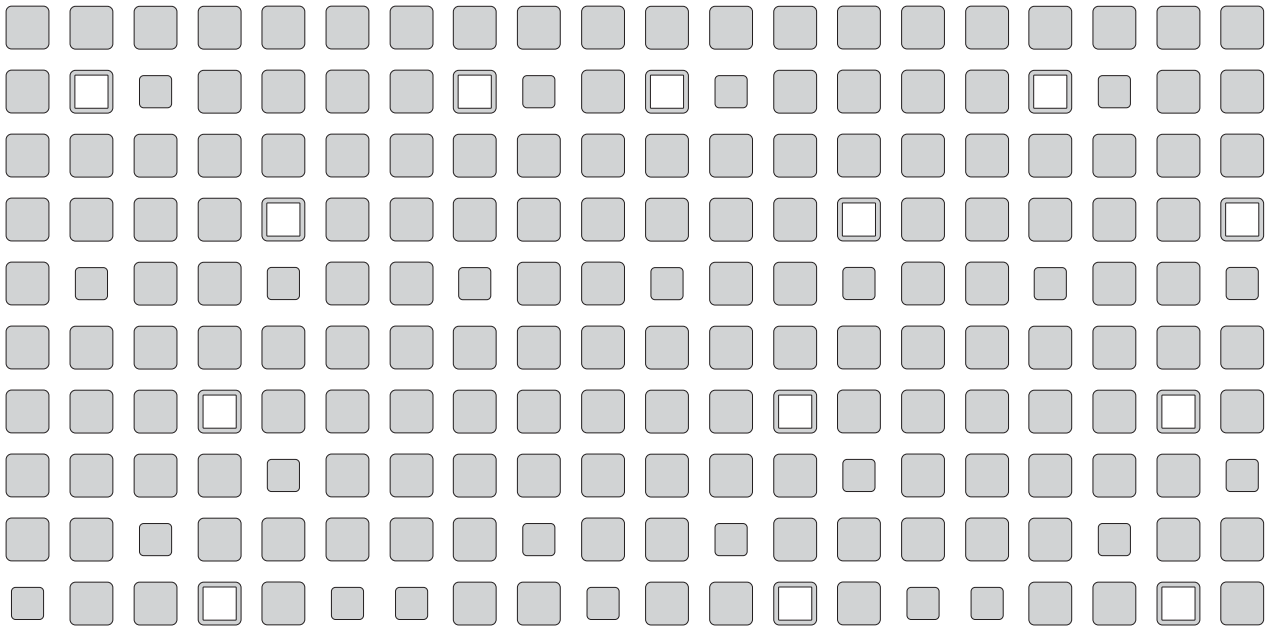


VERSION 2.0

VMware CIM SDK

Programming Guide



VMware, Inc.

3145 Porter Drive
Palo Alto, CA 94304
www.vmware.com

Please note that you will always find the most up-to-date technical documentation on our Web site at <http://www.vmware.com/support/>.

The VMware Web site also provides the latest product updates.

Copyright © 1998-2006 VMware, Inc. All rights reserved. Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022, 6,961,941, 6,961,806, and 6,944,699; patents pending. VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies. Revision 20060522 Version 2.0 Item: SDK-ENG-Q206-116

Table of Contents

Introducing the VMware CIM SDK	5
What's New in This Release	6
Recommended Documentation	7
CIM SDK Document Set	7
Technical Support Resources	7
Using This Manual	8
Intended Audience	8
Content By Chapter	8
Suggested Reading Approach	9
Working with the CIM SDK Development Environment	11
Overview of VMware CIM SDK Components	12
CIM SDK Files	12
CIM SDK Processes	12
Starting the Pegasus CIMOM	14
Setting Up Your Development Environment	15
Using the ESX Server Firewall	15
Connecting Your Client to the CIMOM	16
VMware CIM SDK Schema	17
Relationship to Industry Standards	19
DMTF CIM Schema	19
SMI-S Profile	20
Introduction to the VMware CIM Schema	27
Types of Clients	27
Conventions Used in This Document	27
Conventions Used in Illustrations	28
Traversing Associations	29
Sample System Environments	31
Virtual Machine with Single Virtual Disk on Local VMFS Storage	31
Virtual Machine with Virtual Disk and Snapshot on Local Storage	33
Virtual Machine with Single Virtual Disk on a LUN in a Storage Array	36
Virtual Machine Using Raw Device Mapping in a Storage Array	39
Two Virtual Machines on Different Servers Accessing Shared LUN	42
Multipath SAN Environment with Two Storage Arrays	47

Using the CIM SDK Schema	51
Virtual Machines on a Host	52
Virtual Host Bus Adapters on a Virtual Machine	53
Virtual Disks Connected to a Virtual Host Bus Adapter	55
Virtual Disks on a VMFS	58
VMFS Volumes Sharing a Directly Attached Disk	59
VMFS Volumes Sharing a Fibre Channel LUN	64
VMFS Spanning Two LUNs	66
Raw Device Mapping	67
System Devices of a Host	69
Sample Code	73
Connecting to the Pegasus CIMOM	75
Retrieving Information about an ESX Server Host	76
Enumerating StorageExtent (LUN) Objects Starting from the ESXComputerSystem	78
Listing Virtual Storage Available to Virtual Machines	81
Enumerating Virtual Machines Starting from StoragePool (VMFS) Objects	85
	88
Glossary	89
Terms and Conditions	93
VMware® Software Developer Kit (SDK) Agreement	93
Revision History	97
Index	99

1

CHAPTER

Introducing the VMware CIM SDK

The VMware CIM SDK provides a CIM-compliant object model for virtual machines and their related storage devices. The VMware model supports the standard SMI-S schema. This allows enterprise storage management developers to easily adapt existing CIM-compliant code to explore virtual machine resources and incorporate them into their management applications.

With the VMware CIM SDK, developers can:

- Explore the virtual machines on the ESX Server machine and view their storage resources using any CIM client.
- Examine virtual machine storage allocation to determine if availability and utilization policies are being satisfied.
- Examine the physical storage allocated to a virtual machine.
- Verify the operational status of virtual machine storage, including all storage devices involved in supplying storage to virtual machines.

Note: Some parts of the CIM SDK schema are experimental. The interface may change in future releases to align more closely with evolving standards. For more detail on this topic, refer to Exceptions to Standard Schema on page 23.

What's New in This Release

This release of the CIM SDK features:

- Support for ESX Server 3.0 and CIM 2.9.0
- Compliance with the SMI-S In-band Virtualizer profile developed by the Storage Network Industry Association
- Visibility for Fibre Channel, parallel SCSI, iSCSI, or block device storage for virtual machine files
- Support for indications
- Code samples in C++, Java, and Python

Recommended Documentation

CIM SDK Document Set

The following documents are provided with the CIM SDK:

- *VMware CIM SDK Reference*
The reference guide is a set of HTML files describing in detail the objects and associations used by the VMware CIM schema.
- *VMware CIM SDK Programming Guide*
This book describes how the CIM object model applies to the virtual machine storage environment. Included in this book are code samples and data diagrams for typical storage environments.

Technical Support Resources

The VMware CIM SDK documentation assumes familiarity with VMware ESX Server, CIM applications, and the SNIA (Storage Networks Industry Association) SMI-S (Storage Management Initiative Specification) profiles in particular. Refer to the following Web sites for additional information:

- VMware ESX Server — http://www.vmware.com/products/server/esx_features.html
- Distributed Management Task Force — <http://www.dmtf.org>
- Common Information Model — <http://www.dmtf.org/standards/cim/>
- OpenPegasus — <http://www.openpegasus.org>
- SNIA — <http://www.snia.org/home>
- SMI-S — http://www.snia.org/smi/tech_activities/smi_spec_pr/spec/
- WBEM Services — <http://wbemservices.sourceforge.net/>

Using This Manual

The purpose of this programming guide is to familiarize you with the logical structure of the VMware CIM SDK, the VMware extension schema, and the components needed to explore your ESX Server machines using CIM-enabled clients.

The *VMware CIM SDK Programming Guide* contains:

- Information to help you set up a development environment.
- Data models of the VMware CIM extension schema.
- Code samples to demonstrate typical operations.

This guide is meant to be used in conjunction with the HTML-based *VMware CIM SDK Reference*, which provides greater detail for each class in the VMware CIM extension schema.

Intended Audience

This programming guide is written for programmers who are familiar with:

- The practical application of CIM concepts and principles.
- The SMI-S schema profile.
- Developing system administration and system monitoring applications.
- Operating and managing ESX Server.

Refer to Technical Support Resources on page 7 for references to useful background information.

Content By Chapter

This *VMware CIM SDK Programming Guide* is organized into the following chapters:

1. Introducing the VMware CIM SDK on page 5
Chapter 1 is a brief overview of the CIM SDK and an introduction to the *Programming Guide*.
2. Working with the CIM SDK Development Environment on page 11
Chapter 2 provides information about the CIM client development environment. The chapter describes how to configure and start the Pegasus CIM object manager (CIMOM) installed with ESX Server, and how to prepare your environment for application development.
3. VMware CIM SDK Schema on page 17
Chapter 3 gives an in-depth description of the VMware CIM SDK extension schema. The chapter describes how to work with the CIM data model that represents ESX Server, virtual machines, and their storage resources.
4. Sample Code on page 73

Chapter 4 presents sample code for typical operations. The chapter presents code samples that demonstrate basic CIM client functions, such as accessing the CIMOM and querying for data. These can be used as building blocks for working CIM clients.

5. Glossary on page 89

Chapter 5 is a glossary of terms related to the CIM SDK.

Suggested Reading Approach

If this is your first time reading this manual, here is a suggested approach to understanding the content and applying it to your application:

1. Read the introductory chapter to familiarize yourself with the assumptions and structure of the manual.
2. Skim the schema introductory material at the beginning of chapter 3 (Relationship to Industry Standards on page 19) for the context in which the VMware schema applies.
3. Read the section (Introduction to the VMware CIM Schema on page 27) explaining the conventions used in the manual.
4. Browse the system environments in chapter 3 (Sample System Environments on page 31) to select one or more that are similar to your ESX Server environment.
5. Select and study one or more of the object relationships in chapter 3 (Using the CIM SDK Schema on page 51) that apply to your ESX Server environment.
6. Read the setup and development instructions in chapter 2 (Working with the CIM SDK Development Environment on page 11).
7. Browse the sample code in chapter 4 (Sample Code on page 73) to choose an approach for your client code.
8. Refer to glossary terms (Glossary on page 89) as needed.

Working with the CIM SDK Development Environment

This chapter describes the VMware CIM SDK development environment. The chapter includes tips on configuring your local development environment and ESX Server machine for CIM client application development.

This chapter contains the following sections:

- Overview of VMware CIM SDK Components on page 12 briefly discusses the components of the CIM SDK.
- Starting the Pegasus CIMOM on page 14 describes how to start the Pegasus CIMOM on the ESX Server machine.
- Setting Up Your Development Environment on page 15 discusses the basics required to begin application development using a CIM-compliant library, the installed CIMOM, and VMware Extended Schema.

For more detail on using the object model, refer to VMware CIM SDK Schema on page 17.

For code samples for basic operations, refer to Sample Code on page 73.

Overview of VMware CIM SDK Components

CIM SDK Files

The CIM SDK is composed of the following:

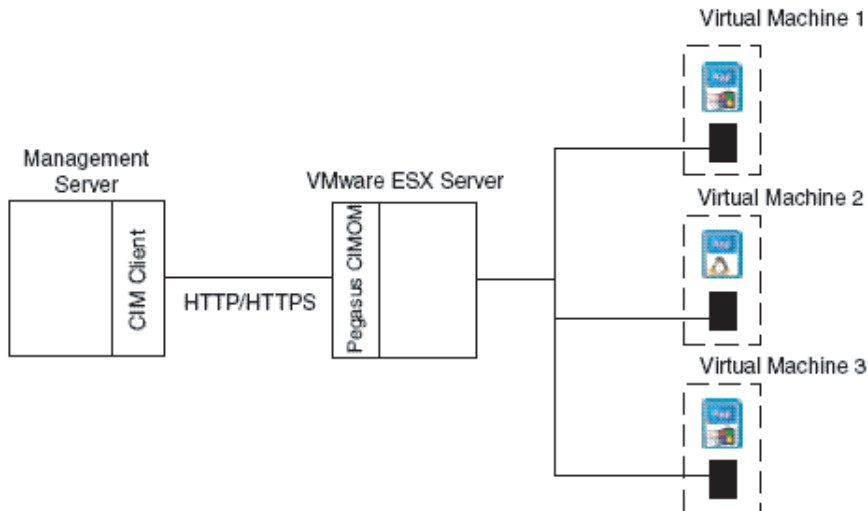
- Pegasus CIMOM to allow clients to explore virtual machines and their allocated resources on ESX Server hosts.
- MOF (Managed Object Format) files specifying the CIM SDK schema.
- *VMware CIM SDK Programming Guide* to provide developers with an introduction to the VMware CIM Extension Schema and with examples of usage.
- *VMware CIM SDK Reference* to provide developers with an in-depth resource about the classes in the VMware CIM Extension Schema.
- Sample clients in several languages that demonstrate CIM SDK functionality.

The CIMOM is installed automatically with ESX Server 3. All other files are provided for download as a zip package. To download the package, refer to <http://www.vmware.com/support/developer>.

CIM SDK Processes

The following figure provides an overview of the software components in a running CIM SDK environment.

Figure 1



The CIM client may be any management application that complies with the SMI-S profile. The message transport mechanism for the VMware CIM SDK is CIM XML over HTTP between the CIM client and the CIMOM. The CIMOM is a part of the ESX Server installation. The virtual machine boxes represent those virtual machines that run on the ESX Server machine, and therefore are visible to a CIM client that communicates with the ESX Server machine.

Starting the Pegasus CIMOM

The Pegasus CIMOM of the OpenPegasus project is installed with ESX Server and can be accessed by any CIM implementation capable of using HTTP or HTTPS to communicate with it. The Pegasus CIMOM is responsible for receiving CIM operation requests from CIM clients and returning appropriate data (CIM operation responses) from underlying provider applications. In the case of ESX Server, the Pegasus CIMOM returns information about the various class objects defined in the VMware extension schema to requesting client applications.

The Pegasus CIMOM runs by default on server startup. To check the status or restart the CIMOM, use the following procedures.

To check the status of the Pegasus CIMOM:

1. Log on to the VMware Service Console using a local or remote connection. Secure Shell (SSH) is supported in the service console.
2. Switch to the `/etc/init.d` directory.
3. Use the command `./pegasus status` to determine the current operational status of the Pegasus CIMOM on the server.

To start the Pegasus CIMOM:

1. Log on to the service console using a local or remote connection. Secure Shell (SSH) is supported on ESX Server.
2. Switch to the `/etc/init.d` directory.
3. Use the command `./pegasus start` to start the Pegasus CIMOM on the server.

To enable the Pegasus CIMOM to run by default on server startup:

1. Log on to the service console using a local or remote connection. Secure Shell (SSH) is supported on ESX Server.
2. Execute the command `/sbin/chkconfig --add pegasus`.

Setting Up Your Development Environment

Due to the platform and language independence of the CIM standard, there are a number of CIM client libraries to choose from for application development. For each CIM implementation, such as OpenPegasus or WBEM Services, consult the product documentation for appropriate requirements, limitations, and setup information. Whatever language and platform you choose for the client, it will need HTTP or HTTPS access from the host on which it runs to the appropriate port on the ESX Server machine.

The VMware CIM SDK does not include a client library. Open source CIM toolkits for client development include:

- OpenPegasus (C++)
- WBEM Services (Java)
- OpenWBEM (C++)

Using the ESX Server Firewall

ESX Server 3.0 includes a built-in firewall. By default, the firewall allows connections to the CIM SDK ports. If the ports are disabled in your installation, you can enable the ports in the VI Client using these steps:

1. Click the Configuration tab.
2. Click Properties (under the Security profile).
3. Enable all the CIM services.

To enable the CIM SDK ports with a command line utility, log on to the service console and run the following commands:

```
esxcfg-firewall --enableService CIMHttpsServer
esxcfg-firewall --enableService CIMHttpServer
esxcfg-firewall --enableService CIMSLP
```

Once you have unblocked the ports, you may connect using the standard protocol for CIM XML over HTTP.

The HTTP server runs on port 5988 (http) or 5989 (https). The CIM SDK resides within the `vmware/esxv2` namespace. Clients must log on using a valid user ID and password known to the ESX Server system.

Note: If you want to use CIM indications in your installation, you need to disable any network address translation firewall between the ESX Server host and the client machine. CIM indications can not be returned through a firewall.

Connecting Your Client to the CIMOM

To connect your client applications to the Pegasus CIMOM on an ESX Server machine, you need the following information:

- Hostname or IP address of the ESX Server machine. Alternatively, you can use SLP discovery to locate the ESX Server host machine.
- Port number on which the Pegasus CIMOM is listening. The port numbers are 5988 for HTTP communication and 5989 for HTTPS communication. These are the settings recommended by the DMTF.
- Username and password.
- SSL context information (if applicable).
- Namespace for the VMware schema extension, which is `vmware/esxv2`.

To use indications with the CIM SDK, the ESX Server host machine must be on the same network as the client machine.

VMware CIM SDK Schema

This chapter explains the schema used by the VMware CIM SDK. Besides conforming to CIM standards in general, the VMware CIM SDK is based on SMI-S standards for storage device modeling. Where the SMI-S profile fails to model ESX Server and virtual machines adequately, VMware extends the schema. The VMware schema extension applies the SMI-S model to elements within the VMware environment while adhering as closely as possible to the SMI-S model.

The chapter contains the following sections:

- Relationship to Industry Standards on page 19 explains how the VMware CIM SDK schema relates to the CIM and SMI-S standards. This includes the specific model profile used by VMware, as well as exceptions to the standard model.
 - Introduction to the VMware CIM Schema on page 27 explains assumptions and conventions used in this chapter to document the VMware CIM SDK schema.
 - Sample System Environments on page 31 provides instance diagrams for a number of VMware storage environment configurations. These configurations are similar to real-world situations your CIM client may encounter. Each sample environment has pointers to more detailed information later in the chapter.
-

- Using the CIM SDK Schema on page 51 explains the structure of VMware storage environments in terms of certain frequent relationship patterns. This section also explains how to navigate the associations and retrieve information. You can use the explanations as building blocks to develop a CIM client that works with the VMware extensions to the standard schema.

Relationship to Industry Standards

The schema for the VMware CIM SDK derives from a standard SMI-S storage management profile, which in turn derives from the CIM standards created by the DMTF (Distributed Management Task Force). In particular, VMware extends the SMI-S In-Band Virtualizer Profile (IBVP), which draws largely on the Device, Network, and System models defined by DMTF. The SMI-S — and the VMware extension — are primarily concerned with storage. The VMware CIM SDK implements version 2.9.0 of the CIM standard and version 1.0.2 of the SMI-S.

DMTF CIM Schema

DMTF defines CIM standards for managing systems, networks, applications, and services.

What CIM Provides

The CIM standards provide the following foundation for creating management applications:

- IDL (Interface Definition Language) — A machine-readable language to describe the classes, methods, and properties in which CIM is defined
- Core Schema — A meta-schema and core model that serve as the foundation on which the CIM model set is built
- Common Data Models — A set of models relating to different areas such as applications, devices, events, and networks
- Web Transport Model — A standard for HTTP-based communication between CIM clients and CIM providers

Parts of the Schema Implemented by VMware

The IBVP profile specifies the implementation of certain CIM intrinsic methods but not others.

VMware supports the following intrinsic methods:

- Associators
- AssociatorNames
- EnumerateClasses
- EnumerateClassNames
- EnumerateInstances
- EnumerateInstanceNames
- GetClass
- GetInstance
- References

- ReferenceNames

VMware does not support the following intrinsic methods:

- CreateClass
- CreateInstance
- DeleteClass
- DeleteInstance
- DeleteQualifier
- ExecQuery
- InvokeMethod
- ModifyClass
- ModifyInstance
- SetProperty
- SetQualifier

Furthermore, VMware does not define any extrinsic methods.

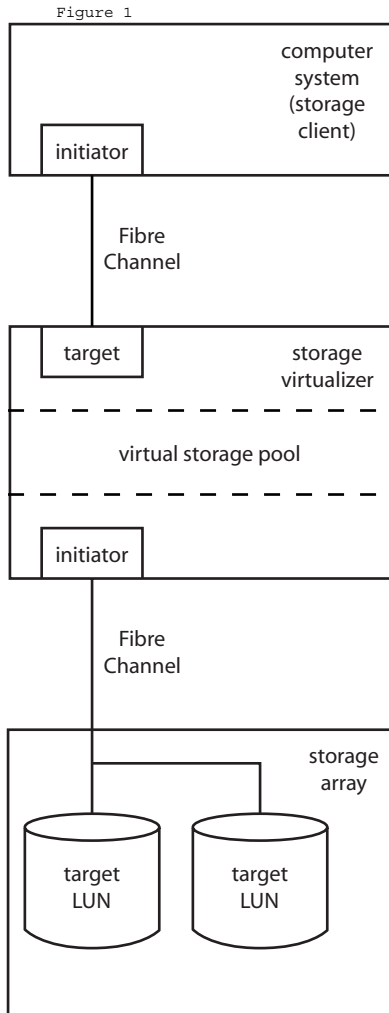
VMware actively participates in the development of DMTF standards for modeling virtualization.

SMI-S Profile

The CIM standards are used by the Storage Networking Industry Association (SNIA) as the basis for standards profiles specific to certain areas of storage management. These profiles specify how the CIM should be applied to a specific kind of storage management. VMware has chosen to implement the IBVP because it is well adapted to model VMware's storage virtualization technology.

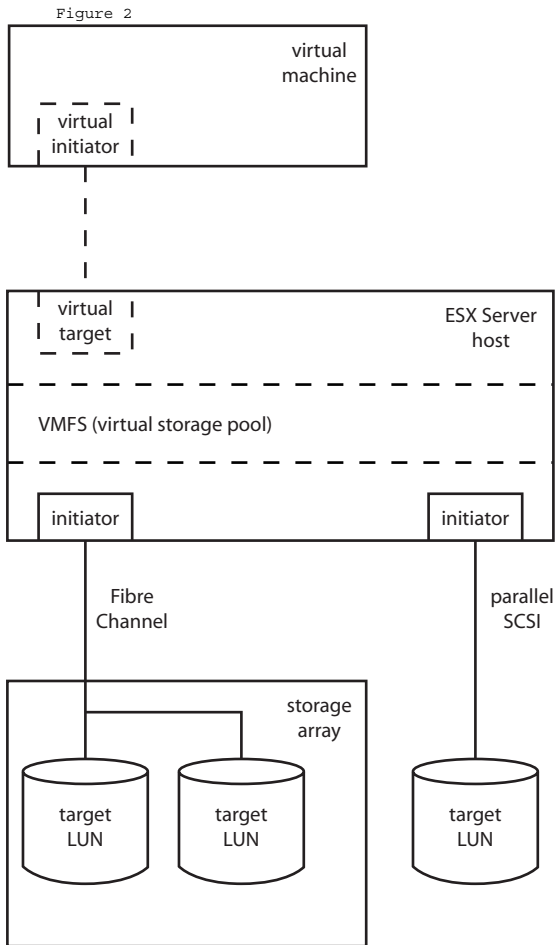
The IBVP is a profile in the Storage category defined by the SNIA. It also has a dependency on the Server profile. The core profile is read-only and does not provide manipulation capabilities.

An in-band virtualizer (Figure 1) merges discrete storage devices into logical pools of storage that it manages for its storage clients. As clients need storage, the virtualizer locates available storage from the pools it manages and allocates the storage to its clients.



VMware ESX Server (Figure 2) closely approximates a storage virtualizer as described in the IBVP model specified by SMI-S version 1.0.2. ESX Server handles Fibre Channel, parallel SCSI, block mode, or iSCSI storage systems. It manages these as logical storage pools in the form of VMFS volumes.

Blocks of VMFS storage are allocated to virtual machines in the form of virtual disk files. The virtual machine acts as the client of a storage virtualizer.



VMware CIM SDK Schema

VMware's schema, based on SMI-S version 1.0.2, implements a set of subprofiles of the IBVP to model virtual storage and its underlying physical storage in an ESX Server environment. These include the core profile and related subprofiles that allow the user to browse the VMware storage environment.

In addition to IBVP, VMware implements the following subprofiles for this release:

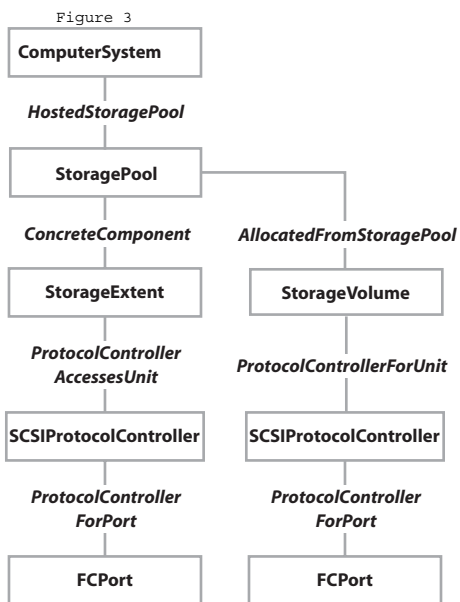
- Software — Version information of the running software

Parts of the VMware CIM SDK schema are experimental. DMTF continues to develop subprofiles that may affect virtual machines. VMware actively participates in future DMTF standards development.

Exceptions to Standard Schema

The VMware extension closely approximates the IBVP, so that existing CIM clients can be adapted to VMware environments with little code change. However, the SMI-S profiles in version 1.0 were designed with only Fibre Channel devices in mind. As a result, the VMware schema departs from the IBVP in one important respect.

In a typical storage virtualization environment, the storage virtualizer has a Fibre Channel connection to the client. Consequently, the IBVP model includes an FCPort object attached to the SCSIProtocolController for the client. The SMI-S schema models the connections as shown in Figure 3, below.



The objects on the right, in conjunction with the two objects at the bottom center of Figure 4, are marked with asterisks to distinguish them from standard objects shared with the SMI-S specification. The objects representing the virtual machine and the virtual initiator port derive from classes used by the SMI-S specification. These objects are associated in patterns very similar to those of the specification.

Figure 4 also shows the use of a DiskPartition object, which derives from the StorageExtent class. The DiskPartition is related to the StorageExtent with a BasedOn association, which can be used to layer arbitrary depths of partitions, consistent with the SMI-S specification. The VMware extension schema uses the DiskPartition object because ESX Server always creates a VMFS within a partition on the target LUN.

Introduction to the VMware CIM Schema

This section explains the assumptions and conventions used in this chapter to document the VMware CIM schema.

Types of Clients

Clients of the CIM SDK may be classified into two categories:

- Clients that are not aware of the VMware extension elements

This kind of client follows the SMI-S model only; when it encounters VMware extension elements, it should ignore those elements and continue to follow the expected paths of the standard model. The result is an accurate presentation of the physical storage visible to an ESX Server host, but the virtual machine storage aspects are missing from the client's view of the storage environment.
- Clients that are aware of VMware extensions to the SMI-S standard

Where VMware has used standard classes in ways that are meaningful only in a VMware environment, this kind of client is designed to follow the extended paths and retrieve additional information about virtual machines and their relationships to host resources.

Conventions Used in This Document

- Names of properties are generally given in `Courier` type so they can be clearly distinguished.
- Literal values of properties or parameters are generally given in quotes. The quotes are not part of the value. Literal values sometimes contain spaces.
- In the CIM interface definition, and consequently in client code, base classes in the CIM model include the `CIM_` prefix. Derived classes created by VMware include the `VMWARE_` prefix. These prefixes are usually omitted in this manual, where the meaning is clear from the context. When you use these names in method calls, you should prefix `"VMWARE_"` to the name if you are specifying the derived class, or prefix `"CIM_"` to the name if you are specifying the parent class.

For example, to invoke a method on the VMware class modeling a virtual machine's SCSI controller, specify `"VMWARE_VirtualInitiatorSCSIProtocolController"`; to invoke a method on the parent class, specify `"CIM_SCSIProtocolController"`.
- Code examples in this chapter are written in C++ using the OpenPegasus client interface. The capitalization of identifiers used in the examples may not match the capitalization used in other client interfaces or in the text. In particular, the text in this chapter uses initial capital letters for method names.

Conventions Used in Illustrations

The illustrations in this chapter show a number of instance diagrams representing sample storage virtualization environments or parts of such environments. Certain conventions are used in the instance diagrams.

- Entities (servers, virtual machines, and storage devices) are shown as boxes drawn with solid lines.
- Associations are shown as solid lines between boxes, with the name of the association placed on each line in italic font.
- Entities and associations are typically labeled with the CIM parent class name in bold type (omitting the `CIM_` prefix). In some cases, the VMware extension adds to the name; in these cases, the addition is in regular (not bold) type. For example, a SCSI controller may be labeled:

VirtualInitiator**SCSIProtocolController**

The full name of the parent class for this example is:

`CIM_SCSIProtocolController`

The full name of the derived class is:

`VMWARE_VirtualInitiatorSCSIProtocolController`

- Long class names in the illustrations are sometimes split into two lines. For example,

VirtualInitiator**SCSIProtocolController**

may appear as

VirtualInitiator
SCSIProtocolController

This is done solely to make the illustrations more compact.

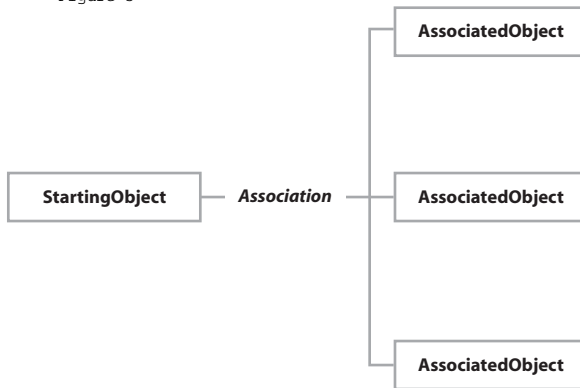
- Some of the boxes in the instance diagrams have an asterisk in the upper left corner of the box. Boxes with asterisks represent parts of the model specific to the VMware environment. A client designed to work strictly with the IBVP can be expected to ignore these instances. CIM clients usually require some code change to explore the areas of the model that are shown with asterisks. These areas should also be considered experimental, as future standards developments may model these areas in a different way.
- The illustrations often show associations that appear to relate more than two objects. This is done for convenience only. CIM associations are always individual instances.

Some of the illustrations are block diagrams of the system environments. These are included to aid in visualizing the entities modeled in the instance diagrams.

Traversing Associations

The CIM uses association classes to model the object-oriented notions of equality, aggregation, and other specialized relationships between objects. If you have a reference to an arbitrary object, and you want to discover and retrieve related objects, you do it by traversing the association instance that relates them (see Figure 5).

Figure 5



There are several approaches available to traverse an association instance and discover the related objects referenced by the association. The approach you use depends on what information you need and your discovery strategy. Here are some approaches you can use:

- Use the `Associators` method with a reference to the starting object. This returns all objects associated with the starting object. While this is the simplest approach, it also generates the largest network transfer when it returns the contents of a number of objects at once.
- Use the `AssociatorNames` method with a reference to the starting object to get references to all associated objects. Select one or more of the references and use the `GetInstance` method to get the content of the object or objects of interest. This takes two steps, but the total network traffic is less. This way is a good choice if you are interested only in one associated object or a few associated objects out of a large number.
- Use the `References` method with a reference to the starting object and the class name of an association. This returns all instances of the association, each instance containing a reference to one of the associated objects. This way is a good choice if you are interested in properties of the association.
- Use the `AssociatorNames` method several times in a row to traverse a series of association objects to get to the object of interest. Each invocation of `AssociatorNames` returns a reference or a set of references to associated objects. Choose an object from the set (if there is

more than one reference returned) and invoke `AssociatorNames` on that reference to obtain another reference or set of references, until you reach the object of interest. This way is especially useful for traversing a series of one-to-one associations.

Sample System Environments

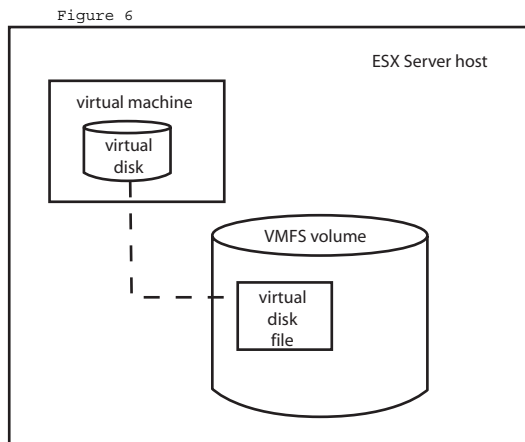
These illustrations show how the CIM SDK represents various sample environments containing different arrangements of servers, virtual machines, and storage arrays. This section contains the following sample environments:

- Virtual Machine with Single Virtual Disk on Local VMFS Storage on page 31
- Virtual Machine with Virtual Disk and Snapshot on Local Storage on page 33
- Virtual Machine with Single Virtual Disk on a LUN in a Storage Array on page 36
- Virtual Machine Using Raw Device Mapping in a Storage Array on page 39
- Two Virtual Machines on Different Servers Accessing Shared LUN on page 42
- Multipath SAN Environment with Two Storage Arrays on page 47

Each sample environment has a textual description and an accompanying block diagram that shows the logical parts of the environment from the perspective of the ESX Server host. After the block diagram there is an instance diagram that shows how the environment is modeled to a CIM client. Portions of the instance diagram are described in accompanying text, which refers you to a later section in the chapter that describes in detail how to work with that portion of the model.

Virtual Machine with Single Virtual Disk on Local VMFS Storage

This example environment has a single virtual machine and a single virtual disk file stored on the host's local storage, using a parallel SCSI connection. Logically, it looks like the diagram in Figure 6, below.



This environment is represented to a CIM client as shown in Figure 7, below.



Figure 7 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 6 on page 31.

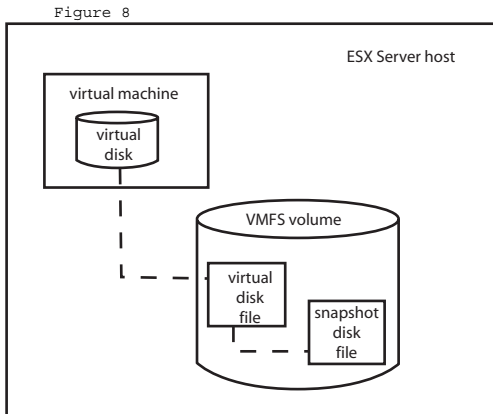
- The objects and their associations in the lower left area of Figure 7 represent the virtual machine and its virtual HBA. For more detail on this area, refer to Virtual Host Bus Adapters on a Virtual Machine on page 53.

- The four objects and their associations in the lower right area of Figure 7 represent the virtual disk attached to the virtual HBA. For more detail on this area, refer to Virtual Disks Connected to a Virtual Host Bus Adapter on page 55.
- The StoragePool object represents a VMFS. For more detail on this topic, refer to Virtual Disks on a VMFS on page 58.
- The three objects and their associations in the upper right area of Figure 7 represent the direct-attached host storage. For more detail on this area, refer to VMFS Volumes Sharing a Directly Attached Disk on page 59.

Note: The VMware CIM SDK models iSCSI with the same objects as direct-attached storage. If your host has networked storage (except for Fibre Channel storage), it can't be distinguished from the direct-attached storage. For more detail on Fibre Channel storage, refer to VMFS Volumes Sharing a Fibre Channel LUN on page 64.

Virtual Machine with Virtual Disk and Snapshot on Local Storage

This sample environment has a single virtual machine with a single virtual disk using the host's local storage. In this example, the virtual disk file is supplemented with a snapshot file. Logically, it looks like the diagram in Figure 8, below.



This environment is represented to a CIM client as shown in Figure 9, below. The two disk files are associated as synchronized storage.

Figure 9

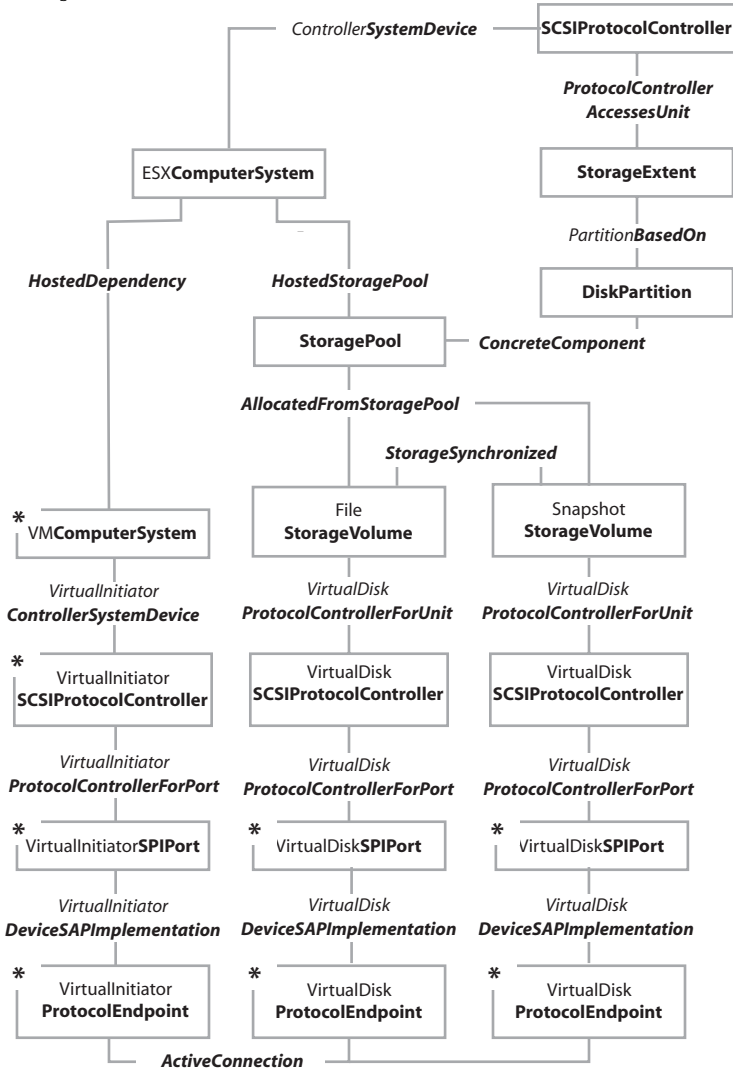
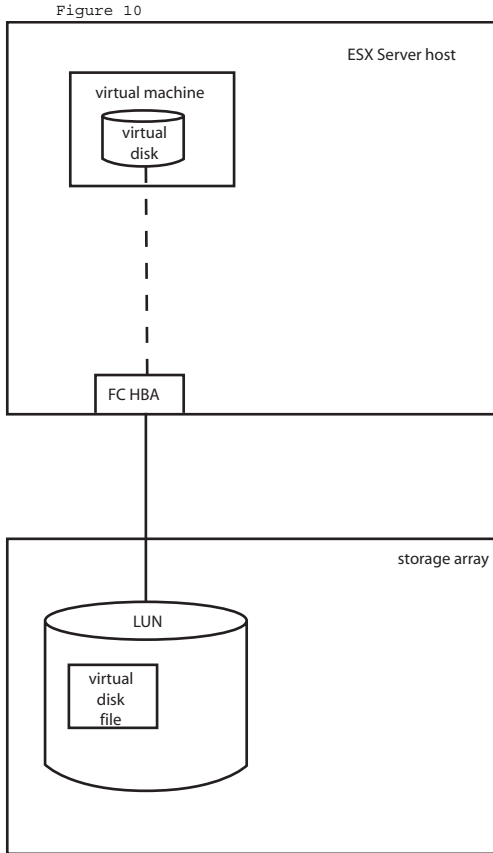


Figure 9 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 8 on page 33.

- The four objects and their associations along the left side of Figure 9 represent the virtual machine and its virtual HBA. For more detail on this area, refer to *Virtual Host Bus Adapters on a Virtual Machine* on page 53.
- The four objects and their associations in the middle of the lower part of Figure 9 represent the file or set of files implementing the base virtual disk. For more detail on this area, refer to *Virtual Disks Connected to a Virtual Host Bus Adapter* on page 55.
- The four objects and their associations in the lower right area of Figure 9 represent the file or set of files representing the snapshot virtual disk. The guest operating system in the virtual machine is unaware of the snapshot file. The host presents it as an integral part of the virtual disk. Except for the *StorageSynchronized* association between the snapshot disk and the base disk, the snapshot disk is modeled in the same way as any virtual disk. Refer to the section *Virtual Disks Connected to a Virtual Host Bus Adapter* on page 55 for information on working with virtual disk objects.
- The *StoragePool* object represents the VMFS. For more detail on this area, refer to *VMFS Volumes Sharing a Directly Attached Disk* on page 59.

Virtual Machine with Single Virtual Disk on a LUN in a Storage Array

This example environment has a single virtual machine with a single virtual disk located on a Fibre Channel SAN rather than on local storage. Logically, it looks like the diagram in Figure 10, below.



This environment is represented to a CIM client as shown in Figure 11, below.

Figure 11

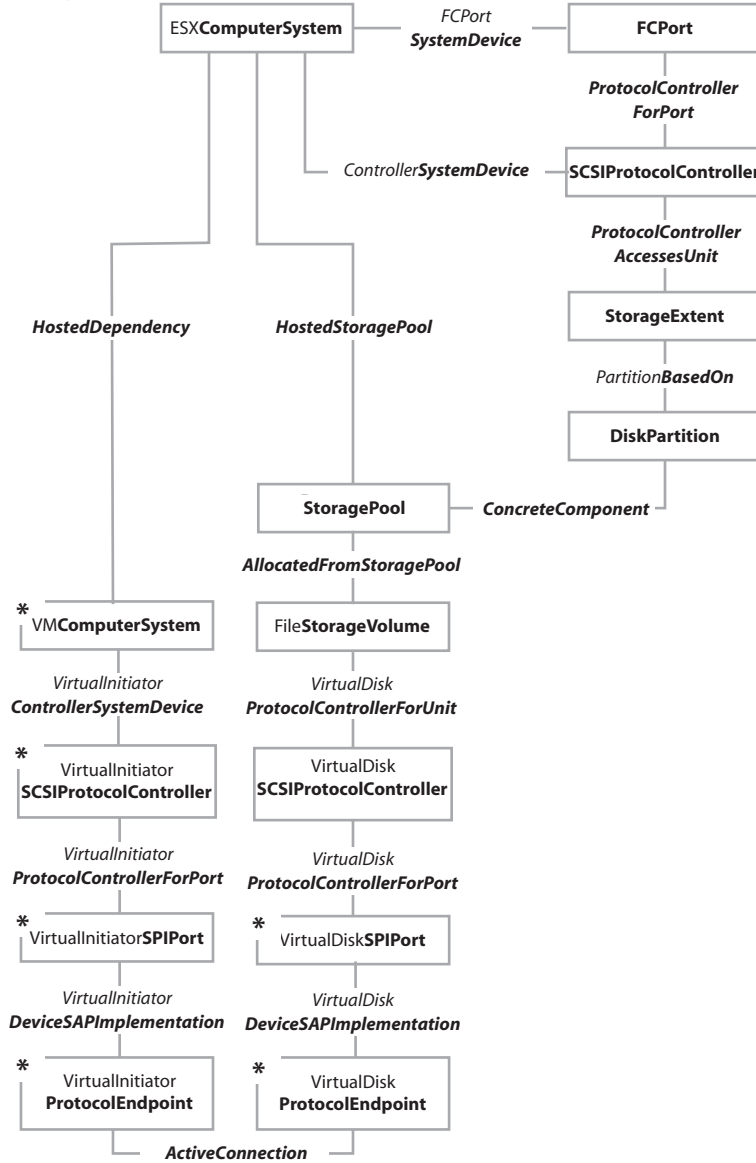
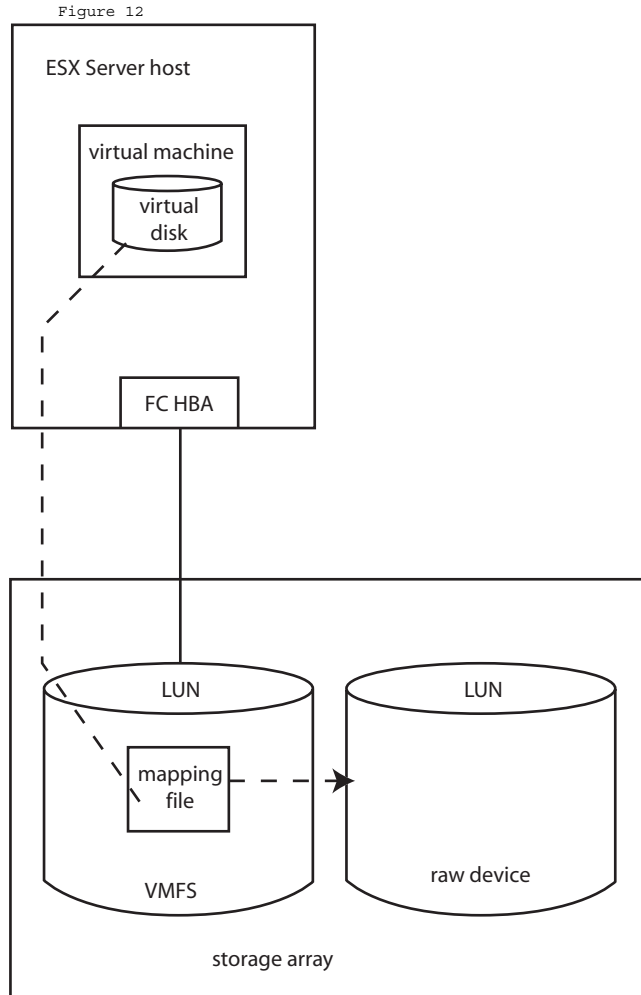


Figure 11 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 10 on page 36.

- The four objects and their associations along the left side of Figure 11 represent the virtual machine and its virtual HBA. For more detail on this area, refer to [Virtual Host Bus Adapters on a Virtual Machine](#) on page 53.
- The four objects and their associations in the middle of the lower part of Figure 11 represent the file or set of files implementing the base virtual disk. For more detail on this area, refer to [Virtual Disks Connected to a Virtual Host Bus Adapter](#) on page 55.
- The four objects and their associations in the upper right area of Figure 11 represent the Fibre Channel HBA and the LUN that holds the VMFS. For more detail on this area, refer to [VMFS Volumes Sharing a Fibre Channel LUN](#) on page 64.
- The StoragePool object in the center represents the VMFS. For more detail on this area, refer to [VMFS Volumes Sharing a Fibre Channel LUN](#) on page 64.

Virtual Machine Using Raw Device Mapping in a Storage Array

This example environment has a single virtual machine with a single virtual disk whose data is stored on a raw LUN on a Fibre Channel SAN, but accessed with a raw device mapping. Logically, it looks like the diagram in Figure 12, below.



This environment is represented to a CIM client as shown in Figure 13, below.

Figure 13

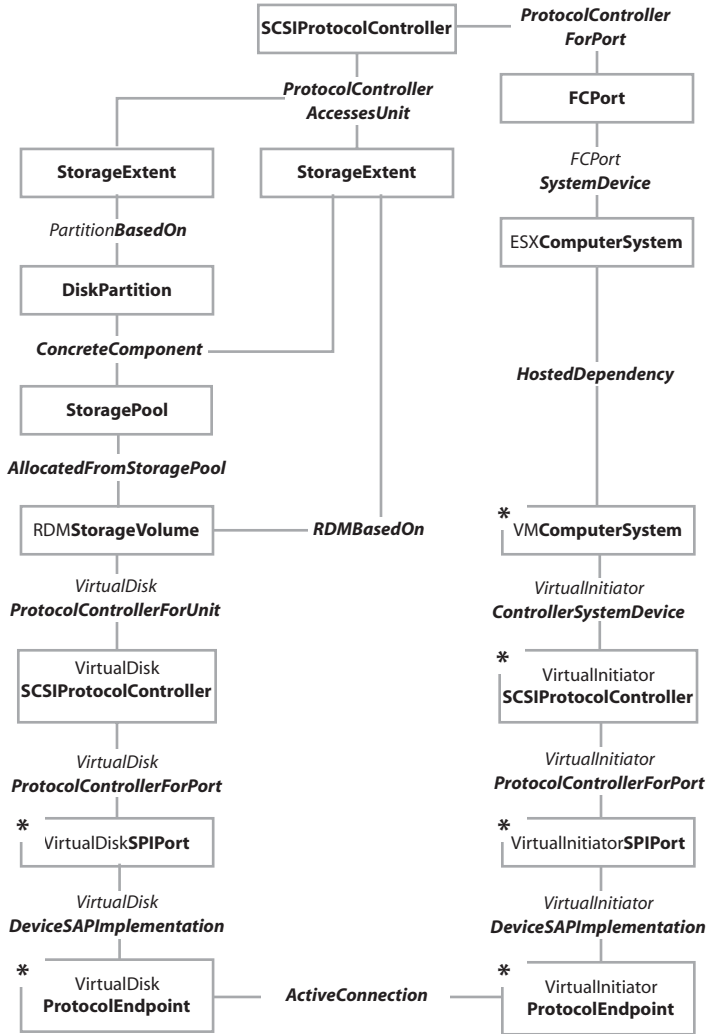
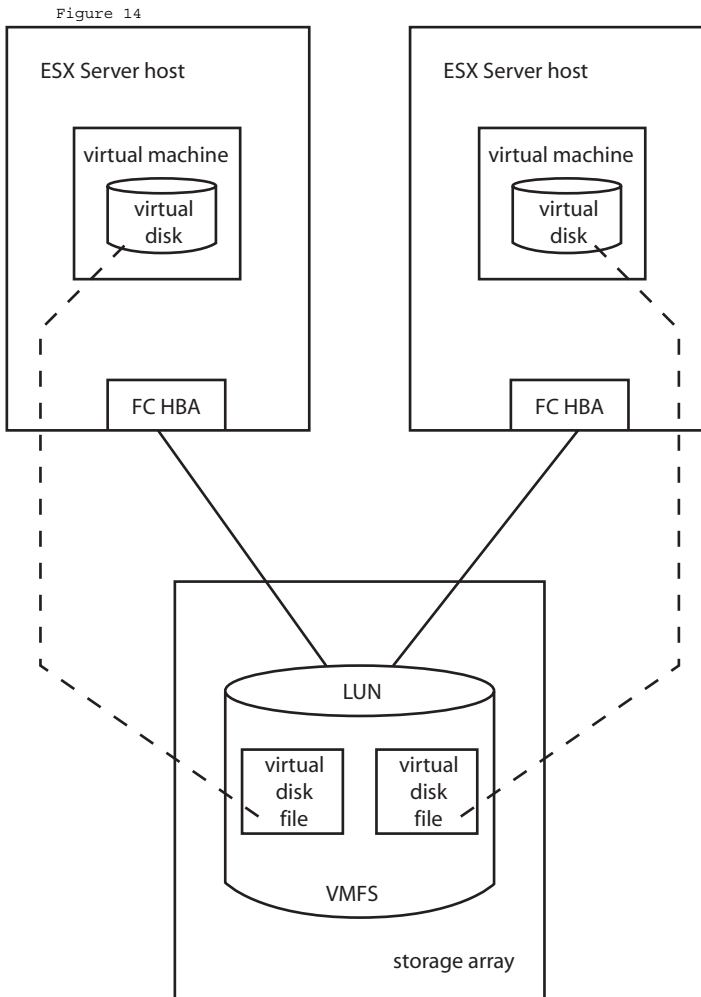


Figure 13 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 12 on page 39.

- The four objects and their associations in the lower right area of Figure 13 represent the virtual machine and its virtual HBA. For more detail on this area, refer to Virtual Host Bus Adapters on a Virtual Machine on page 53.
- The SCSIProtocolController and the FCPort objects at the top of Figure 13 represent the host's Fibre Channel HBA. For more detail on this area, refer to VMFS Volumes Sharing a Fibre Channel LUN on page 64.
- The four objects and their associations in the lower left area of Figure 13 represent the virtual disk (implemented with a mapping file, in this case). For more detail on this area, refer to Virtual Disks Connected to a Virtual Host Bus Adapter on page 55.
- The StorageExtent, DiskPartition, and StoragePool objects in the upper left area of Figure 13 represent a partitioned LUN with a VMFS. For more detail on this area, refer to VMFS Volumes Sharing a Directly Attached Disk on page 59. The mapping file is always stored on a VMFS.
- The StorageExtent object in the middle near the top of Figure 13 represents the raw LUN that holds virtual disk data. It is the raw storage part of the mapping. Unlike the StorageExtent associated with the mapping file, this LUN has no associated DiskPartition as its basis. The ConcreteComponent association represents the raw LUN's function as an extension of the VMFS pool storage. The RDMBasedOn association models the mapping relationship between the raw LUN containing the storage and the VMFS mapping file containing the reference.

Two Virtual Machines on Different Servers Accessing Shared LUN

This example environment has two virtual machines running on different hosts. Each virtual machine has a single virtual disk stored on a shared VMFS volume. Logically, it looks like the diagram in Figure 14, below.



In this situation, the CIM SDK client cannot see both halves of the environment simultaneously. Each host can return information about the storage environment of which it

is aware — including the shared LUN — but not about the storage connections to the other host. To get complete information about the shared environment, the client needs to make a separate connection to each host. Figure 15 and Figure 16, below, show the parts of the environment visible to each of the two hosts.

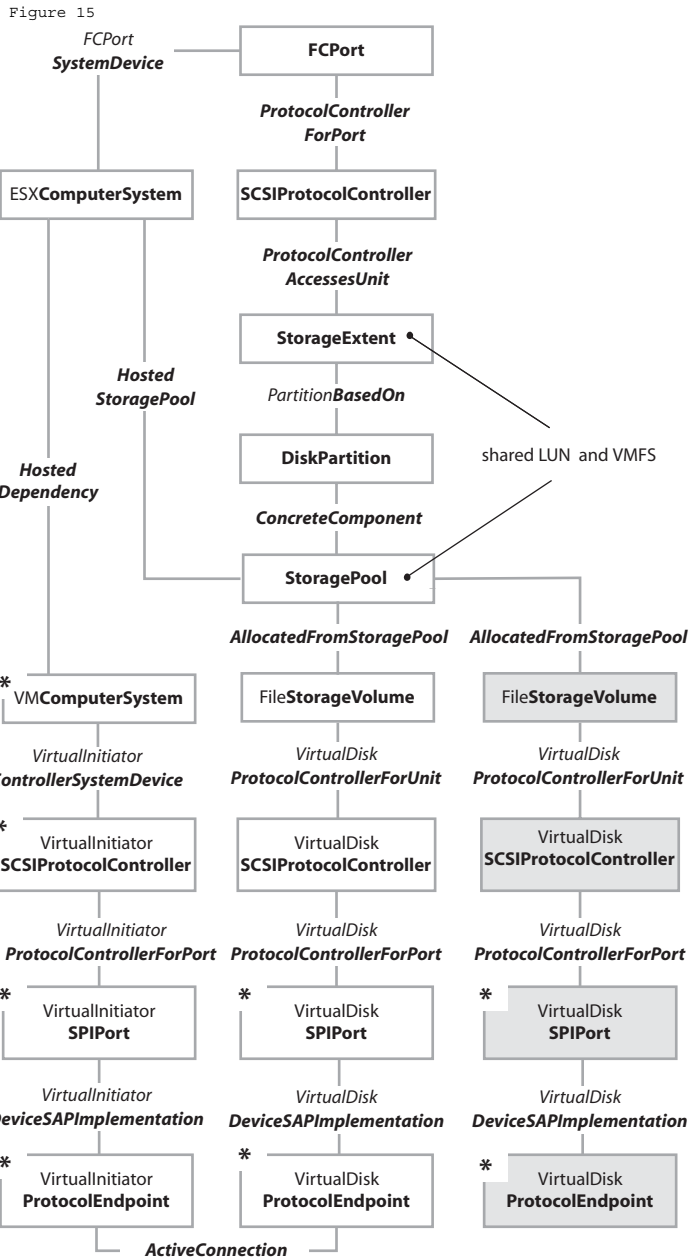


Figure 15 is the part of the virtual storage environment visible to one host. It can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 14 on page 42. The shaded boxes on the right side of Figure 14 represent the objects visible to the client when connected to the second host.

- The three objects and their associations at the top of Figure 15 represent the host and its Fibre Channel HBA. For more detail on the Fibre Channel HBA model, refer to VMFS Volumes Sharing a Fibre Channel LUN on page 64.
- The four objects and their associations in the lower left area of Figure 15 represent the virtual machine running on the first host, including its virtual HBA. For more detail on this topic, refer to Virtual Host Bus Adapters on a Virtual Machine on page 53.
- The four objects and their associations in the middle lower part of Figure 15 represent the virtual disk belonging to the virtual machine on this host. For more detail on this area, refer to Virtual Disks Connected to a Virtual Host Bus Adapter on page 55.
- The four shaded boxes on the lower right represent the virtual disk belonging to the virtual machine running on the second host. Although that virtual machine is not visible to the first host, the corresponding virtual disk on the shared VMFS volume is visible to both hosts.
- The StorageExtent object and DiskPartition object represent the partitioned LUN visible to the first host. This is a shared LUN, and is likewise visible to the second host. For more detail on the LUN model, refer to VMFS Volumes Sharing a Directly Attached Disk on page 59.
- Similarly, the StoragePool object represents the shared VMFS volume — also visible to both hosts.

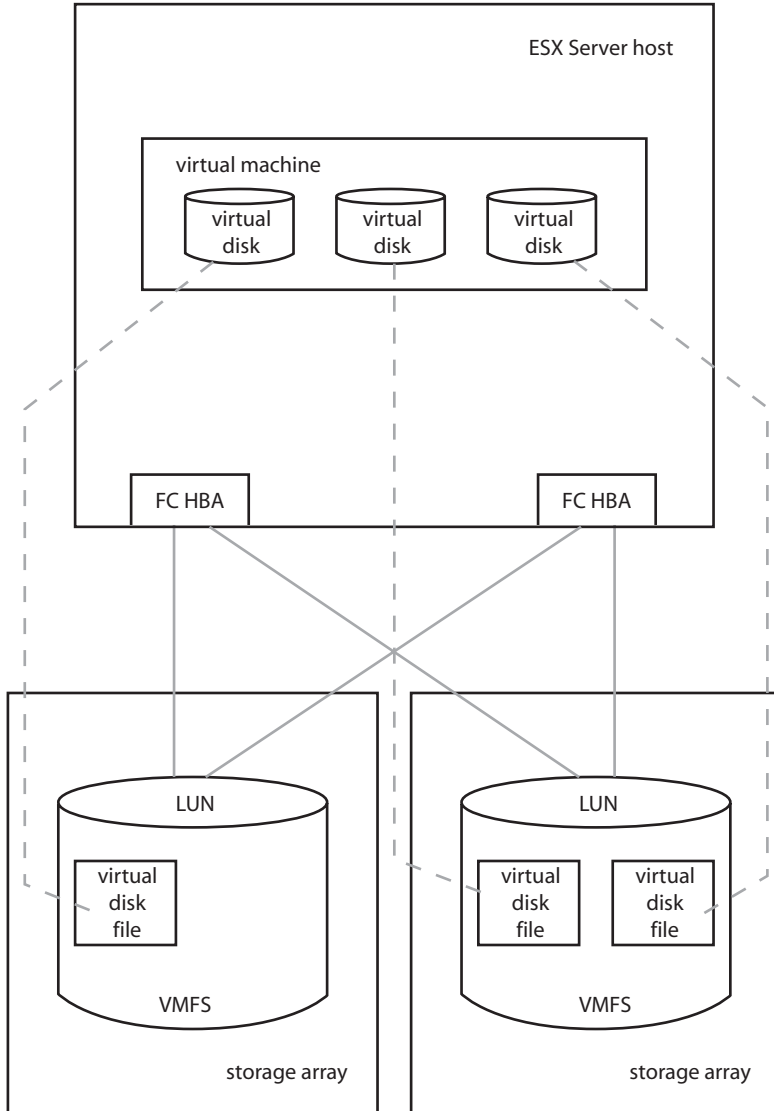
Figure 16, below, shows the part of the virtual storage environment visible to the second host. Figure 16 contains the same kinds of objects and associations as Figure 15. The shaded boxes in both figures represent the objects pertaining to the second host, while the clear boxes represent the objects pertaining to the first host. However, both figures include boxes representing the shared VMFS and the partitioned LUN on which it is stored.

To detect that the LUNs and the VMFS volumes are the same, the client needs to compare their UUIDs. Getting the UUID of a VMFS volume is described in the section Virtual Disks on a VMFS on page 58.

Multipath SAN Environment with Two Storage Arrays

This sample environment has a single virtual machine running on a host with redundant Fibre Channel paths to the virtual machine's three virtual disks. There is only a single virtual machine for the sake of simplicity; in practice, a multipath environment likely includes a number of virtual machines running on more than one host. Logically, this simplified environment looks like the diagram in Figure 17, below.

Figure 17



This environment is represented to a CIM client as shown in Figure 18, below. Note that the CIM SDK does not model Fibre Channel ports for the storage arrays.

Figure 18 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 17 on page 48.

- The four objects and their associations in the lower left area of Figure 18 represent a virtual disk. For more detail on this area, refer to *Virtual Disks Connected to a Virtual Host Bus Adapter* on page 55. Next to it is a second virtual disk stored on the same VMFS volume, as indicated by the `AllocatedFromStoragePool` link to the same `StoragePool` object above them.
- The four objects and their associations to the right of the middle lower part of Figure 18 represent a third virtual disk, which is linked to a different `StoragePool` (VMFS volume) by the `AllocatedFromStoragePool` association.
- The four objects and their associations on the right edge of Figure 18 represent the virtual machine and its virtual HBA. For more detail on this area, refer to *Virtual Host Bus Adapters on a Virtual Machine* on page 53. The virtual HBA is linked to all three virtual disks with the `ActiveConnection` association.
- The `StorageExtent`, `DiskPartition`, and `StoragePool` objects along the left edge of Figure 18 represent a partitioned LUN with a VMFS. For more detail on this area, refer to *VMFS Volumes Sharing a Directly Attached Disk* on page 59. Across from them to the right are the corresponding objects and associations for the other LUN and VMFS attached to the host.
- The `FCPort` and `SCSIProtocolController` objects at the upper left represent one of the host's Fibre Channel HBAs. For more detail on this area, refer to *VMFS Volumes Sharing a Fibre Channel LUN* on page 64. Across from them to the right are the corresponding objects and associations for the host's other HBA. Note that the `SCSIProtocolController` objects both have cross-connections to both `StorageExtent` objects, via the `ProtocolControllerAccessesUnit` associations. These cross-connections represent the redundant Fibre Channel connections between the two HBAs and the two LUNs.

Using the CIM SDK Schema

This section shows how the VMware association classes are used to represent relationships between objects representing systems and storage devices modeled by the VMware extension. Each illustration shows only a small part of a real-world environment. Some larger sample environments are illustrated under Sample System Environments on page 31.

If your discovery approach begins with the host, you need to begin by getting a reference to the `ESXComputerSystem`. The easiest way is to use the `EnumerateInstances` method on the `ESXComputerSystem` class.

An alternative approach, useful if your client is not VMware-aware, is to use `EnumerateInstances` on the `ComputerSystem` class, and reject all but the host system. The host can be distinguished by the presence of the `Dedicated` property set to `"Block Server"`.

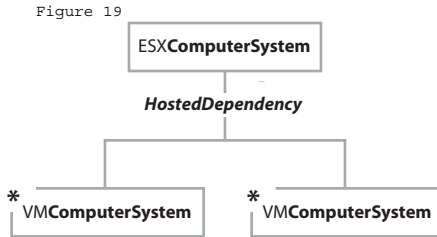
The following subsections show typical relationship patterns in the VMware extension schema:

- Virtual Machines on a Host on page 52
- Virtual Host Bus Adapters on a Virtual Machine on page 53
- Virtual Disks Connected to a Virtual Host Bus Adapter on page 55
- Virtual Disks on a VMFS on page 58
- VMFS Volumes Sharing a Directly Attached Disk on page 59
- VMFS Volumes Sharing a Fibre Channel LUN on page 64
- VMFS Spanning Two LUNs on page 66
- Raw Device Mapping on page 67
- System Devices of a Host on page 69

Each subsection presents a diagram of the CIM SDK model for a typical small part of an ESX Server storage environment. Following each diagram is one or more typical ways to navigate that part of the model and retrieve information.

Virtual Machines on a Host

The CIM SDK models host machines and virtual machines as subclasses of `CIM_ComputerSystem`. They are linked by a `VMWARE_HostedDependency` association, which is a subclass of `CIM_HostedDependency`. See Figure 19, below.



Typical Uses

To get the name of the ESX Server system, a simple way is to enumerate all instances of the `ESXComputerSystem` class. There is only one `ESXComputerSystem` available to a given client connection. Example 1 shows this way to find the host object.

Example 1

```

Array<CIMInstance> esxcs;
esxcs = client.enumerateInstances(NAMESPACE,
                                "VMWARE_ESXComputerSystem");
// Can only get 1 ESXComputerSystem from each connection.
CIMInstance theESXInstance = esxcs[0];
CIMProperty prop = theESXInstance.getProperty(
    theESXInstance.findProperty("ElementName"));
String EsxName = prop.getValue().toString();
  
```

A less direct way to get the `ESXComputerSystem` object avoids the VMware-specific class name by using the name of the parent class. To do this, enumerate all `ComputerSystem` objects and choose the one representing the host. The host can be distinguished by the presence of the `Dedicated` property set to "Block Server". Example 2 shows this way to find the host object.

Example 2

```

Array<CIMInstance> cs;
CIMInstance theEsxInstance = NULL;
cs = client.enumerateInstances(NAMESPACE, "CIM_ComputerSystem");
if (cs.size() > 0) {
    for (int i = 0; i < cs.size(); i++) {
        CIMProperty prop = cs[i].getProperty(
            cs[i].findProperty("Dedicated"));
        if (!prop.isUninitialized()) {
  
```

```

CIMValue val = prop.getValue();
// ESXComputerSystem must have at least 2 specified properties.
if (val.isNull() || !val.isArray()) {
    break;
}
Array<String> props;
val.get(props);
for (int i = 0; i < props.size(); i++) {
    if (props[i] == 14 /* Block Server */) {
        val.get(theEsxInstance);
    }
} // props loop
} // cs loop
} // !prop.isUninitialized()
} // cs.size() > 0

```

To find all the virtual machines managed by the ESX Server host system, you can use the `Associators` method with a reference to the `ESXComputerSystem` object and the class name `"VMWARE_HostedDependency"`. This returns copies of all the `VMComputerSystem` objects. Example 3 determines the number of virtual machines.

```

Example 3
Array<CIMObject> vmcs;
int VMcount = 0;
vmcs = client.associators(NAMESPACE, theESXInstance.getPath(),
                        "VMWARE_HostedDependency");
VMcount = vmcs.size();

```

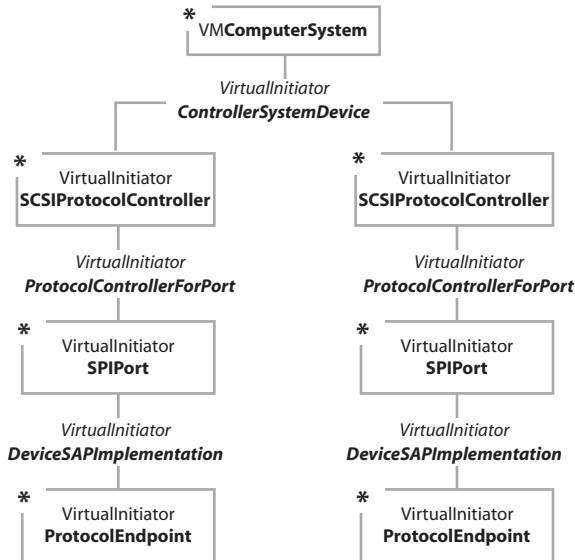
Virtual Host Bus Adapters on a Virtual Machine

The CIM SDK models for virtual storage can be viewed as two groups of objects: virtual host bus adapters (HBAs) and virtual disks. Figure 20 (below) shows the objects present for a virtual machine with two virtual HBAs. Each virtual HBA is modeled with three objects:

- A `SCSIProtocolController` object
- A `SPIPort` object
- A `ProtocolEndpoint` object

Each of these objects models a different aspect of a virtual HBA. Each association between them is one-to-one.

Figure 20



Typical Uses

To identify the virtual HBAs configured for a virtual machine, use the `AssociatorNames` method with a reference to the `VMComputerSystem` object and the class name `"CIM_SystemDevice"`, also qualified by the result class name `"CIM_SCSIProtocolController"`. This produces a list of `VirtualInitiatorSCSIProtocolController` object references — one reference for each virtual HBA.

Example 4

```
Array<CIMObjectPath> vctlr;
vctlr = client.associatorNames(NAMESPACE, theVMInstance.getPath(),
    "CIM_SystemDevice",
    "CIM_SCSIProtocolController");
```

If you also want to identify the virtual disks on a virtual HBA, then you need to traverse the associations to the `ProtocolEndpoint` objects:

Example 5

```
// Visit all virtual HBAs for the virtual machine.
for (int i = 0; i < vctlr.size(); i++) {
```

1. Use the `AssociatorNames` method with a reference to the `VirtualInitiatorSCSIProtocolController` object and the class name of the

ProtocolControllerForPort association (either the parent class or the derived class). This is a one-to-one association that produces a VirtualInitiatorSPIPort object reference.

Example 6

```
Array<CIMObjectPath> viport;
viport = client.associatorNames(NAMESPACE, vctlr[i],
    "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViPortRef = viport[0];
```

2. Use the AssociatorNames method with a reference to the VirtualInitiatorSPIPort object and the class name of the DeviceSAPImplementation association (either the parent class or the derived class). This is a one-to-one association that produces a VirtualInitiatorProtocolEndpoint object reference.

Example 7

```
Array<CIMObjectPath> viendpt;
viendpt = client.associatorNames(NAMESPACE, theViPortRef,
    "CIM_DeviceSAPImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViEndpointRef = viendpt[0];
```

3. From the VirtualInitiatorProtocolEndpoint object, you can proceed to identify attached virtual disks, as described in Virtual Disks Connected to a Virtual Host Bus Adapter, below.

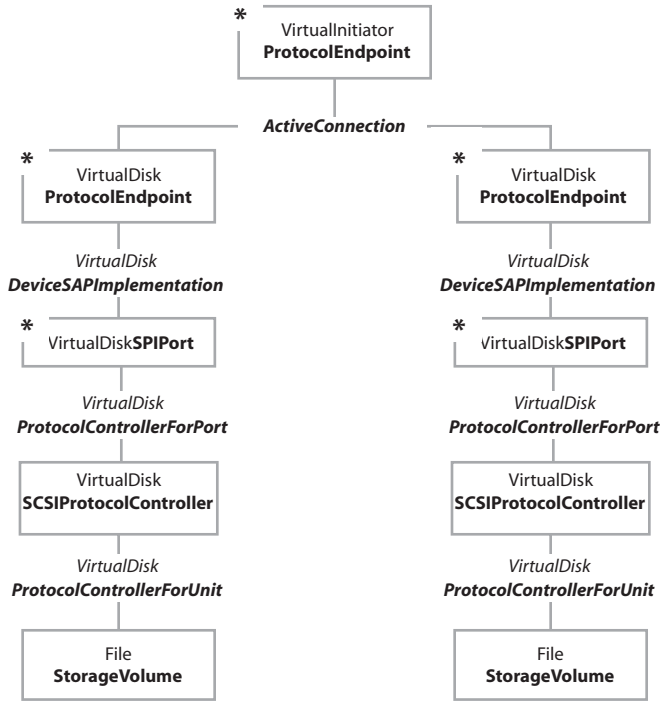
Virtual Disks Connected to a Virtual Host Bus Adapter

The CIM SDK models virtual storage in two parts: virtual HBAs and virtual disks. Figure 21 (below) shows the objects present for a virtual HBA with two virtual disks. Each virtual disk is modeled with four objects:

- A ProtocolEndpoint object that connects to the virtual HBA's ProtocolEndpoint object
- A SPIPort object (Note that this is a departure from the SMI-S standard. It is present in the VMware extension schema because ESX Server virtualizes a SCSI device with a parallel SCSI connection rather than a Fibre Channel connection.)
- A SCSIProtocolController object
- A StorageVolume object

Each of these objects models a different aspect of a virtual HBA. Each association between them is one-to-one.

Figure 21



Typical Uses

To identify the virtual disks attached to a virtual HBA, use the `AssociatorNames` method with a reference to the `VirtualInitiatorProtocolEndpoint` object for the virtual HBA and the class name of the `ActiveConnection` association (either the parent class or the derived class). This produces a list of `VirtualDiskProtocolEndpoint` object references — one reference for each virtual disk attached to the virtual HBA.

Example 8

```

Array<CIMObjectPath> vndndpt;
vndndpt = client.associatorNames(NAMESPACE, theViEndpointRef,
    "CIM_ActiveConnection");
    
```

To retrieve information about a specific virtual disk, you need to traverse the associations to reach the `FileStorageVolume` object:

1. Use the `AssociatorNames` method with a reference to one of the `VirtualDiskProtocolEndpoint` objects and the class name of the `DeviceSAPImplementation` association (either the parent class or the derived class). This produces a reference to the `VirtualDiskSPIPort` object.

Example 9

```
Array<CIMObjectPath> vdport;
vdport = client.associatorNames(NAMESPACE, theVdEndpointRef,
    "CIM_DeviceSAPImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdPortRef = vdport[0];
```

2. Use the `AssociatorNames` method with a reference to the `VirtualDiskSPIPort` object and the class name of the `ProtocolControllerForPort` association (either the parent class or the derived class). This produces a reference to the `VirtualDiskSCSIProtocolController` object.

Example 10

```
Array<CIMObjectPath> vdctlr;
vdctlr = client.associatorNames(NAMESPACE, theVdPortRef,
    "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdCtlrRef = vdctlr[0];
```

3. Use the `Associators` method with a reference to the `VirtualDiskSCSIProtocolController` object and the class name of the `ProtocolControllerForUnit` association (either the parent class or the derived class). This produces a copy of the `FileStorageVolume` object.

Example 11

```
Array<CIMObject> fsvol;
fsvol = client.associators(NAMESPACE, theVdCtlrRef,
    "CIM_ProtocolControllerForUnit");
// One-to-one mapping produces a 1-element array.
CIMObject theFSVolInstance = fsvol[0];
```

4. To determine the capacity of the virtual disk, use the client's accessor method to get the `NumberOfBlocks` property and the `BlockSize` property from the `FileStorageVolume` object.

Example 12

```
CIMProperty prop;
prop = theFSVolInstance.getProperty(
    theFSVolInstance.findProperty("NumberOfBlocks"));
uint64 numBlocks = prop.getValue();
prop = theFSVolInstance.getProperty(
    theFSVolInstance.findProperty("BlockSize"));
uint64 blockSize = prop.getValue();
```

- Multiply the `NumberOfBlocks` by the `BlockSize` to get the number of bytes available.

Example 13

```
uint64 vdSizeBytes = numBlocks * blockSize;
```

Virtual Disks on a VMFS

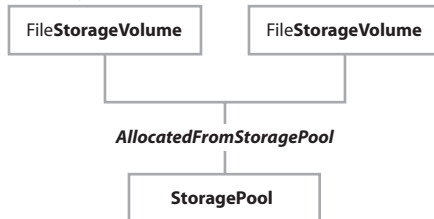
Figure 22 (below) shows the relationship between virtual storage volumes (virtual disk files) and the VMFS file system on which they are stored. Every virtual disk is backed by a file or set of files on a VMFS volume. Individual files are not modeled by the SMI-S standard, but virtual disk files represent a logical unit of storage to the virtual machine; thus, they are modeled by a `FileStorageVolume` object. The VMFS is modeled by a `StoragePool` object.

The virtual storage volumes are linked by an `AllocatedFromStoragePool` association to their VMFS volume. This many-to-one association has a `SpaceConsumed` property for each `FileStorageVolume` associated with a `StoragePool`. The `SpaceConsumed` property represents the actual size of the virtual disk file.

Typically, the value of `SpaceConsumed` is different from the size of the virtual disk reported to the virtual machine. This happens for two reasons:

- For a sparse virtual disk, the value of `SpaceConsumed` is a function of the amount of storage written to by the virtual machine: As the virtual machine writes new sectors to the virtual disk, it causes the virtual disk file to consume more space from the VMFS.
- For a flat virtual disk, the storage is pre-allocated from the VMFS to the full capacity of the virtual disk. However, the value of `SpaceConsumed` also includes redundancy used by RAID storage methods, so the value may exceed the size reported to the virtual machine.

Figure 22



Typical Uses

To determine the amount of physical storage space in use by a virtual disk:

- Use the `References` method with a reference to the `FileStorageVolume` object and the class name of the `AllocatedFromStoragePool` association (either the parent class or the derived class). This produces a copy of the `AllocatedFromStoragePool` association.

Example 14

```

Array<CIMObject> alloc;
alloc = client.references(NAMESPACE, theFSVolInstance.getPath(),
    "CIM_AllocatedFromStoragePool");
// Many-to-one mapping produces a 1-element array.
CIMObject theAllocFromPoolInstance = alloc[0];

```

2. Use the client's accessor method to get the `SpaceConsumed` property from the association instance.

Example 15

```

CIMProperty prop = theAllocFromPoolInstance.getProperty(
    theAllocFromPoolInstance.findProperty("SpaceConsumed"));
CIMValue val = prop.getValue();
uint64 spaceConsumed = 0;
if (!val.isNull) {
    val.get(spaceConsumed);
}

```

To get the unique identifier for the VMFS containing the virtual disk file:

1. Use the `Associators` method with a reference to the `FileStorageVolume` object and the class name of the `AllocatedFromStoragePool` association (either the parent class or the derived class). This produces a copy of the `StoragePool` object.

Example 16

```

Array<CIMObject> pool;
pool = client.associators(NAMESPACE, theFSVolInstance.getPath(),
    "CIM_AllocatedFromStoragePool");
// Many-to-one mapping produces a 1-element array.
CIMObject theStoragePoolInstance = pool[0];

```

2. Use the client's accessor method to get the `InstanceID` property from the `StoragePool` object.

Example 17

```

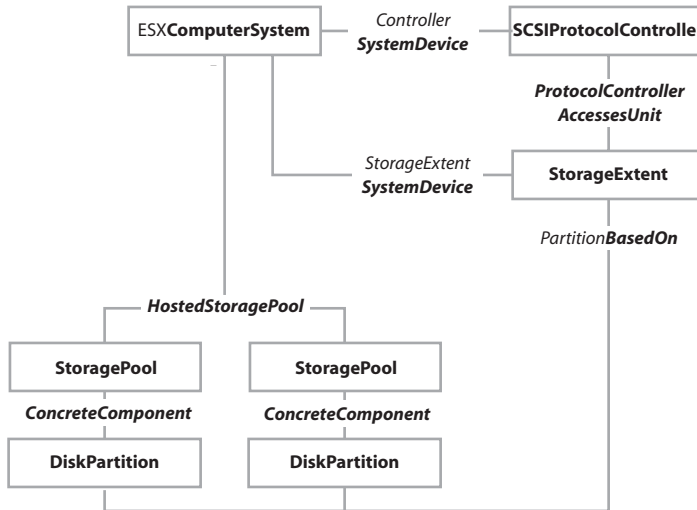
CIMProperty prop = theStoragePoolInstance.getProperty(
    theStoragePoolInstance.findProperty("InstanceID"));
String uuid = prop.getValue().toString();

```

VMFS Volumes Sharing a Directly Attached Disk

Figure 23 (below) shows the association of objects modeling two VMFS volumes and a SCSI disk they share, as well as the host system to which they belong. The `StoragePool` objects represent the VMFS volumes, while the `DiskPartition` objects represent the partitions on the LUN that implement the VMFS volumes. Both the partitions belong to the same LUN, which is represented by the `StorageExtent` object.

Figure 23



Typical Uses

If you want to access all VMFS volumes belonging to a host, use the `AssociatorNames` method with a reference to the `ESXComputerSystem` object and the class name of the `HostedStoragePool` association. This produces a list of `StoragePool` object references — one reference for each VMFS visible to the ESX Server host.

Example 18

```

Array<CIMObject> vmfs;
vmfs = client.associatorNames(NAMESPACE, theESXInstance.getPath(),
    "CIM_HostedStoragePool");

```

To determine total VMFS storage on a particular controller and LUN:

1. Use the `Associators` method with a reference to the `ESXComputerSystem` object and the class name "`CIM_SystemDevice`", also qualified by the result class name "`CIM_SCSIProtocolController`". This produces a list of `SCSIProtocolController` object references — one reference for each controller attached to the ESX Server host.

Example 19

```

Array<CIMObjectPath> ctrlr;
ctrlr = client.associators(NAMESPACE, theESXInstance.getPath(),
    "CIM_SystemDevice",
    "CIM_SCSIProtocolController");

```

2. Use the client's accessor method to get the `ElementName` properties of the `SCSIProtocolController` objects.

Example 20

```
Array<String> name;
for (int i = 0; i < ctrlr.size(); i++) {
    CIMProperty prop = ctrlr[i].getProperty(
        ctrlr[i].findProperty("ElementName"));
    name[i] = prop.getValue().toString();
}
```

3. Locate the desired controller in the list.

Example 21

```
CIMObject theControllerInstance = NULL;
for (int i = 0; i < name.size(); i++) {
    if (name [i] == userChoice) {
        theControllerInstance = ctrlr[i];
    }
}
```

4. Use the `AssociatorNames` method with a reference to the `SCSIProtocolController` object and the class name of the association parent class, `"CIM_ProtocolControllerAccessesUnit"`. This produces a list of `StorageExtent` object references — one reference for each LUN attached to the controller.

Example 22

```
Array<CIMObject> extent;
extent = client.associatorNames (NAMESPACE,
                                theControllerInstance.getPath(),
                                "CIM_ProtocolControllerAccessesUnit");
```

5. For each `StorageExtent` object, use the `GetProperty` method with the object reference and the property name `Name`. Each invocation produces a unit ID.

Example 23

```
Array<String> id;
for (int i = 0; i < extent.size(); i++) {
    CIMProperty prop = extent[i].getProperty(
        extent[i].findProperty("Name"));
    id[i] = prop.getValue().toString();
}
```

6. Locate the desired LUN in the list.

Example 24

```
CIMObject theLunInstance = NULL;
```

```

for (int i = 0; i < id.size(); i++) {
    if (id[i] == userChoice) {
        theLunInstance = extent[i];
    }
}

```

- Use the `Associators` method with a reference to the `StorageExtent` object and the class name of the `BasedOn` association (either the parent class or the derived class). This produces a list of `DiskPartition` objects — one for each partition of the LUN.

Example 25

```

Array<CIMObject> part;
part = client.associators(NAMESPACE,
                        theLunInstance.getPath(),
                        "CIM_BasedOn");

```

- For each `DiskPartition` object, use the `AssociatorNames` method with a reference to the `DiskPartition` object and the class name of the `ConcreteComponent` association (either the parent class or the derived class), also qualified by the result class name `"VMWARE_StoragePool"`. Discard all `DiskPartition` objects that are not associated with a `VMWARE_StoragePool` object.

Example 26

```

Array<CIMObject> vmfspart;
for (int i = 0; i < part.size(); i++) {
    Array<CIMObjectPath> pool;
    pool = client.associatorNames(NAMESPACE, part[i],
                                "CIM_ConcreteComponent",
                                "VMWARE_StoragePool");

    if (pool.size() > 0) {
        vmfspart.append(part[i]);
    }
}

```

- For each remaining `DiskPartition` object, use the client's `accessor` method to get the `NumberOfBlocks` property and the `BlockSize` property from the `DiskPartition` object. Multiply the two values to get the size of each partition.

Example 27

```

Array<uint64> partsize;
for (int i = 0; i < vmfspart.size(); i++) {
    CIMProperty prop = vmfspart[i].getProperty(
        vmfspart[i].findProperty("NumberOfBlocks"));
    partsize[i] = prop.getValue();
    prop = vmfspart[i].getProperty(

```

```
        vmfspart[i].findProperty("BlockSize"));  
    partsize[i] *= prop.getValue();  
}
```

10. Add the partition sizes to get the number of bytes of VMFS storage available to the controller on the chosen LUN.

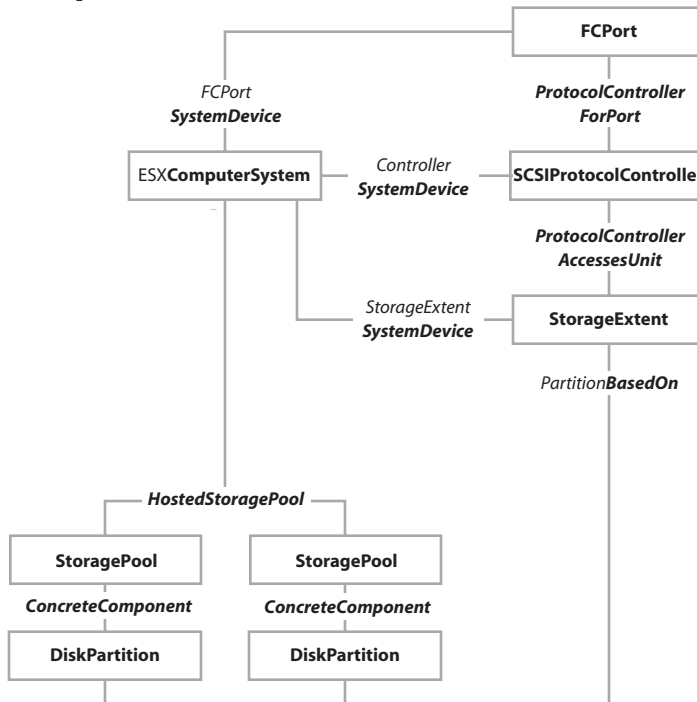
Example 28

```
uint64 bytesVmfsOnLun = 0;  
for (int i = 0; i < partsize.size(); i++) {  
    bytesVmfsOnLun += partsize[i];  
}
```

VMFS Volumes Sharing a Fibre Channel LUN

Figure 24 (below) shows the association of objects modeling two VMFS volumes and a Fibre Channel LUN they share, as well as the host system to which they belong. The only difference from the previous diagram (Figure 23 on page 60) is the presence of an FCPort object that models the Fibre Channel characteristics of the SCSI controller, including its WWPN.

Figure 24



Typical Uses

To discover the Fibre Channel port address of a SCSI controller for a particular VMFS volume — assuming you already have a reference to the StoragePool object — do the following steps:

1. Use the `AssociatorNames` method with a reference to the StoragePool object and the class name of the ConcreteComponent association (either the parent class or the derived class). This produces a reference to the associated DiskPartition object.

Example 29

```

Array<CIMObjectPath> part;
part = client.associatorNames(NAMESPACE,
                             theStoragePoolInstance.getPath(),
  
```

```

        "CIM_ConcreteComponent");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theDiskPartitionRef = part[0];

```

2. Use the `AssociatorNames` method with a reference to the `DiskPartition` object and the class name of the `PartitionBasedOn` association (either the parent class or the derived class). This produces a reference to the associated `StorageExtent` object (representing the LUN). Note that a VMFS volume may span more than one LUN in general. This example assumes a single LUN.

Example 30

```

Array<CIMObjectPath> extent;
extent = client.associatorNames(NAMESPACE, theDiskPartitionRef,
                               "CIM_PartitionBasedOn");
// Assuming a single LUN, not a spanning VMFS:
// Many-to-one mapping produces a 1-element array.
CIMObjectPath theLunRef = extent[0];

```

3. Use the `AssociatorNames` method with a reference to the `StorageExtent` object and the class name of the `ProtocolControllerAccessesUnit` association (either the parent class or the derived class). This produces a reference to the associated `SCSIProtocolController` object.

Example 31

```

Array<CIMObjectPath> ctrlr;
ctrlr = client.associatorNames(NAMESPACE, theLunRef,
                              "CIM_ProtocolControllerAccessesUnit");
// Many-to-one mapping produces a 1-element array.
CIMObjectPath theCtrlrRef = ctrlr[0];

```

4. Use the `Associators` method with a reference to the `SCSIProtocolController` object and the class name of the `ProtocolControllerForPort` association (either the parent class or the derived class). This produces a copy of the associated `FCPort` object.

Example 32

```

Array<CIMObject> fc;
fc = client.associators(NAMESPACE, theCtrlrRef,
                      "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObject theFCPortInstance = fc[0];

```

5. Use the client's `accessor` method to get the `PermanentAddress` property from the `FCPort` object.

Example 33

```

CIMProperty prop = theFCPortInstance.getProperty(
    theFCPortInstance.findProperty("PermanentAddress"));

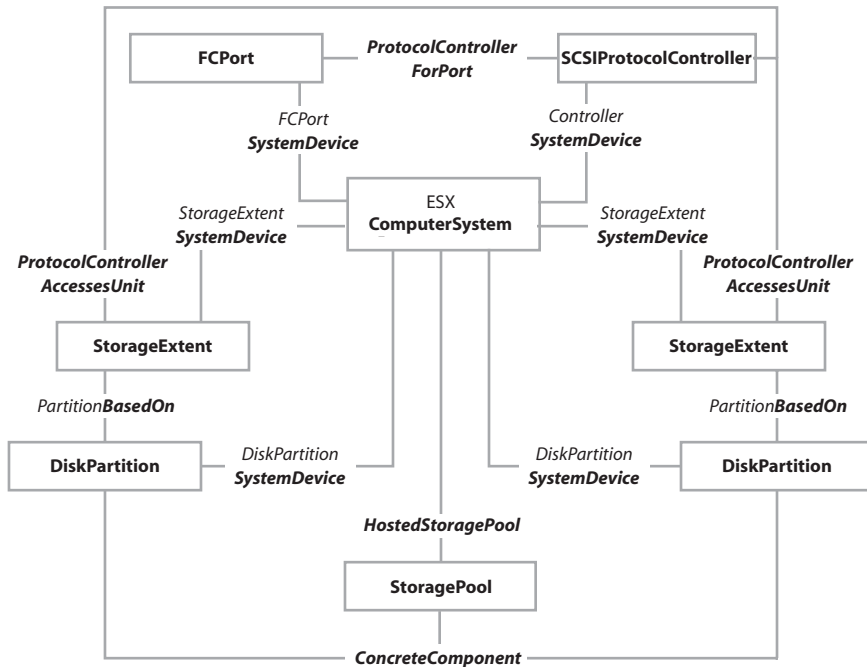
```

```
String theAddress = prop.getValue().toString();
```

VMFS Spanning Two LUNs

Figure 25 (below) shows a single VMFS volume that spans two LUNs. The StoragePool object represents the logical VMFS entity. Each DiskPartition object represents one of the two physical extents composing the VMFS. Each StorageExtent represents a LUN containing one of the two physical extents. In this case, the physical extents exist on different LUNs.

Figure 25



Typical Uses

To identify the LUNs composing a spanning VMFS volume:

1. Use the AssociatorNames method with a reference to the StoragePool object (representing the VMFS) and the class name of the ConcreteComponent association (either the parent class or the derived class). This produces a list of references to the DiskPartition objects.

Example 34

```
Array<CIMObjectPath> part;
part = client.associatorNames(NAMESPACE,
```

```
theStoragePoolInstance.getPath(),
"CIM_ConcreteComponent");
```

- For each DiskPartition object, use the Associators method with a reference to the DiskPartition object and the class name of the BasedOn association (either the parent class or the derived class). Each invocation produces a copy of the associated StorageExtent object (representing the LUN).

Example 35

```
Array<CIMObject> extentList;
for (int i = 0; i < part.size(); i++) {
    Array<CIMObject> extent;
    extent = client.associators(NAMESPACE, part[i], "CIM_BasedOn");
    extentList.appendArray(extent);
}
```

- For each StorageExtent object, use the client's accessor method to get the DeviceID property of the StorageExtent object.

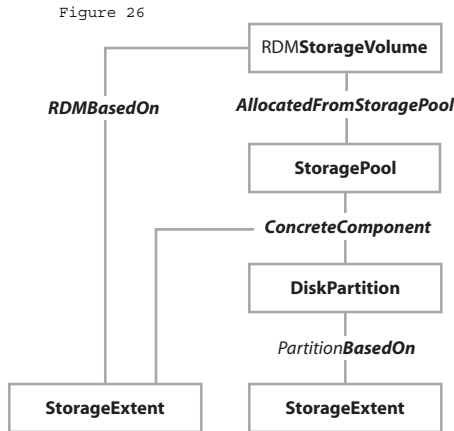
Example 36

```
Array<String> id;
for (int i = 0; i < extentList.size(); i++) {
    CIMProperty prop = extentList[i].getProperty(
        extentList[i].findProperty("DeviceID"));
    id[i] = prop.getValue().toString();
}
```

Raw Device Mapping

Figure 26 (below) shows the association of objects modeling a raw device mapping. VMWARE_RDMSStorageVolume, a subclass of CIM_StorageVolume, represents the logical storage available to the virtual machine by means of the mapping. (For a more complete picture of a raw device mapping environment, refer to Virtual Machine Using Raw Device Mapping in a Storage Array on page 39.) The mapping file maps the StorageExtent on the left, representing a raw LUN. The StorageExtent on the right represents the LUN that contains the VMFS on which the mapping file is stored. Note that the StoragePool, representing the VMFS, logically aggregates the storage

space belonging to the raw LUN; that relationship is represented by the ConcreteComponent association between the two objects.



Typical Uses

To determine the storage available to the virtual machine through the raw device mapping (assuming you already have a reference to the RDMStorageVolume object):

1. Use the `Associators` method with a reference to the `RDMStorageVolume` object and the class name of the `BasedOn` association (either the parent class or the derived class). This produces a copy of the `StorageExtent` object representing the raw LUN.

Example 37

```

Array<CIMObject> extent;
extent = client.associators(NAMESPACE, theRdmVolRef, "CIM_BasedOn");
// One-to-one mapping produces a 1-element array.
CIMObject theRdmLunInstance = extent[0];
  
```

2. Use the client's `accessor` method to get the `NumberOfBlocks` property and the `BlockSize` property of the `StorageExtent` object.

Example 38

```

CIMProperty prop = NULL;
prop = theRdmLunInstance.getProperty(
    theRdmLunInstance.findProperty("NumberOfBlocks"));
uint64 numBlocks = prop.getValue();
prop = theRdmLunInstance.getProperty(
    theRdmLunInstance.findProperty("BlockSize"));
uint64 blockSize = prop.getValue();
  
```

3. Multiply the `NumberOfBlocks` by the `BlockSize` to get the number of bytes available.

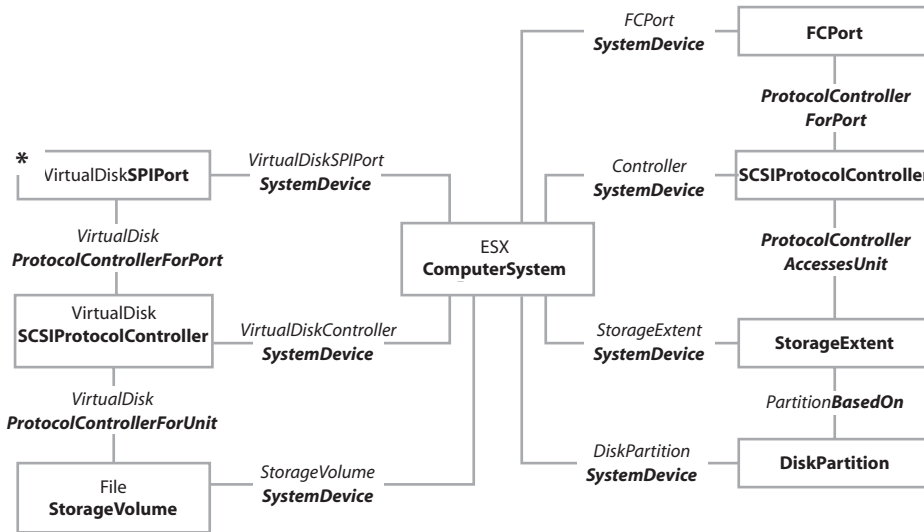
Example 39

```
uint64 rdmSizeBytes = numBlocks * blockSize;
```

System Devices of a Host

Figure 27 (below) shows the use of various subclasses of the `SystemDevice` association that link the host to its physical devices. These devices have been seen in the previous illustrations; however, other illustrations omit some of the `SystemDevice` associations.

Figure 27



Typical Uses

To identify all virtual disks visible to a host:

1. Use the `Associators` method with a reference to the `ESXComputerSystem` object and the class name `"CIM_SystemDevice"`, also qualified by the result class name `"CIM_StorageVolume"`. This produces a list of references to the associated `FileStorageVolume` objects, representing virtual disks.

Example 40

```
Array<CIMObject> vdisk;
vdisk = client.associators(NAMESPACE, theESXInstance.getPath(),
    "CIM_SystemDevice",
    "CIM_StorageVolume");
```

- For each FileStorageVolume object, use the client's accessor method to get the value of the `ElementName` property from the FileStorageVolume object.

Example 41

```
Array<String> name;
for (int i = 0; i < vdisk.size(); i++) {
    CIMProperty prop = vdisk[i].getProperty(
        vdisk[i].findProperty("ElementName"));
    name[i] = prop.getValue().toString();
}
```

To identify all Fibre Channel SCSI controllers available to the host:

- Use the `EnumerateInstances` method with the class name `"VMware_SCSIProtocolController"`. This produces a list of all the physical `SCSIProtocolController` objects, omitting the `VirtualDiskSCSIProtocolController` objects.

Example 42

```
Array<CIMInstance> ctrlr;
ctrlr = client.enumerateInstances(NAMESPACE,
    "VMWARE_SCSIProtocolController");
```

- For each `SCSIProtocolController` object, use the `AssociatorNames` method with a reference to the `SCSIProtocolController` object and the class name of the `ProtocolControllerForPort` association (either the parent class or the derived class), also qualified by the result class name of the `FCPort` association (either the parent class or the derived class). Discard all `SCSIProtocolController` objects that are not associated with an `FCPort` object.

Example 43

```
Array<CIMObject> fchba;
for (int i = 0; i < ctrlr.size(); i++) {
    Array<CIMObjectPath> fcport;
    fcport = client.associatorNames(NAMESPACE, ctrlr[i].getPath(),
        "CIM_ProtocolControllerForPort",
        "CIM_FCPort");

    if (fcport.size() > 0) {
        fchba.append(ctrlr[i]);
    }
}
```

- For each remaining `SCSIController` object, use the client's accessor method to get the `ElementName` property from the `SCSIController` object.

Example 44

```
Array<String> name;
for (int i = 0; i < fchba.size(); i++) {
```

```
CIMProperty prop = fchba[i].getProperty(  
    fchba[i].findProperty("ElementName"));  
name[i] = prop.getValue();  
}
```


4

CHAPTER

Sample Code

The following examples are included as starting points to help you assemble the basic building blocks presented in the chapter VMware CIM SDK Schema on page 17. These examples show how to connect to the Pegasus CIMOM on ESX Server and how to send and receive CIM operation requests and responses in the context of a simple utility program.

Several approaches to navigating the CIM SDK schema are outlined in different examples. Each example represents a small part of the code that would be present in a large application and is designed only to show the basic logic and programming constructs needed to use the VMware CIM SDK effectively.

All examples use the C++ programming language and the OpenPegasus CIM client library. Your choice of language and library may vary.

Additional samples, including other programming languages, can be found in the samples directory that accompanies this SDK.

All sample code that is provided as part of the VMware SDK is subject to certain restrictions, which you can find in the appendix Terms and Conditions on page 93. Please read the restrictions before using this sample code.

This chapter contains the following samples:

- Connecting to the Pegasus CIMOM on page 75
- Retrieving Information about an ESX Server Host on page 76
- Enumerating StorageExtent (LUN) Objects Starting from the ESXComputerSystem on page 78
- Listing Virtual Storage Available to Virtual Machines on page 81
- Enumerating Virtual Machines Starting from StoragePool (VMFS) Objects on page 85

Connecting to the Pegasus CIMOM

This example demonstrates the ability to connect to the Pegasus CIMOM.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>

using namespace Pegasus;
using namespace std;
const String NAMESPACE = "vmware/esxv2";
const int PORT = 5988;

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <hostname> <user> <password>"
             << endl;
        return (1);
    }
    // Establish a client-side connection object.
    CIMClient client;
    // Connect to the CIMOM.
    try {
        //      hostname, port, user, password
        client.connect(argv[1], PORT, argv[2], argv[3]);
    } catch (Exception &e) {
        cerr << "Error: Unable to connect: " << e.getMessage() << endl;
        return (1);
    }
    cout << "Connection established successfully." << endl;
    return (0);
}
```

Retrieving Information about an ESX Server Host

This example displays the name of the ESX Server host to which the client is connected.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>

using namespace Pegasus;
using namespace std;
const String NAMESPACE = "vmware/esxv2";
const int PORT = 5988;

CIMInstance GetESXServerObject(const CIMClient &client);
void PrintESXServerInfo(const CIMObject &theESXInstance);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <hostname> <user> <password>"
            << endl;
        return (1);
    }
    // Establish a client-side connection object.
    CIMClient client;
    // Connect to the CIMOM.
    try {
        // hostname, port, user, password
        client.connect(argv[1], PORT, argv[2], argv[3]);
    } catch (Exception &e) {
        cerr << "Error: Unable to connect: " << e.getMessage() << endl;
        return (1);
    }
    try {
        // Get ESXComputerSystem object.
        CIMInstance esx = GetESXServerObject(client);
        // Print info about ESX Server host.
        PrintESXServerInfo(esx);
    } catch (Exception &e) {
        cerr << "Error: CIM access failed: " << e.getMessage() << endl;
        return (1);
    }
    return (0);
}
```

```
}

CIMInstance GetESXServerObject(const CIMClient &client) {
    Array<CIMInstance> esxcs;
    esxcs = client.enumerateInstances(NAMESPACE,
                                     "VMWARE_ESXComputerSystem");
    // Can only get 1 ESXComputerSystem from each connection.
    CIMInstance theESXInstance = esxcs[0];
    return (theESXInstance);
}

void PrintESXServerInfo(const CIMObject &theESXInstance) {
    CIMProperty prop = theESXInstance.getProperty(
        theESXInstance.findProperty("ElementName"));
    String EsxName = prop.getValue().toString();
    cout << "ESX Server host: " << EsxName << endl;
}
```

Enumerating StorageExtent (LUN) Objects

Starting from the ESXComputerSystem

This example shows a simple way to locate the physical disks available to the ESX Server host. This way requires the client to specify a VMware extension schema class name. The LUNs are modeled by the StorageExtent objects linked by the StorageExtentSystemDevice associations.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>

using namespace Pegasus;
using namespace std;
const String NAMESPACE = "vmware/esxv2";
const int PORT = 5988;

CIMInstance GetESXServerObject(const CIMClient &client);
Array<CIMObject> GetLuns(const CIMClient &client,
                        const CIMObject &theESXInstance);
void PrintESXServerInfo(const CIMObject &theESXInstance);
void PrintLUNInfo(const CIMObject &theLUNObject);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <hostname> <user> <password>"
             << endl;
        return (1);
    }
    // Establish a client-side connection object.
    CIMClient client;
    // Connect to the CIMOM.
    try {
        // hostname, port, user, password
        client.connect(argv[1], PORT, argv[2], argv[3]);
    } catch (Exception &e) {
        cerr << "Error: Unable to connect: " << e.getMessage() << endl;
        return (1);
    }
    try {
        // Get ESXComputerSystem object.
        CIMInstance esx = GetESXServerObject(client);
        // Print info about ESX Server host.
```

```

    PrintESXServerInfo(esx);
    // Get LUN list.
    Array<CIMObject> luns = GetLuns(client, esx);
    // Print info about LUNs.
    for (int i = 0; i < luns.size(); i++) {
        PrintLUNInfo(luns[i]);
    }
} catch (Exception &e) {
    cerr << "Error: CIM access failed: " << e.getMessage() << endl;
    return (1);
}
return (0);
}

CIMInstance GetESXServerObject(const CIMClient &client) {
    Array<CIMInstance> esxcs;
    esxcs = client.enumerateInstances(NAMESPACE,
                                     "VMWARE_ESXComputerSystem");
    // Can only get 1 ESXComputerSystem from each connection.
    CIMInstance theESXInstance = esxcs[0];
    return (theESXInstance);
}

Array<CIMObject> GetLuns(const CIMClient &client,
                        const CIMObject &theESXInstance) {
    Array<CIMObject> vdisk;
    vdisk = client.associators(NAMESPACE, theESXInstance.getPath(),
                               "CIM_SystemDevice",
                               "CIM_StorageExtent");

    return vdisk;
}

void PrintESXServerInfo(const CIMObject &theESXInstance) {
    CIMProperty prop = theESXInstance.getProperty(
        theESXInstance.findProperty("ElementName"));
    String EsxName = prop.getValue().toString();
    cout << "ESX Server host: " << EsxName << endl;
}

void PrintLUNInfo(const CIMObject &theLUNObject) {
    CIMProperty prop = NULL;
    prop = theLUNObject.getProperty(
        theLUNObject.findProperty("NumberOfBlocks"));
    uint64 numBlocks = prop.getValue();
}

```

```
prop = theLUNObject.getProperty(  
    theLUNObject.findProperty("BlockSize"));  
uint64 blockSize = prop.getValue();  
uint64 LUNSizeBytes = numBlocks * blockSize;  
cout << " LUN: " << LUNName  
    << " (" << LUNSizeBytes << " bytes)" << endl;  
}
```

Listing Virtual Storage Available to Virtual Machines

This example lists the virtual disks belonging to each virtual machine on the host. It demonstrates how to follow a chain of objects from the ESXComputerSystem through the virtual machines, through their virtual disk controllers, and to the attached virtual disks.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>

using namespace Pegasus;
using namespace std;
const String NAMESPACE = "vmware/esxv2";
const int PORT = 5988;

CIMInstance GetESXServerObject(const CIMClient &client);
Array<CIMObject> GetVirtualMachines(const CIMClient &client,
                                   const CIMObject &theESXInstance);
void PrintVmInfo(const CIMClient &client,
                const CIMObject theVMInstance);
void FindAllVirtualDisks(const CIMClient &client,
                        const CIMObject theVMInstance);
void FindOneVirtualDisk(const CIMClient &client,
                       const CIMObjectPath &theVdEndpointRef);
void PrintVirtualDiskInfo(const CIMClient &client,
                         const CIMObject &theFSVolInstance);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <hostname> <user> <password>"
              << endl;
        return (1);
    }
    // Establish a client-side connection object.
    CIMClient client;
    // Connect to the CIMOM.
    try {
        //      hostname, port, user, password
        client.connect(argv[1], PORT, argv[2], argv[3]);
    } catch (Exception &e) {
        cerr << "Error: Unable to connect: " << e.getMessage() << endl;
    }
}
```

```

        return (1);
    }
    try {
        // Get ESXComputerSystem object.
        CIMInstance esx = GetESXServerObject(client);
        // Print info about ESX Server host.
        PrintESXServerInfo(client, esx);
        // Get VMComputerSystem objects.
        Array<CIMObject> vmcs = GetVirtualMachines(client, esx);
        // For each virtual machine, print info and find its virtual disks.
        for (int i = 0; i < vmcs.size(); i++) {
            PrintVmInfo(client, vmcs[i]);
            FindAllVirtualDisks(client, vmcs[i]);
        }
    } catch (Exception &e) {
        cerr << "Error: CIM access failed: " << e.getMessage() << endl;
        return (1);
    }
    return (0);
}

CIMInstance GetESXServerObject(const CIMClient &client) {
    Array<CIMInstance> esxcs;
    esxcs = client.enumerateInstances(NAMESPACE,
                                     "VMWARE_ESXComputerSystem");
    // Can only get 1 ESXComputerSystem from each connection.
    CIMInstance theESXInstance = esxcs[0];
    return (theESXInstance);
}

Array<CIMObject> GetVirtualMachines(const CIMClient &client,
                                   const CIMObject &theESXInstance) {
    Array<CIMObject> vmcs;
    int VMcount = 0;
    vmcs = client.associators(NAMESPACE, theESXInstance.getPath(),
                             "VMWARE_HostedDependency");
    return (vmcs);
}

void PrintVmInfo(const CIMClient &client,
                 const CIMObject theVMInstance) {
    CIMProperty prop = theVMInstance.getProperty(
        theVMInstance.findProperty("ElementName"));
    cout << <Virtual Machine: " << prop.getValue() << endl;
}

```

```

);

void FindAllVirtualDisks(const CIMclient &client,
                       const CIMObject theVMInstance) {
    // Navigate to its virtual HBAs.
    Array<CIMObjectPath> vctrlr;
    vctrlr = client.associatorNames(NAMESPACE, theVMInstance.getPath(),
                                   "VMWARE_VirtualInitiatorControllerSystemDevice");
    // For each virtual controller, navigate to its virtual disks.
    for (int i = 0; i < vctrlr.size(); i++) {
        Array<CIMObjectPath> viport;
        viport = client.associatorNames(NAMESPACE, vctrlr[i],
                                       "CIM_ProtocolControllerForPort");
        // One-to-one mapping produces a 1-element array.
        CIMObjectPath theViPortRef = viport[0];
        Array<CIMObjectPath> viendpt;
        viendpt = client.associatorNames(NAMESPACE, theViPortRef,
                                       "CIM_DevicesAPIImplementation");
        // One-to-one mapping produces a 1-element array.
        CIMObjectPath theViEndpointRef = viendpt[0];
        Array<CIMObjectPath> vdendpt;
        vdendpt = client.associatorNames(NAMESPACE, theViEndpointRef,
                                       "CIM_ActiveConnection");
        // For each virtual disk, navigate to the FileStorageVolume object.
        for (int j = 0; j < vdendpt.size(); j++) {
            FindOneVirtualDisk(client, vdendpt[j]);
        }
    }
}
}
}

```

```

void FindOneVirtualDisk(const CIMClient &client,
                       const CIMObjectPath &theVdEndpointRef) {
    Array<CIMObjectPath> vdport;
    vdport = client.associatorNames(NAMESPACE, theVdEndpointRef,
                                   "CIM_DevicesAPIImplementation");
    // One-to-one mapping produces a 1-element array.
    CIMObjectPath theVdPortRef = vdport[0];
    Array<CIMObjectPath> vdctrlr;
    vdctrlr = client.associatorNames(NAMESPACE, theVdPortRef,
                                   "CIM_ProtocolControllerForPort");
    // One-to-one mapping produces a 1-element array.
    CIMObjectPath theVdCtrlrRef = vdctrlr[0];
    Array<CIMObject> fsvol;
    fsvol = client.associators(NAMESPACE, theVdCtrlrRef,

```

```
        "CIM_ProtocolControllerForUnit");
    // One-to-one mapping produces a 1-element array.
    CIMObject theFSVolInstance = fsvol[0];
    PrintVirtualDiskInfo(client, theFSVolInstance);
}

void PrintVirtualDiskInfo(const CIMClient &client,
    const CIMObject &theFSVolInstance) {
    CIMProperty prop;
    prop = theFSVolInstance.getProperty(
        theFSVolInstance.findProperty("NumberOfBlocks"));
    uint64 numBlocks = prop.getValue();
    prop = theFSVolInstance.getProperty(
        theFSVolInstance.findProperty("BlockSize"));
    uint64 blockSize = prop.getValue();
    uint64 vdSizeBytes = numBlocks * blockSize;
    cout << " Virtual disk: " << vdSizeBytes << " bytes." << endl;
}
```

Enumerating Virtual Machines Starting from StoragePool (VMFS) Objects

This example is largely the reverse of the previous example. This example starts from the storage pools and follows a chain of objects through the virtual disks stored in the pools, through the virtual controllers, to the virtual machines. The StoragePool object models a VMFS volume on an ESX Server host.

```
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>

using namespace Pegasus;
using namespace std;
const String NAMESPACE = "vmware/esxv2";
const int PORT = 5988;

Array<CIMInstance> GetStoragePools(const CIMClient &client);
void PrintVmfsInfo(const CIMClient &client,
                  const CIMObject theVmfsInstance);
CIMObject FindVirtualMachine(const CIMClient &client,
                             const CIMObject &theDiskInstance) {
void RemoveDupVms(Array<CIMObject> &vmlist);
void PrintVmInfo(const CIMClient &client,
                 const CIMObject theVMInstance);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <hostname> <user> <password>"
             << endl;
        return (1);
    }
    // Establish a client-side connection object.
    CIMClient client;
    // Connect to the CIMOM.
    try {
        //      hostname, port, user, password
        client.connect(argv[1], PORT, argv[2], argv[3]);
    } catch (Exception &e) {
        cerr << "Error: Unable to connect: " << e.getMessage() << endl;
        return (1);
    }
}
```

```

try {
    Array<CIMObject> listVirtualMachines;
    // Enumerate all StoragePool (VMFS) objects.
    Array<CIMInstance> pool = GetStoragePools(client);
    // For each VMFS, print info.
    for (int i = 0; i < pool.size(); i++) {
        PrintVmfsInfo(client, pool[i]);
        // Navigate to VMWARE_FileStorageVolume objects (virtual disks).
        Array<CIMObject> disk = GetStorageVolumes(client, pool[i]);
        // For each virtual disk, find its virtual machine.
        for (int j = 0; j < disk.size(); j++) {
            CIMObject vm = FindVirtualMachine(client, disk[j]);
            listVirtualMachines.append(vm);
        }
        RemoveDupVms(listVirtualMachines);
        for (int k = 0; k < listVirtualMachines.size(); k++) {
            PrintVmInfo(client, listVirtualMachines[k]);
        }
    }
} catch (Exception &e) {
    cerr << "Error: CIM access failed: " << e.getMessage() << endl;
    return (1);
}
return (0);
}

Array<CIMInstance> GetStoragePools(const CIMClient &client) {
    Array<CIMInstance> pool;
    pool = client.enumerateInstances(NAMESPACE,
                                    "CIM_StoragePool");
    return (pool);
}

void PrintVmfsInfo(const CIMClient &client,
                  const CIMObject theVmfsInstance) {
    CIMProperty prop = theVmfsInstance.getProperty(
        theVmfsInstance.findProperty("ElementName"));
    cout << "VMFS: " << prop.getValue() << endl;
};

CIMObject FindVirtualMachine(const CIMClient &client,
                             const CIMObject &theDiskInstance) {
    // Navigate to VirtualDiskProtocolEndpoint.
    Array<CIMObjectPath> vdctlr;

```

```

vdctlr = client.associatorNames(NAMESPACE, theDiskInstance.getPath(),
                               "CIM_ProtocolControllerForUnit");
// Many-to-one mapping produces a 1-element array.
CIMObjectPath theVdCtlrRef = vdctlr[0];

Array<CIMObjectPath> vdport;
vdport = client.associatorNames(NAMESPACE, theVdCtlrRef,
                               "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdPortRef = vdport[0];

Array<CIMObjectPath> vdendpt;
vdendpt = client.associatorNames(NAMESPACE, theVdPortRef,
                                 "CIM_DevicesAPIImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdEndptRef = vdendpt[0];

Array<CIMObjectPath> viendpt;
viendpt = client.associatorNames(NAMESPACE, theVdEndptRef,
                                 "CIM_ActiveConnection");
// Many-to-one mapping produces a 1-element array.
CIMObjectPath theViEndptRef = viendpt[0];

Array<CIMObjectPath> viport;
viport = client.associatorNames(NAMESPACE, theViEndptRef,
                               "CIM_DevicesAPIImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViPortRef = viport[0];

Array<CIMObjectPath> victlr;
victlr = client.associatorNames(NAMESPACE, theViPortRef,
                               "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViCtlrRef = victlr[0];

Array<CIMObject> vmcs;
vmcs = client.associator(NAMESPACE, theViCtlrRef,
                        "CIM_ControllerSystemDevice");
// Many-to-one mapping produces a 1-element array.
CIMObject theVmcsInstance = vmcs[0];
return (theVmcsInstance);
}

void RemoveDupVms(Array<CIMObject> &vmlist) {

```

```

// Compare UUIDs and reject duplicates.
for (int i = 0; i < vmlist.size(); i++) {
    CIMProperty propi = vmlist[i].getProperty(
        vmlist[i].findProperty("Name"));
    CIMValue vali = propi.getValue();
    int j = i+1;
    while (j < vmlist.size()) {
        CIMProperty propj = vmlist[j].getProperty(
            vmlist[j].findProperty("Name"));
        CIMValue valj = propj.getValue();
        if (valj == vali) {
            vmlist.remove(j);
        } else {
            j++;
        }
    }
}

void PrintVmInfo(const CIMclient &client,
                const CIMObject theVMInstance) {
    CIMProperty prop = theVMInstance.getProperty(
        theVMInstance.findProperty("ElementName"));
    cout << <Virtual Machine: " << prop.getValue() << endl;
};

```

Glossary

API

Application programming interface. A specified set of functions that allows one to access a module or service programmatically.

CIM

Common Information Model (CIM). The CIM conceptual information model for describing management that is not bound to a particular implementation.

CIMOM

Common Information Model Object Manager. A CIMOM stores class definitions and populates requests for them with information returned from specific data providers.

DMTF

Distributed Management Task Force. The organization responsible for the CIM standard.

HBA

Host bus adapter; can be either physical (attached to a host) or virtual (part of a virtual machine).

Host machine

A machine with ESX Server installed, capable of hosting virtual machines.

IBVP

In-band Virtualizer Profile. The standard profile that the CIM SDK adapts to describe ESX Server.

IDL

Interface definition language. A human-readable syntax used to specify an API. The IDL may be compiled into stubs on a client machine. See Stub.

ISV

Independent software vendor — Systems management vendors, enterprise management frameworks, imaging and provisioning vendors, storage management vendors, and so on.

LUN

Logical Unit Number. A disk volume in a storage array.

MOF

Managed Object Format. File format for the CIM IDL describing model classes.

Pegasus

OpenPegasus is an open-source implementation of the DMTF CIM and WBEM standards. It is the CIMOM used by ESX Server. Pegasus also provides a client library used to write CIM client applications.

RDM

Raw device mapping.

SDK

Software developer kit. It comprises some or all of: an API, an IDL, client stubs, sample code, and manuals.

Server

A server is a system capable of managing and executing virtual machines.

SMI-S

Storage Management Initiative-Specification. A standard containing the CIM profile (IBVP) used by the CIM SDK.

SNIA

Storage Networks Industry Association. Organization responsible for the SMI-S profiles.

Storage virtualizer

A system that abstracts and aggregates physical storage on behalf of a storage-using client.

Stub

A procedure that implements the client side of a remote procedure call. The client calls the stub to perform a task, and the stub then transmits parameters over the network to the server and returns the results to the client.

UUID

Universally Unique Identifier (ID). This is a 128-bit number represented in hexadecimal (HEX) format when passed as a string; for example, `f81d4fae-7dec-11d0-a765-00a0c91e6bf6`.

Virtual disk

A virtual disk is a file or set of files that appear as a physical disk drive to a guest operating system. These files can be on the host machine or on a remote file system.

Virtual machine

A virtual machine contains system and configuration states for a machine and may execute on a host machine.

VMFS

See VMware ESX Server file system.

VMware ESX Server file system

A file system that is optimized for storing virtual machines. Also referred to as VMFS.

One VMFS partition is supported per SCSI storage device or SAN. Each version of ESX Server uses a corresponding version of VMFS. For example, VMFS3 was introduced with ESX Server 3.

XML

Extensible Markup Language, a text-based markup language that is especially designed for Web documents.

Terms and Conditions

VMware® Software Developer Kit (SDK) Agreement

VMware, Inc. ("VMware") provides this Software Developer Kit ("SDK") to you subject to the following terms and conditions. If you disagree with any of the following terms, then do not use this SDK.

1. This SDK contains a variety of materials, including but not limited to, interface definitions, documentation, and sample code regarding programming interfaces to one or more VMware products as referenced in such materials ("VMware Software"). This SDK is intended to serve as a guide for writing programs to interact with the VMware Software.
2. Subject to the restrictions below, you may download and make a reasonable number of copies of the SDK contents for your personal use solely for the purpose of creating software that communicates with VMware Software ("Developer Software"). You agree to defend, indemnify and hold harmless VMware, and any of its directors, officers, employees, affiliates or agents, from and against any and all claims, losses, damages, liabilities and other expenses (including

reasonable attorneys' fees), arising from your modification and distribution of the sample code or breach of this SDK Terms and Conditions.

3. Restrictions: You may not (1) use the SDK to design or develop anything other than Developer Software; (2) make any more copies of the SDK than are reasonably necessary for the authorized use and backup and archival purposes; (3) modify, create derivative works of, reverse engineer, reverse compile, or disassemble the SDK, except that you may modify and create derivative works of the sample code and distribute the modified sample code in connection with Developer Software; (4) distribute, sell, lease, rent, lend, or sublicense any part of the SDK to any third party except as designated herein and as necessary to distribute Developer Software; (5) use the SDK to design or develop software to upload or otherwise transmit any material containing software viruses or other computer code, files or programs designed to interrupt, destroy, or limit the functionality of any software or hardware.

4. VMware retains ownership of the SDK, including without limitation all copyrights and other intellectual property rights therein.

5. You may not represent that the programs you develop using the SDK are certified or otherwise endorsed by VMware. You may not use the VMware name or any other trademarks or service marks of VMware in connection with programs that you develop using the SDK.

6. You will not receive any VMware support or subscription services for the SDK or any other services from VMware in connection with the SDK. If you have purchased support and/or subscription services for a VMware product, such support and/or subscription services shall not apply to the SDK or your use of the SDK.

7. Term, Termination and Changes: This Agreement shall continue as long as you are in compliance with the terms specified herein or until otherwise terminated. You and or VMware each may terminate this Agreement for any reason at any time. You agree, upon termination, to destroy all copies of the SDK within your possession or control. The Confidential Information, Limitations of Warranties, Liability and Indemnification sections set out in this Agreement shall survive any termination or expiration of this Agreement.

8. Limitations of Warranties and Liability: THE SDK IS PROVIDED "AS IS" WITHOUT ANY WARRANTIES OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, VMWARE

DISCLAIMS ANY IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL VMWARE BE LIABLE FOR ANY LOST PROFITS OR BUSINESS OPPORTUNITIES, LOSS OF USE, BUSINESS INTERRUPTION, LOSS OF DATA, OR ANY OTHER INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE SDK OR YOUR USE OF THE SDK, UNDER ANY THEORY OF LIABILITY, WHETHER BASED IN CONTRACT, TORT, NEGLIGENCE, PRODUCT LIABILITY, OR OTHERWISE. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE PRECEDING LIMITATION MAY NOT APPLY TO YOU.

VMWARE'S LIABILITY ARISING OUT OF THE SDK PROVIDED HEREUNDER WILL NOT, IN ANY EVENT, EXCEED US\$5.00.

THE FOREGOING LIMITATIONS SHALL APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, REGARDLESS OF WHETHER VMWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND REGARDLESS OF WHETHER ANY REMEDY FAILS OF ITS ESSENTIAL PURPOSE.

9. These terms are governed by the laws of the State of California and the United States of America without regard to conflict of laws principles. You may not assign any part of this Agreement without the prior written consent of VMware. Any attempted assignment without consent shall be void. These terms constitute the entire agreement between you and VMware with respect to the SDK, and supersede all prior written or oral communications, understandings and agreements. Any waiver of these terms must be in writing to be effective. If any provision of these terms is found to be invalid or unenforceable, the remaining terms will continue to be valid and enforceable to the fullest extent permitted by law.

Revision History

The following table lists the revision history for the *CIM SDK Programming Guide*.

Date	Description
June 15, 2006	General availability of CIM SDK 2.0 Programming Guide.

Index

A

associations
traversing **29**
audience **8**
authentication **16**

B

"Block Server" property value
51, 52
BlockSize property **57, 58, 62, 68, 69**

C

CIM schema
DMTF **19**
VMware **23, 27, 51**
CIM_ prefix **27**
"CIM_ProtocolController
AccessesUnit" parameter value
61
"CIM_SCSIProtocolContro
ller" parameter value **54, 60**
"CIM_StorageVolume" parame
ter value **69**
"CIM_SystemDevice" parameter
value **54, 60, 69**

CIMOM **12, 13, 14, 16**

client **13, 15, 16**

conventions
illustrations **28**
text **27**

D

Dedicated property **51, 52**

DeviceID property **67**

disks
physical **36, 59, 64**
virtual **31, 33, 36, 55, 58**

Distributed Management Task Force **7**

DMTF CIM schema **19**

document set **7**

E

ElementName property **61, 70**

F

features **6**
Fibre Channel **64**
Fibre Channel SCSI controllers **70**
firewall **15**
flat virtual disks **58**

H

host bus adapters, virtual **53, 54, 55, 56**
hosts **52**
HTTP **13, 14, 15, 16**

I

IBVP **6**
industry standards
relationship of VMware CIM SDK **19**
SMI-S **20**
InstanceID property **59**

L

LUNs **36, 64, 66**
shared **42**

M

MOF **12**
multipath SAN environment **47**

N

Name property **61**
namespace **15, 16**
new features **6**
NumberOfBlocks property **57, 58, 62, 68, 69**

O

OpenPegasus **7, 14**

P

Pegasus **7, 15**
PermanentAddress property **65**
port numbers **16**

R

raw device mapping **39, 67, 68**
resources, technical support **7**

S

SAN, multipathing **47**
shared LUNs **42**
SMI-S **7**
SMI-S profile **20**
snapshots **33**
SNIA **7**
SpaceConsumed property **58, 59**
spanning VMFS volume **66**
sparse virtual disks **58**
storage arrays **36, 39, 47**
system environments **31**
SystemDevice associations **69**

T

technical support resources **7**
traversing associations **29**

U

UUID **59**

V

virtual disks **31, 33, 36, 55, 56, 58, 69**
virtual host bus adapters **53, 54, 55, 56**
virtual machines **31, 33, 36, 39, 52, 53**
VMFS **31, 58, 59, 64, 66**
VMFS storage size **60**
VMFS volume, spanning **66**
VMware CIM schema **23, 51**
VMWARE_ prefix **27**
"VMWARE_HostedDependenc
y" parameter value **53**
"VMware_SCSIProtocolCon
troller" parameter value **70**
"VMWARE_StoragePool"
parameter value **62**

W

WBEM **7, 15**

X

XML **13**