerifyOKToApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The VerifyOKToApplyToMSE method is used to verify that this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The ApplyToMSE method performs the actual application of the Setting to the referenced ManagedSystemElement.
VerifyOKToApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The VerifyOKToApplyToCollection method is used to verify that this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The ApplyToCollection method performs the application of the Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyToMSE method against each of the Elements aggregated by the Collection.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE,	The ApplyIncrementalChangeToMSE method performs the actual application of a subset of the properties in the Setting

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
		[IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	to the referenced ManagedSystemElement.
VerifyOKToApplyIncrementalChangeToCollection ()	UInt32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The VerifyOKToApplyIncrementalChangeToCollection method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyIncrementalChangeToCollection()	UInt32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The ApplyIncrementalChangeToCollection method performs the application of a subset of the properties in this Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyIncrementalChangeToMSE method against each of the Elements aggregated by the Collection.
StorageSetting			
SettingID	MaxLen(256)	Key	A unique ID for the instance.
NoSinglePointOfFailure	Bool		Desired value for No Single Point of Failure. Possible values are false = can't, true = can do.
DataRedundancyMax	UInt16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter is the desired maximum value. Possible values are 1 to n
DataRedundancyMin	UInt16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter is the desired minimum value. Possible values are 1 to n
SpindleRedundancyMax	uint16		Spindle redundancy describes how many disk spindles can fail without data loss

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
			including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the desired maximum value. Possible values are 0 to n
SpindleRedundancyMin	uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the desired minimum value. Possible values are 0 to n
DeltaReservation	uint16		Delta reservation is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations.

Table 106: StorageSetting Class Derivation

C.97 **StorageSettingWithHints**

This subclass of StorageSetting allows a client to specify 'hint's for optimization of the volume performance. The effect that these hints have will be implementation dependent.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Setting</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
VerifyOKToApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The VerifyOKToApplyToMSE method is used to verify that this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyToMSE()	Uint32	[IN]CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy	The ApplyToMSE method performs the actual application of the Setting to the referenced ManagedSystemElement.
VerifyOKToApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The VerifyOKToApplyToCollection method is used to verify that this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyToCollection()	Uint32	[IN]CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [OUT] string CanNotApply[]	The ApplyToCollection method performs the application of the Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyToMSE method against each of the Elements aggregated by the Collection.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
VerifyOKToApplyIncrementalChangeToMSE()	Uint32	[IN] CIM_ManagedSystemElement ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	The VerifyOKToApplyIncrementalChangeToMSE method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced ManagedSystemElement, at the given time or time interval.
ApplyIncrementalChangeTo	Uint32	[IN] CIM_ManagedSystemElement	The ApplyIncrementalChangeToMSE

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
MSE()		ent ref MSE, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[]	method performs the actual application of a subset of the properties in the Setting to the referenced ManagedSystemElement.
VerifyOKToApplyIncrementalChangeToCollection ()	Uint32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The VerifyOKToApplyIncrementalChangeToCollection method is used to verify that a subset of the properties in this Setting can be 'applied' to the referenced Collection of ManagedSystemElements, at the given time or time interval, without causing adverse effects to either the Collection itself or its surrounding environment.
ApplyIncrementalChangeToCollection()	Uint32	[IN] CIM_CollectionOfMSEs ref Collection, [IN] datetime TimeToApply, [IN] boolean ContinueOnError, [IN] datetime MustBeCompletedBy, [IN] string PropertiesToApply[], [OUT] string CanNotApply[]	The ApplyIncrementalChangeToCollection method performs the application of a subset of the properties in this Setting to the referenced Collection of ManagedSystemElements. The net effect is to execute the ApplyIncrementalChangeToMSE method against each of the Elements aggregated by the Collection.
<i>StorageSetting</i>			
SettingID	MaxLen(256)	Key	A unique ID for the instance.
NoSinglePointOfFailure	Bool		Desired value for No Single Point of Failure. Possible values are false = can't, true = can do.
DataRedundancyMax	Uint16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or more copies are maintained. This parameter is the desired maximum value. Possible values are 1 to n
DataRedundancyMin	Uint16		DataRedundancy describes the number of complete copies of data maintained. Examples would be RAID 5 where 1 copy is maintained and RAID 1 where 2 or

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
			more copies are maintained. This parameter is the desired minimum value. Possible values are 1 to n
SpindleRedundancyMax	uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the desired maximum value. Possible values are 0 to n
SpindleRedundancyMin	uint16		Spindle redundancy describes how many disk spindles can fail without data loss including, at most, one spare. Examples would be RAID5 with a Spindle Redundancy of 1, RAID6 with 2, RAID 6 with 2 spares would be 3. This parameter describe the desired minimum value. Possible values are 0 to n
DeltaReservation	uint16		Delta reservation is a number between 1 (1%) and a 100 (100%) that specifies how much space should reserved in a replica for caching changes. For a complete copy this would be 100%, but it can be lower in some implementations.
StorageSettingWithHints			
DataAvailabilityHint	Uint16		This hint is an indication from a client of the importance placed on data availability. Values are 0=Don't Care..10=Very Important
AccessRandomnessHint	Uint16		This hint is an indication from a client of the randomness of accesses. Values are 0=Totally Sequential..10=Totally Random
AccessDirectionHint	Uint16		This hint is an indication from a client of the direction of accesses. Values are 0=Totally Read..10=Totally Write
AccessSizeHint[]	Uint16[]		This hint is an indication from a client of the access sizes to optimize for. Several sizes can be specified.
AccessLatencyHint	Uint16		This hint is an indication from a client how important access latency is Values are 0=Don't Care .. 10=Very Important
AccessBandwidthHint	Uint16		This hint is an indication from a

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
dthWeight			client of bandwidth prioritization Values are 0=Don't Care..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..10=Very Important.
StorageEfficiencyHint	Uint16		This hint is an indication of the importance placed on storage efficiency by the client. Values are 0=Don't Care..10=Very Important. A StorageVolume provider might choose different RAID levels based on this hint

Table 107: StorageSettingWithHints Derivation

C.98 StorageVolume

(As defined by CIM)

A StorageVolume is an Extent that is presented to the Operating System (for example, by a hardware RAID cabinet), to a File System (for example, by a software volume manager) or to another entity. StorageVolumes do NOT participate in StorageRedundancy Groups. They are directly Realized in hardware or are the end result of assembling lower level Extents.

(As refined by Bluefin)

The StorageVolume class is used to represent storage that a disk array exports to a host system or to another SAN component.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
Caption	string	MaxLen (64)	Short (one line) description
Description	string		Longer description
<i>ManagedSystemElement</i>			
InstallDate	datetime		
Name	string	MaxLen (256)	
Status	string	MaxLen (10)	
<i>LogicalElement</i>			
<i>LogicalDevice</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
SystemCreationClassName	string	MaxLen(256) , Key	The scoping System's CreationClassName.
SystemName	string	MaxLen(256) , Key	The scoping System's Name.
CreationClassName	string	MaxLen(256) , Key	The name of the concrete subclass
DeviceID	string	MaxLen(64), Key	unique identifying information
PowerManagementSupported	boolean		
PowerManagementCapabilities	Int16[]		
Availability	Int16		
StatusInfo	Int16		
LastErrorCode	UInt32		
ErrorDescription	string		
ErrorCleared	boolean		
OtherIdentifyingInfo	String[]		
PowerOnHours	UInt64		
TotalPowerOnHours	UInt64		
IdentifyingDescriptions	String[]		
AdditionalAvailability	UInt16[]		
MaxQuiesceTime	UInt64		
StorageExtent			
DataOrganization	UInt16		
Purpose	String		
Access	UInt16		
ErrorMethodology	String		
BlockSize	UInt64		
NumberOfBlocks	UInt64		
ConsumableBlocks	UInt64		
IsBasedOnUnderlyingRedundancy	boolean		
SequentialAccess	boolean		
StorageVolume			

Table 108: StorageVolume Derivation

C.99 SystemDevice

(As defined by CIM)

LogicalDevices may be aggregated by a System. This relationship is made explicit by the SystemDevice association.

(As refined by Bluefin)

Provides the association from the disk array ComputerSystemDefinition (storage controller) to volumes and ports.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
[Aggregation] Component			
[Aggregation] SystemComponent			
[Aggregation] SystemDevice			
GroupComponent	REF	Override	System Reference
PartComponent	REF	Override	LogicalDevice Reference

Table 109: SystemDevice Derivation

C.100 TapeDrive

(As defined by CIM)

Capabilities and management of a TapeDrive, a subtype of MediaAccessDevice.

Property/ Method	Type	Qualifier or Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			
<i>LogicalDevice</i>			
<i>MediaAccessDevice</i>			
TapeDrive			
EOTWarningZoneSize	Uint32	Units ("Bytes")	
MaxPartitionCount	Uint32		
Padding	Uint32	Units ("Bytes")	
MaxRewindTime	Uint64	Units ("MilliSeconds")	

Table 110: TapeDrive Derivation

C.101 UnitAccess

The UnitAccess relationship indicates which Devices are exposed through the Target Controller to initiators.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
Dependency			
UnitAccess			
Antecedent	REF	Key	Controller reference
Dependent	REF	Key	LogicalDevice reference

Table 111: UnitAccess Association Derivation

C.102 Zone

(As defined by CIM)

A set of ZoneMembers or ZoneAliases that collectively specify a set of endpoints in a fabric that are allowed to participate together in the fabric.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Collection</i>			
<i>SystemSpecificCollection</i>			
SystemCreationClassName	string	Propagated,Key	REQUIRED
SystemName	string	Propagated,Key	REQUIRED
InstanceID	string	Key	REQUIRED
InstanceName	string		The Zone Name FCGS Zone.Name
ZoneType	uint16 (enum)		Default or Protocol, REQUIRED
ProtocolType	uint16 (enum)		FCP, VI, IP
ReadOnly	Boolean		
Active	Boolean		Indicates that this ZoneSet is active and cannot be changed. REQUIRED

Table 112: Zone Derivation

C.103 ZoneAlias

(As defined by CIM)

A set of ZoneMembers and is used as a management entity to collect subsets of ZoneMembers to be added to one or more Zones.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Collection</i>			
<i>SystemSpecificCollection</i>			
SystemCreationC lassName	string	Propagated,Key	REQUIRED
SystemName	string	Propagated, Key	REQUIRED
InstanceID	string	Key	REQUIRED
InstanceName	string		Zone Alias Name FCSW ZoneAlias.Name, REQUIRED
<i>ZoneAlias</i>			
Active	boolean		Indicates that this ZoneAlias is part of an active ZoneSet and cannot be changed. REQUIRED

Table 113: ZoneAlias Derivation

C.104 ZoneCapabilities

(As defined by CIM)

The capabilities of the zoning in the AdminDomain.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Capabilities</i>			
<i>SystemSpecificCapabilities</i>			
SystemCreationClassName	string	Propagated,Key	REQUIRED
SystemName	string	Propagated, Key	REQUIRED
InstanceID	string	Key	REQUIRED
InstanceName	string		REQUIRED
<i>ZoneCapabilities</i>			
ZoneNameMaxLen	uint16		
ZoneNameFormat	uint16 (enum)		
MaxNumZoneSets	uint32		
MaxNumZones	uint32		
MaxNumZoneMembers	uint32		
MaxNumZoneAliases	uint32		
MaxNumZonesPerZoneSet	uint32		

Table 114: ZoneCapabilities Derivation

C.105 ZoneMember

(As defined by CIM)

Is an association to fabric element that is used to define the participating end port in the Zone.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>MemberOfCollection</i>			
Collection	Collection	Key	Zone
Member	ManagedElement	Key	FCPort, ZoneAlias, or LogicalPortGroup
<i>ZoneMember</i>			
ZoneMemberType	uint16 (enum)		Other, Unknown, PortWWN, FCID, DomainPort FCGS ZoneMember.Identifier Type

Table 115: ZoneMember Derivation

C.106 ZoneService

(As defined by CIM)

The Service responsible for defining the zone enforcement for the fabric. The ZoneService is Hosted on an AdminDomain and defines the containment and scope of the zoning entities.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>ManagedSystemElement</i>			
<i>LogicalElement</i>			

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>Service</i>			
Name	string	Key	
<i>ZoneService</i>			
OperationalStatus	uint16 (enum)		REQUIRED
CreateZone()	uint16		
DeleteZone()	uint16		
CreateZoneSet()	uint16		
DeleteZoneSet()	uint16		
CreateZoneAlias()	uint16		
DeleteZoneAlias()	uint16		
ActivateZoneSet()	uint16		
DeactivateZoneSet()	uint16		

Table 116: ZoneService Derivation

C.107 ZoneSet

(As defined by CIM)

A set of zone definitions for a fabric containing one or more Zones which when activated define the zone enforcement for a Fabric.

Property/ Method	Type	Qualifier/ Parameter	Description/Notes
<i>ManagedElement</i>			
<i>Collection</i>			
<i>SystemSpecificCollection</i>			
SystemCreationClassName	string	Propagated, Key	REQUIRED
SystemName	string	Propagated, Key	REQUIRED
InstanceID	string	Key	REQUIRED
InstanceName	string		The ZoneSet name. FCGS ZoneSet.Name, REQUIRED
<i>ZoneSet</i>			
Active	boolean		Indicates that this ZoneSet is active and cannot be changed.

Table 117: ZoneSet Derivation

Appendix D: Futures

D.1 HBA LUN masking and persistent binding

This section should be refined and expanded to assure that the specification is properly integrated with other standards for LUN masking and persistent binding (i.e., HBA API).

D.2 Managed Hub Section

The current Bluefin specification doesn't address managed hubs as a possible SAN component. If they continue to be of interest, the specification will need to be expanded to address any concerns particular to managed hubs.

D.3 IPSEC

The current Bluefin specification doesn't address IPSEC. As part of the specification refinement and completion, it will need to be expanded to appropriately address IP security and authentication.

D.4 Multi-Path Modeling

The current specification doesn't include support for multipathing within its array profile. The profile and its models should be expanded to provide appropriate support and infrastructure.

D.5 Provider Modeling

Provider Modeling is emerging DMTF work that needs to be monitored.

D.6 Requirements Highlighting

The current specification is not structured to make it easy for an implementer to identify and enumerate the set of implementation and compliance requirements that to which he is subject. A future revision of the Bluefin specification should create and include a standard format for compliance/requirements statements throughout the document. In addition, these requirements should be fully integrated with any compliance tests that are developed as part of the deployment of a formal SAN management standard.

D.7 Non-Fibre Fabrics

In future versions of this specification, it is intended that the fabric model and durable names for ports will be extended to cover other types of connectivity, such as InfiniBand, IP networks, etc.

Although the current fabric model is specific to Fibre Channel, a best-efforts approach was taken in defining it to allow for future extensions to include additional types of connectivity.

When the fabric model is extended for a new type of connectivity, it will be necessary to define durable names for the ports on the new type of connectivity. For instance, the durable names for ports on an IP network might be MAC addresses. It is expected that durable names for ports will be connectivity specific, and may be different for each different type of connectivity.

One important thing should be noted when extending the model to handle additional types of connectivity. In the current model, durable names are not defined for `SCSIController` objects, since there is always a one-to-one relationship between `SCSIController` objects and `FCPort` objects in Fibre Channel. Since `FCPort` objects have durable names, `SCSIController` object instances can always be unambiguously identified using the association to the corresponding `FCPort` object instance.

Note that this one-to-one relationship may not hold for other types of connectivity. If the relationship between `SCSIController` objects and port objects can be one-to-many (or vice versa), then a different method of uniquely identifying `SCSIController` object instances must be defined.

D.8 Compliance Notification

A method is needed to allow a provider to inform clients that it complies to Bluefin's indications profiles. Emerging work from the DMTF Interop Workgroup relates to this problem. Rather than offering a competing approach, Bluefin will re-evaluate this work in the near future. The Interop work is described in the "Modeling Profiles" section.

D.9 Cascaded Agents

The following wording was developed as the subcommittee considered the ramifications of a multi-level agent hierarchy, in which a given agent could provide a management interface to higher-level devices while simultaneously relying on the management interface provided by agents "below" it. While this was felt to be an important and likely scenario, time did not allow it to be fully developed.

1. Description

As discussed in section 2.3.2, components may implement both agent and client behavior. One or more such components can be configured into a chain or cascade terminating in non-cascading agents. Such configurations are not supported in the first release of Bluefin. This section outlines an approach for adding support; compliance statements of "must" and "should" are not intended to apply to the first release of Bluefin.

Cascaded agents can add value in several ways.

- Provide a single point of control proxy for multiple device agents.
- Implement storage or management services, such as virtualization, using device agents.
- Improve scalability through aggregation and consolidation.
- Distribute a set of management functions across several computer systems.
- However, cascaded agents need some constraints and support to avoid several problems.
- Redundant control over a given device
- Ambiguous or unstated control over a given device
- Cyclic dependencies among agents and other improper configurations

2. Client-to-Agent Dependencies

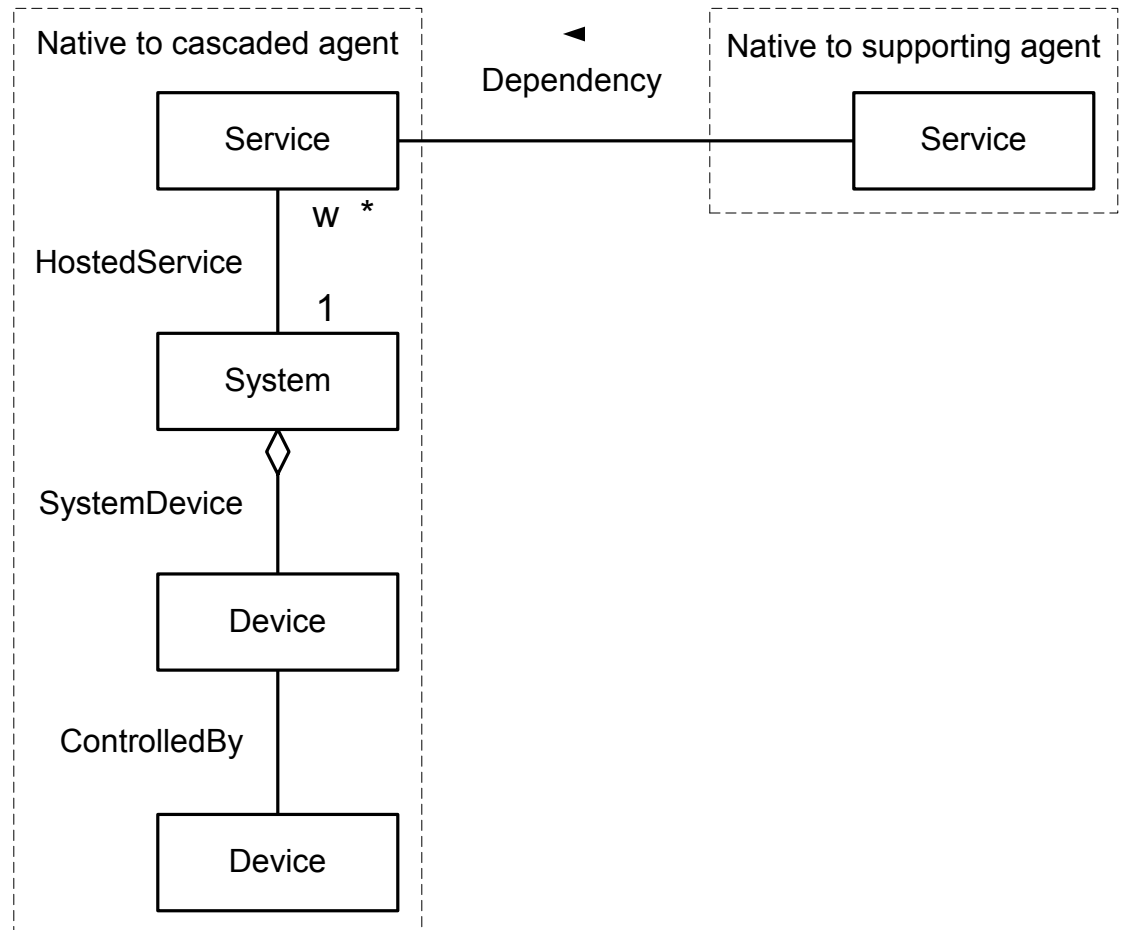
All Bluefin agents are required to implement the interop schema (see section 7.4). In particular, each agent creates an ObjectManager instance to represent its own agent role. A cascaded agent must explicitly represent its client dependency on other Bluefin agents by creating Dependency instances between its own ObjectManager and the ObjectManager instances created by the agents it depends on. The dependent end of the Dependency association is the client's local ObjectManager and the antecedent end is the server's remote ObjectManager. The client (cascaded agent) does not create local proxies for the server ObjectManager instances.

A cascaded agent must model the Bluefin agents it depends on, and this client-to-agent link can be relied on by clients. Optionally, a cascaded agent may represent additional, more specific service dependencies. For example, a virtualization system's `StorageAccessService` might depend on the `StorageAccessServices` of one or more arrays. The agent for the virtualization system is permitted to explicitly model these dependencies.

If the cascaded agent models *any* additional Service-to-Service dependencies, a client may infer that this is the complete and exclusive set of such dependencies. In particular, the absence of a `Dependency` between two specific Services is significant. If the cascaded agent models *only* `ObjectManager`-to-`ObjectManager` dependencies, a client must assume the cascaded agent depends on the server agents in unrestricted ways.

Cascading relationships reported through `Dependency` are not exclusive of any other dependencies. Presence of client-to-agent links does not imply anything about unique or comprehensive control over the agent. For example, a virtualization system may depend on several disk storage systems but we can not conclude the storage systems are totally under the control of the virtualization system.

To traverse one step in an agent cascade, start with the client service of interest and follow `Dependency` in the reverse direction (client is the dependent end of the association). If there are no such dependencies, follow `HostedService` to `System`, then follow `HostedService` to an `ObjectManager`. From `ObjectManager` follow `Dependency` to other agent `ObjectManagers`. Because `Dependency` is a generic association, it may be used for purposes other than showing cascading relationships. While traversing agent cascades, users should check for appropriate Service subclasses and ignore objects of other classes.



3. Device Multiple “Ownership” Not Supported

One crucial feature of our profiles is that some objects are modeled by multiple agents. For example, the `StorageVolume` exported by a disk storage array (and reported by its agent) is the same volume imported by a host or virtualization system (and reported by its agent as well) and has the same durable name.

However, use of this feature must be carefully controlled. In the example just given, the storage array uniquely “owns” the volume, while the host only acknowledges it. To manipulate the properties of the device in the array, a client must interact with the array’s agent.

Definition: A `Service` “owns” a `Device` if it is a `HostedService` on a `System` with a `SystemDevice` relation to the device, or if the device is `ControlledBy` such a system device. In other words, we can construct the sequence of associations shown vertically in the left side of the figure above.

We presume that a service controls only the devices it owns, either through service extrinsic methods or CIM intrinsics (such as `set attribute`). Many agents may enumerate a given device and expose its attributes and associations with other objects, but control or modification of the device is provided by owning services.

Bluefin does not support ownership by competing services. It is OK for a device to have no owning service, or multiple distinct unrelated owning services, but not two services of classes related by inheritance. Classes with a common ancestor are (necessarily) OK, but not one class that is an ancestor of the other. The intent is to provide a unique service (and agent) to perform any given control action for a device. Bluefin does not prevent or prohibit agent cascades from implementing ambiguous or redundant control, however the behavior of such a configuration is not defined.

This is primarily an issue for cascaded agents that perform simple aggregation or single point of control proxy services. If both the cascaded agent and the agents it depends on “own” the same resource, clients may choose to work on the same resource through different agents. This quickly leads to fatal problems like deadlock. Therefore, Bluefin does not support this situation. Either the non-cascading agents must be hidden from clients of the cascaded agent, or the cascaded agent can not host services in common with the non-cascading agents.

4. Locking Considerations

Some additional rules are required for lock management (clause 6) in the presence of cascaded agents. Without these restrictions, the likelihood of deadlock or persistent lock request failure increases substantially.

The Dependency associations between agents form a directed graph which generally should have no cycles. This DAG has richer information about client-agent interactions than the set of LMGroups and can be used to control the risk of deadlocks. For the purposes of this section, “up” and “above” mean the toward-client direction in the graph, while “down” and “below” mean the toward-agent direction.

Resources controlled by different levels of an agent cascade (e.g., a virtualization system above a pair of storage arrays) may be apparently independent but actually dependent. To lock and operate on a resource at one level may require locks and operations on resources at levels below. These dependencies are not apparent to the client making the initial request.

Use well-defined, common LMGroups

Bluefin lock management clients are supposed to identify all the resources needed for a given operation and required to acquire the corresponding locks before manipulating any of the resources. To acquire multiple locks in a single lock request, the client and lock management agents must belong to a common LMGroup.

In principle, a different LMGroup could be used for each complex operation by a lock management client. It is strongly recommended that a cascaded agent and *all* agents below it belong to a common LMGroup and that this LMGroup be used for *all* locking operations by the cascaded agent (as a lock management client). In any event, it is critical for cascaded agents to obey the requirement for acquiring all locks before manipulating any resources (sections 6.9, 6.13), preferably by issuing a single LockRequest.

A cascaded agent may belong to multiple LMGroups. In particular, it may use LMGroup ONE as a Bluefin lock management client and LMGroup TWO as a Bluefin lock management agent. The resources it requests from agents below will be locked in group ONE, and the resources it exposes to clients above will be locked in group TWO. This will be especially common for virtualization systems.

Grant locks bottom-up

To avoid deadlock and minimize the risk of persistent lock refusal, resources must be locked bottom-up. That is, a client lock request from above can not be granted until all implied lock requests to lower levels have been granted. Because clients do not know what lower lock requests may be required, a cascaded agent must hold a client's lock request and initiate requests to agents below. Figure X illustrates this behavior. The cascaded agent in this figure corresponds to the LM Agent 1 of Figure 65.

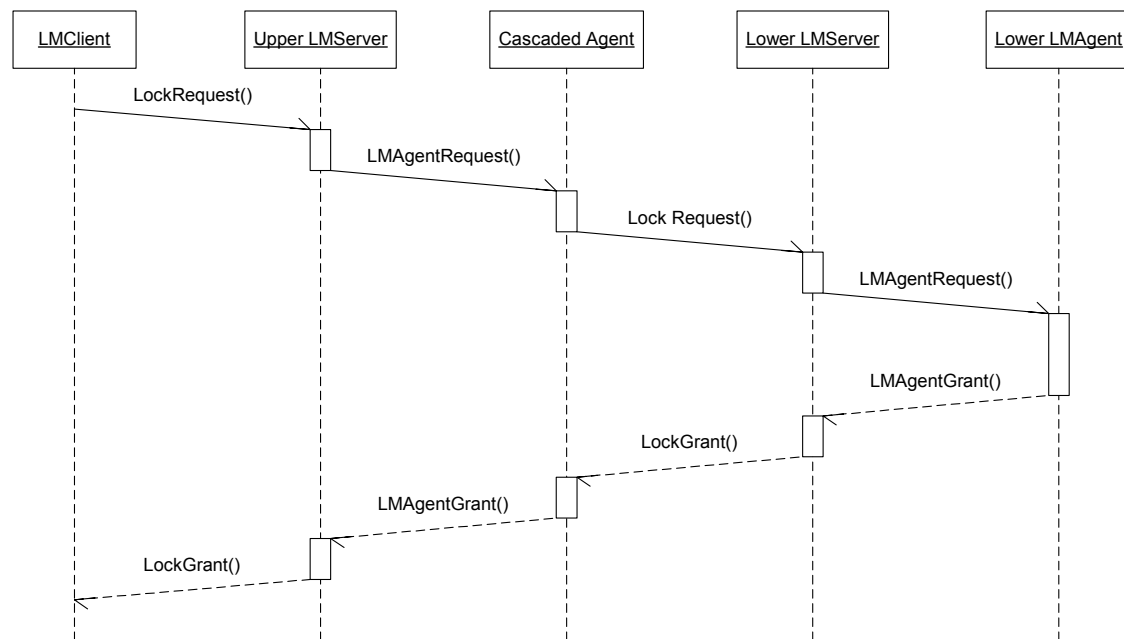


Figure X: Nested Locking with Bottom-up Grants

As noted in C.X.4.1, the cascaded agent may belong to different LMGroups (and use different lock management servers) in its roles as agent (above) and client (below). This is illustrated in Figure X by Upper LMServer and Lower LMServer. Because the use of different LMGroups is not mandatory, these may in fact be the same lock management server.

A lock management AgentRequest contains a specific list of CIMObjectPath's identifying the resources to be locked, and optionally a list of methods and properties to be manipulated on these resources. An empty or missing list implies that all methods and properties may be manipulated. Therefore, a cascaded agent can predict what resources below it may have to be locked. It must issue lock requests for all such resources and receive grants before granting the request from above.

The prediction of resources below must be inclusive of all necessary resources. It is expressly prohibited for a cascaded agent to defer a lock request for a resource below until a subsequent client operation “ensures” that particular resource is needed.

A cascaded agent may need to lock resources below it, even when carrying out an unlocked operation for a client. This is a natural consequence of virtualization and abstraction; the resources below and related locking requirements are hidden from the client above. Figure Y shows a sequence diagram for implied locking. OperationRequest and OperationResponse are implemented by the cascaded agent in its role as a Bluefin agent (clause 7), rather than lock management agent (clause 6). For example, these may be CIM-XML intrinsic operations.

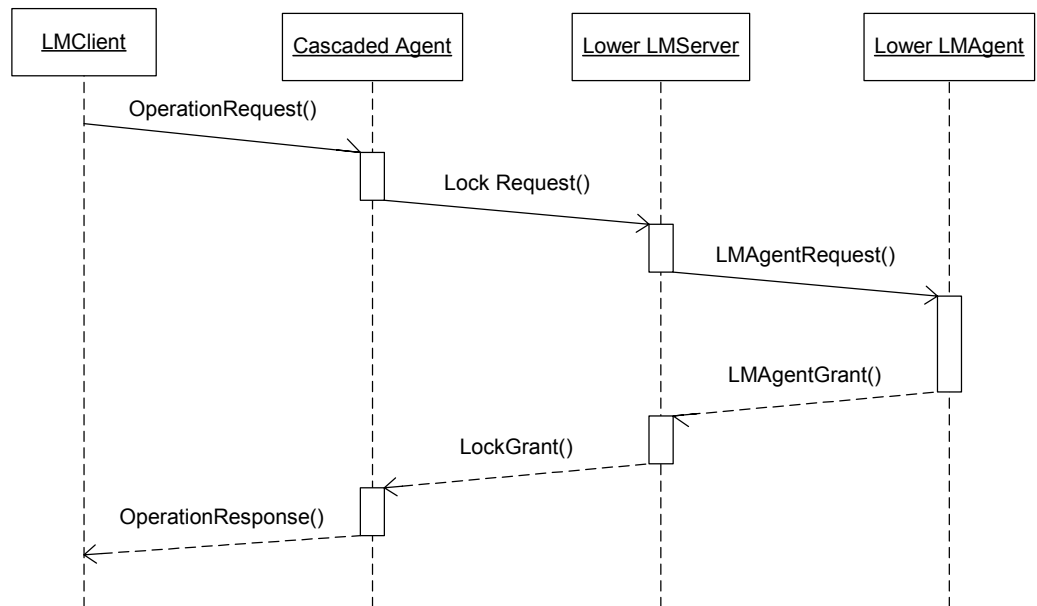


Figure Y: Implied Nested Locking

Lease durations

A cascaded agent, like all lock management agents, must return a TTL value when granting a lock management client's request. Several considerations apply when choosing a legal and appropriate TTL value.

A cascaded agent can not promise to its clients more than it has been promised by agents below. The TTL it returns in an AgentResponse MUST NOT exceed the minimum TTL granted from below. If the cascaded agent makes a single AgentRequest, the lock management server will compute the minimum for the corresponding AgentResponse. If the cascaded agent makes multiple AgentRequests, it MUST determine the minimum TTL of the multiple AgentResponses, taking into consideration the time required to receive each response. This is the same function required of lock management servers.

A cascaded agent SHOULD attempt to account for the time required to generate lock requests and collect lock grants, by requesting somewhat longer leases from agents below. This is most important for lock and lease renewal requests with short LeaseDurations, as the original requested duration might be consumed by nested request-grant overhead. The objective at each level in the cascade is to satisfy the immediate client with its requested LeaseDuration; it is not necessary or desirable for a cascaded agent to attempt to compensate for multiple levels of cascading.

In general, the amount of LeaseDuration compensation for a given cascaded agent must be determined heuristically. Simple implementations may hardcode a value like 50ms. More sophisticated implementations can monitor actual performance and adjust their compensation dynamically.

Minimize implied or hidden locking

The requirements that all locks be acquired at one time and bottom-up have an important implication. A cascaded agent should not initiate a lock request downward during the execution of a locked operation (equivalently, an operation on a locked resource). The cascaded agent **MUST** initiate lock requests only during the execution of explicit lock requests (as shown in Figure X) or during unlocked operations from above (as shown in Figure Y).

An end client never performs a locked operation in this sense, therefore it **MAY** issue lock requests at any time.

There are two general situations. First, a cascaded agent intends to add to a set of previously locked resources while servicing a client request. This violates the lock-at-once and lock-bottom-up requirements. Strict adherence to the deadlock management rules of section 6.9 will prevent a catastrophic failure, but wasted locking and time is likely with no guarantee of eventual completion of the client request due to persistent lock rejection and lack of forward progress.

Second, a cascaded agent intends to lock a new set of resources independent of any resources currently locked. This might be in support of an unlocked client operation (Figure Y) or due to the cascaded agent's own autonomous operations (i.e., as an end client). If the resources are truly independent of all others currently locked, the LockRequest can be granted. However, a client at a given level of a cascade can only know the dependencies established by itself. Agents below generally hide resource dependencies. To minimize failure of LockRequest's and OperationRequest's due to hidden dependencies, locks should be explicitly acquired as early and as high in a cascade as possible.

5. End Client Considerations

Discover all the services available for a given device

Starting with the device, follow **ControlledBy** (if appropriate) to a system device, then **SystemDevice** to a **System**, then **HostedService** to an enumeration of potentially useful owning services. If no **System** is found, the device is in the agent's DMZ and the agent offers no relevant services that own the device; other agents must be examined for available services.

There will not always be cascaded agent relationships between interacting systems. For example, a virtualization system may import volumes from a storage array without being a Bluefin client of the array. Therefore, a client can not always find the service available for a device by starting at an arbitrary agent and following association links. A comprehensive list of available services *can* be constructed by brute force, repeating this process with all agents discovered through SLP.

Determine the scope of an agent's services

An end client can determine the set of all devices that might be affected by a given agent by following the client-to-agent dependencies together with HostedService to obtain a set of Systems. SystemDevices of these systems and devices ControlledBy these system devices might be affected by the given agent.

This is the maximum possible scope known through the object model. There is no guarantee and no default inference about how much of the scope can *actually* be affected by the agent. The agent may also be able to affect devices through proprietary mechanisms not reported through the object model.

Verify an agent is using the proper set of device agents

An end client can follow the client-to-agent dependencies to obtain a list of Systems and/or ObjectManagers. This list can be compared with a list of expected agents obtained from other sources.

Discover how virtualization systems depend on one another

For any given virtualization system, an end client can follow the client-to-agent links to discover these dependencies explicitly. By starting with an enumeration of virtualization systems, the dependency graph for the entire SAN can be constructed.

Find the appropriate agent to manage a given device

An end client can begin by discovering all the services available for the device, and using only the specific service of interest (e.g., StorageAccessService). If the intent is to manage a device directly, there should be no more than one relevant owning Service. If the intent is to manage the device indirectly, the end client may have to apply some external policy for choosing among multiple agents.

Ambiguous ownership

Multiple agents may claim ownership of a common resource, regardless of Bluefin's lack of support for the situation. For example, several hosts may have access to a JBOD and model the disks in the JBOD as their own system devices. This modeling ambiguity reflects the actual ambiguity about which system has control over which disk. The problem has been referred to as "I claim this disk in the name of Cincinnati!"

Bluefin does not currently have a solution for this situation. We recommend against allowing it to arise.

6. Agent Considerations

Agent addition and removal

As a cascaded agent discovers relevant agent dependencies, through SLP discovery or other processes, it must create Dependency instances between its local Service objects and the relevant Services on the other agent(s). If the other agent has failed or is otherwise unavailable, creation of these Dependency instances may have to be deferred until the relevant Service and System keys of the other agent can be obtained.

As noted above, the dependency between ObjectManager services representing the agent(s) themselves must be represented, and more specific dependencies among other services (e.g., StorageAccessService) should be represented.

Avoidance of cyclic dependencies

A cascaded agent, or indeed any client, can create a dependency graph of all its direct and indirect dependencies. If the agent ever appears as an agent in its own dependency graph, a cycle has been detected.

Bluefin does not specify what actions the agent should take in response to a detected cycle, but the options include generating notifications, shutting down operation, and removing sufficient immediate dependencies to break the cycle.

Indication management

A high-quality cascaded agent will manage indications in a useful way. For example, a virtualization system should transform an incoming indication of disk failure into outgoing indications of failure or reduced availability for any affected exported volumes.

In the agent-to-client direction, this may not require any additional effort. Clients subscribe to whatever is of interest to themselves, and the cascaded agent updates its internal state (perhaps triggering indications to its own clients) in response to notifications from server agents.

However, new subscriptions may require cascading in the client-to-agent direction. If the virtualization system used in this example was subscribing to disk failure indications by default, the arrival of a client subscription for volume status or availability should cause the cascaded agent to issue new subscription requests to the server agents below it.

7. Additional Issues

The following features have not been sufficiently developed at this time.

Providers

The CIM WBEMService class hierarchy should be incorporated into the mandatory cascaded agent dependencies in a well-defined and interoperable way. This will capture dependencies of CIM providers on Bluefin agents below, which is more precise than saying an entire ObjectManager has such dependencies.

It is appropriate for a self-contained agent to model its unqualified dependency on other agents. However, an ObjectManager may contain many providers, each responsible for a different profile or namespace. Those providers which depend on Bluefin agents generally will depend on completely independent sets of agents. We want to capture this level of detail, especially to verify that the ObjectManager and its providers have been configured properly (see consideration C.X.5.3).

Agent-to-client links

The Dependency relation described here is strictly client-to-agent. Some benefits accrue from explicit representation of corresponding links in the opposite direction, but there are also several technical problems in maintenance of these links which we declined to solve at this time.

Given client-to-agent links, the agent-to-client links can be inferred through a brute-force examination of all agents of interest.

Local representation of remote Service and System objects

The Dependency relation described here relates local Service instances to instances implemented by other agents. If these other agents are (temporarily) unavailable, we can not express the continuing relationship robustly. Adding Service (and System because Service is weak on System) instances to the local agent as mirrors or surrogates for the remote agent's instances would allow a robust solution, but again there are some technical problems we declined to solve.

Current CIMOM implementations may autonomously purge associations with stale references to remote objects. Remote Service instances can become "stale" transiently due to agent initialization, and therefore may be removed by a CIMOM even when that is inappropriate. The agent that created the association receives no indication that the association has been removed, and must monitor its Dependencies and restore them as needed. The use of local proxy instances for remote services and systems gives more control to the agent.

7.1. Glossary additions

DMZ Demilitarized Zone. Bluefin jargon for the set of object instances created by an agent as proxies for objects owned by other agents.

DAG Directed Acyclic Graph. Standard terminology for a data structure with directed paths between elements without cycles anywhere in the structure.

D.10 Durable ID Formats

The format of properties like durable IDs (e.g. Fibre Channel world-wide name) must be precisely specified in future revisions of this specification to insure seamless interoperability between multi-vendor implementations.