CIM Storage Management API Programming Guide

VMware ESX/ESXi 4.0
VMware vCenter 4.0
You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com
Virtual Machines on a Host 40
   Typical Uses 41
Virtual Host Bus Adapters on a Virtual Machine 42
   Typical Uses 42
Virtual Disks Connected to a Virtual Host Bus Adapter 43
   Typical Uses 44
Virtual Disks on a VMFS 45
   Typical Uses 46
VMFS Volumes Sharing a Directly Attached Disk 47
   Typical Uses 47
VMFS Volumes Sharing a Fibre Channel LUN 49
   Typical Uses 50
VMFS Spanning Two LUNs 51
   Typical Uses 52
Raw Device Mapping 53
   Typical Uses 53
System Devices of a Host 54
   Typical Uses 54

4 Sample Code 57
   Connecting to the Pegasus CIMOM 58
   Retrieving Information about an ESX/ESXi Host 59
   Enumerating StorageExtent (LUN) Objects Starting from the ESXComputerSystem 60
   Listing Virtual Storage Available to Virtual Machines 61
   Enumerating Virtual Machines Starting from StoragePool (VMFS) Objects 63

A Terms and Conditions 67
   VMware® Software Developer Kit (SDK) Agreement 67

B CIM Server Setup Utility 69
   Running the Setup Script 69
   Command Line Options 69
   Usage Example 70

C Advanced Server Configuration Settings 71
   CIM Server Configuration Files 71
   Pegasus Configuration File 71
   VMware Configuration File 72
   Adjusting the CIM Server Cache Refresh Interval 73
   Adjusting the HTTP Response Buffer Size for vCenter Server 74

Glossary 75

Index 79
About This Book

The CIM Storage Management API Programming Guide provides information about developing applications using the CIM Storage Management API for VMware® ESX/ESXi 4.0 and vCenter 4.0.

VMware provides many different APIs and SDKs for various applications and goals. This book provides information about the VMware CIM Storage Management API for developers who are interested in using CIM to manage the storage resources available to ESX/ESXi machines. The CIM Storage Management API implements the Storage Management Initiative Specification (SMI-S), an industry standard for managing server storage. This book describes the schema implemented by VMware and contains suggestions for using the Common Information Model (CIM) classes to accomplish common use cases.

To view the current version of this book and all VMware API and SDK documentation, go to http://www.vmware.com/support/pubs/sdk_pubs.html.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this book.

Table 1. Revision History

<table>
<thead>
<tr>
<th>Revision Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20090507</td>
<td>Change API name (to accommodate the addition of the CIM SMASH/Server Management API) from CIM SDK to CIM Storage Management API. Change product name from ESX Server to ESX for ESX 4.0 release. Update product names for vSphere 4 release. Add information about the CIM management appliance in ESX/ESXi 4.0. Add appendixes for advanced configuration information. Add WS-Man connection information.</td>
</tr>
<tr>
<td>20060615</td>
<td>General availability of CIM SDK 2.0 Programming Guide.</td>
</tr>
</tbody>
</table>

Intended Audience

This book is intended for anyone who needs to develop applications using the CIM Storage Management API. Users typically include software developers who are creating applications that need to manage virtual storage with interfaces based on CIM standards.

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

- The practical application of CIM concepts and principles.
- The SMI-S schema profile.
- Developing system administration and system monitoring applications.
- Operating and managing VMware vSphere environments.
Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to
docfeedback@vmware.com.

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current versions
of other VMware books, go to http://www.vmware.com/support/pubs.

Online and Telephone Support

To use online support to submit technical support requests, view your product and contract information, and
register your products, go to http://www.vmware.com/support.

Support Offerings

To find out how VMware support offerings can help meet your business needs, go to
http://www.vmware.com/support/services.

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials
designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live
online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides
offerings to help you assess, plan, build, and manage your virtual environment. To access information about
education classes, certification programs, and consulting services, go to http://www.vmware.com/services.
The VMware CIM Storage Management API provides a CIM-compliant object model for virtual machines and their related storage devices. The VMware model supports the standard SMI-S schema. The standard SMI-S schema 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 Storage Management API, developers can do the following:

- Explore the virtual machines on ESX/ESXi hosts 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.

**IMPORTANT** Some parts of the CIM Storage Management API schema are experimental. The interface might change in future releases to align more closely with evolving standards. For more detail on this topic, see “Exceptions to Standard Schema” on page 22.

This chapter contains the following sections:

- “What Is New in This Release” on page 7
- “Recommended Documentation” on page 7
- “Using This Manual” on page 8

**What Is New in This Release**

This release of the CIM Storage Management API features the following:

- New support for ESX/ESXi 4.0 and CIM 2.17
- Support for VMware vCenter
- A management appliance for improved remote management

**Recommended Documentation**

VMware recommends the following documentation for information related to the CIM Storage Management API.
CIM Storage Management API Document Set

The following documents are provided with the CIM Storage Management API:

- **CIM Storage Management API Reference**
  The API reference is a set of HTML files describing in detail the objects and associations used by the VMware CIM Storage Management API schema. You can get the API reference when you download the CIM Storage Management SDK.

- **CIM Storage Management API 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 Resources for CIM Users

The VMware CIM Storage Management API documentation assumes that you are familiar with VMware vSphere products, CIM applications, and the SNIA (Storage Networks Industry Association) SMI-S (Storage Management Initiative Specification) profiles. See the following Web sites for additional information:

- Distributed Management Task Force — [http://www.dmtf.org](http://www.dmtf.org)
- Common Information Model — [http://www.dmtf.org/standards/cim/](http://www.dmtf.org/standards/cim/)
- OpenPegasus — [http://www.openpegasus.org](http://www.openpegasus.org)
- SNIA — [http://www.snia.org/home](http://www.snia.org/home)

Using This Manual

The purpose of this programming guide is to familiarize you with the logical structures used in the VMware CIM Storage Management API, the VMware schema extension, and the components you need to explore your ESX/ESXi machines that use CIM-enabled clients.

The **CIM Storage Management API Programming Guide** contains the following information:

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

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

Content by Chapter

This **CIM Storage Management API Programming Guide** is organized into the following chapters:

1. **Chapter 1, “Introducing the CIM Storage Management API Programming Guide,”** on page 7
   Chapter 1 is a brief overview of the CIM Storage Management API and an introduction to the programming guide.

2. **Chapter 2, “Working with the 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) in the management appliance, and how to prepare your environment for application development.

3. **Chapter 3, “VMware CIM Schema,”** on page 19
Chapter 3 gives an in-depth description of the CIM Storage Management API schema extension. The chapter describes how to work with the CIM data model that represents ESX/ESXi, virtual machines, and their storage resources.

4 Chapter 4, “Sample Code,” on page 57

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 Appendix A, “Terms and Conditions,” on page 67

This appendix contains the licensing terms that govern use of the samples.

6 Appendix B, “CIM Server Setup Utility,” on page 69

This appendix describes the configuration options available in the CIM server setup utility. The appendix also describes how to run the setup utility using command-line options rather than prompting the user for input.

7 Appendix C, “Advanced Server Configuration Settings,” on page 71

This appendix discusses additional configuration settings that apply to the CIMOM interaction with a datacenter server. This appendix is useful for understanding settings that can be changed in atypical environments or when debugging problems with your installation.

8 “Glossary” on page 75

The glossary contains definitions of terms related to the CIM Storage Management API.

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 in Chapter 3, “Introduction to the VMware CIM Storage Management API Schema,” on page 24 explaining the conventions used in the manual.

4 Browse the system environments in Chapter 3, “Sample System Environments,” on page 26 to select one or more that are similar to your ESX/ESXi environment.

5 Select and study one or more of the object relationships in Chapter 3, “Using the CIM Storage Management API Schema,” on page 40 that apply to your ESX/ESXi environment.


7 Browse the sample code in Chapter 4, “Sample Code,” on page 57 to choose an approach for your client code. Be sure to familiarize yourself with the terms of use in Appendix A, “Terms and Conditions,” on page 67.

8 See glossary terms (“Glossary” on page 75) as needed.
This chapter describes installation and configuration steps you must know before you can use the CIM Storage Management API development environment. For some installations, you might also need to consult Appendix C, “Advanced Server Configuration Settings,” on page 71.

This chapter contains the following sections:

- “VMware CIM Storage Management API Components” on page 11
- “Setting Up Your Development Environment” on page 13
- “Starting the CIM Servers Manually” on page 17
- “Other Configuration Settings” on page 17
- “Connecting Your Client to the CIMOM” on page 17

For more detail on using the object model, see “VMware CIM Schema” on page 19.

For code samples for basic operations, see “Sample Code” on page 57.

### VMware CIM Storage Management API Components

The SDK files are provided for download in a zip package. All of these components can be downloaded from the VMware Web site. To download the package, refer to [http://www.vmware.com/support/developer](http://www.vmware.com/support/developer).

VMware provides the following components to develop clients for the CIM Storage Management API:

- **CIM Storage Management API Programming Guide** to provide developers with an introduction to the VMware CIM schema extension and usage examples
  
- Management appliance containing:
  
  - Pegasus CIMOM to support CIM-XML requests from CIM clients
  
  - WS-Management server to support WS-Management requests from CIM clients
  
  - SMI-S providers to allow clients to explore virtual machines and their allocated resources on ESX/ESXi hosts

- SDK containing:
  
  - MOF (Managed Object Format) files specifying the CIM Storage Management API schema
  
  - **CIM Storage Management API Reference** to provide developers with an in-depth resource about the classes in the VMware CIM schema extension
  
  - Sample clients in several languages that demonstrate CIM Storage Management API functionality
CIM Storage Management API Processes

Figure 2-1 provides an overview of the software components in a running CIM Storage Management API environment.

The CIM client might be any management application that complies with the SMI-S profile. The CIM client can run on any operating system, so long as a client library is available to translate client method invocations into CIM messages.

The CIM Storage Management API supports both CIM-XML and WS-Management communication. See “Configuring the Firewall” on page 15 for information about supported TCP ports.

The CIMOM is a part of the management appliance. The management appliance can run on any VMware platform product, including ESX/ESXi. The CIMOM in turn uses the VMware vSphere API to communicate with the vCenter server or with an individual ESX/ESXi host.

If the appliance is configured to connect directly to a single ESX/ESXi host, the appliance can provide management information only for the virtual machines and storage resources on that host. If the appliance is configured to connect to a vCenter server, the appliance can provide management information for all virtual machines and storage resources managed by the vCenter application.

A CIM client can open separate connections to different appliances, but the CIMOM in each appliance presents its own view of virtual machines and storage resources detected by the ESX/ESXi host or vCenter server to which it connects. To use separate connections to different appliances simultaneously, the client must maintain separate connection contexts and collate or aggregate the information received from multiple CIMOMs as needed.

Client Libraries

Due to the platform and language independence of the CIM standard, you can choose from a number of CIM client libraries 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 must have HTTP or HTTPS access from the host on which it runs to the appropriate port on the management appliance.
VMware does not provide a client library. Open source CIM toolkits for client development include the following:

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

**Setting Up Your Development Environment**

To begin using the VMware CIM Storage Management API to manage your vSphere datacenter, you must first set up the management appliance containing the CIM server.

**Installing the Management Appliance**

The appliance is available as a virtual machine in OVF format. You can download the appliance from the VMware Web site.

You can run the appliance virtual machine on any VMware platform product, including VMware Workstation, VMware Server, or VMware ESX/ESXi. The first time the appliance starts up, it runs a script to guide you through initializing the guest operating system. Before you can use the appliance with CIM clients, you must use the setup script described below to configure the appliance for use by the CIM server.

Configuring the appliance for CIM operations involves three steps:

- Binding the appliance to a server in your datacenter
- Setting the appliance memory size
- Setting the number of virtual processors
- Configuring the firewall
- (optional) Installing certificates
- (optional) Configuring the CIM server to start automatically

Configuration requires root privileges in the guest operating system of the appliance. For increased security, the root user is disabled in the appliance. However, the user `vi-admin` is enabled to use the `sudo` utility, which allows you to do the configuration with root privileges.

To configure the appliance, run the setup utility:

```
sudo /etc/vmware/vmware-smis-setup.pl
```

You can choose to bind the appliance in either of two ways:

- To a vCenter server. When bound to a vCenter server, the appliance presents a CIM client with a collection of objects representing the storage and virtualization resources that belong to all the ESX/ESXi hosts managed by the vCenter server.
- To a single ESX/ESXi host. When bound to a single host, the appliance presents a CIM client with only the objects that represent the storage and virtualization resources belonging to that host.

When the setup utility binds the appliance to a server, it also asks you for a username and password, which it saves in an encrypted credential store. The username and password allow you to authenticate on the vCenter server or ESX/ESXi host.

For more detailed information about the setup utility, see Appendix B, “CIM Server Setup Utility,” on page 69.

If you need to change a part of the appliance configuration manually after running the setup utility, use the instructions in the remainder of this chapter as a guide.
Setting the Management Appliance Memory Size

VMware recommends configuring the management appliance with a memory size of 4GB. You can use smaller values for small datacenters, but the minimum memory size for the appliance is 1GB. Larger datacenters require more memory. For example, a datacenter containing 100 hosts, 2000 virtual machines, and 100 datastores requires 4GB.

If the management appliance runs on VMware Workstation, use the Workstation window to change the memory size. Otherwise, use the vSphere Client.

To set the memory size of the appliance using the Workstation window

1. Select the management appliance in the Favorites list.
2. Open the virtual machine settings editor (VM > Settings).
3. Select Memory, and use the slider to adjust the memory size for the virtual machine.
4. Click OK to close the virtual machine settings editor.

To set the memory size of the appliance using the vSphere Client

1. Display the Virtual Machine Properties dialog box.
2. On the Hardware tab, select Memory.
3. Set the memory value by using any of the following techniques:
   - Click a colored triangle (on the slider or in the legend).
   - Use the slider control.
   - Use the numeric field.
4. Click OK.

Setting the Number of Virtual Processors

VMware recommends configuring the management appliance to use two virtual processors.

If the management appliance runs on VMware Workstation, use the Workstation window to change the number of virtual processors. Otherwise, use the vSphere Client.

To set the number of virtual processors using the Workstation window

1. Select the management appliance in the Favorites list.
2. Open the virtual machine settings editor (VM > Settings).
3. Select Processors, and click Two.
4. Click OK to close the virtual machine settings editor.

To set the number of virtual processors for the appliance using the vSphere Client

1. Display the Virtual Machine Properties dialog box.
2. On the Hardware tab, select CPUs.
3. Select the number of virtual processors for the virtual machine.
   - The selector box displays only those options that are supported by the guest operating system.
4. Click OK.
Configuring the Firewall

The appliance includes a built-in firewall. By default, the setup utility allows incoming connections to the following ports:

- 5989 (CIM-XML)
- 8888 (WS-Man)
- 22 (SSH)

The CIM server supports the following protocol and port combinations:

- CIM-XML over HTTPS to port 5989.
- CIM-XML over HTTP to port 5988. This port is not enabled by default.
- WS-Man over HTTPS to port 8888.
- WS-Man over HTTP to port 8889. This port is not enabled by default.

You can open or close a port manually. The guest operating system in the appliance is Red Hat Enterprise Linux 5 Server, which uses the `iptables` firewall. For example, you can open a port as follows.

To open port 5988 in the firewall for incoming CIM requests

1. Using a shell window or an `iptables` front-end application, add the following rule:
   
   ```bash
   sudo iptables -A INPUT -p tcp --dport 5988 -j ACCEPT
   ```

2. Save the `iptables` configuration.
   
   ```bash
   /etc/init.d/iptables save
   ```

After you have enabled the appropriate ports, you can use the standard protocol for CIM-XML or WS-Man over HTTP or HTTPS to make a connection.

The CIM objects reside within the `vmware/esxv2` namespace. Clients must log on using a valid user ID and password known to the ESX/ESXi system or the vCenter server.

You do not need to configure outgoing ports for CIM indications because the CIM appliance does not support indications.

Installing Certificates

The management appliance creates a self-signed certificate the first time it starts up. The certificate is used for client connections to the management appliance and for management appliance connections to the target vSphere server.

You can replace the self-signed certificate if you choose, such as when the private key is compromised or when you purchase a certificate signed by a publicly-known CA.

To replace the default certificate

1. Obtain a new certificate and public-private key pair.
   
   For this example, assume the new private key is stored in the file `~/newkey`, and the new certificate and public key are stored in the file `~/newcert`.

2. Back up the old certificate files.
   
   ```bash
   cd /etc/vmware/ssl
   sudo cp rui.cert rui.cert.old
   sudo cp rui.key rui.key.old
   ```

3. Replace the old files `rui.cert` and `rui.key` with the new files.
   
   ```bash
   sudo cp ~/newcert /etc/vmware/ssl/rui.cert
   sudo cp ~/newkey /etc/vmware/ssl/rui.key
   ```
4 Restart the CIM services.
   sudo /etc/init.d/pegasus restart
   sudo /etc/init.d/wsman restart

The default certificate is used by the management appliance in three ways:

- To respond to a CIM-XML client connection request
- To respond to a WS-Man client connection request
- To request a connection to the target vSphere server

If you wish to use a different certificate for each purpose, you can replace the default certificate with three separate certificates.

**To use three separate certificates**

1 Obtain three new certificates and public-private key pairs.
   For this example, assume the new private keys are stored in the files `~/newkey1`, `~/newkey2`, `~/newkey3`, and the new certificate and public key are stored in the files `~/newcert1`, `~/newcert2`, `~/newcert3`.

2 Store the new files in the `/etc/vmware/ssl` directory with distinct names.
   
   ```bash
   sudo cp ~/newkey1 /etc/vmware/ssl/rui1.key
   sudo cp ~/newcert1 /etc/vmware/ssl/rui1.cert
   sudo cp ~/newkey2 /etc/vmware/ssl/rui2.key
   sudo cp ~/newcert2 /etc/vmware/ssl/rui2.cert
   sudo cp ~/newkey3 /etc/vmware/ssl/rui3.key
   sudo cp ~/newcert3 /etc/vmware/ssl/rui3.cert
   ```

3 Edit the file `/var/pegasus/cimserver_planned.conf` to configure the CIM server to use one of the certificates for CIM-XML responses.
   
   Search for the line containing `sslCertificateFilePath` and change it to look like this:
   ```bash
   sslCertificateFilePath /etc/vmware/ssl/rui1.cert
   ```
   Search for the line containing `sslKeyFilePath` and change it to look like this:
   ```bash
   sslKeyFilePath /etc/vmware/ssl/rui1.key
   ```
   Save the changes.

4 Edit the file `/etc/openwsman/openwsman.conf` to configure the CIM server to use one of the certificates for WS-Management responses.
   
   Search for the line containing `ssl_cert_file` and change it to look like this:
   ```bash
   ssl_cert_file = /etc/vmware/ssl/rui2.cert
   ```
   Search for the line containing `ssl_key_file` and change it to look like this:
   ```bash
   ssl_key_file = /etc/vmware/ssl/rui2s.key
   ```
   Save the changes.

5 Restart the CIM services.
   `sudo /etc/init.d/pegasus restart`
   `sudo /etc/init.d/wsman restart`

**NOTE** The filenames `/etc/vmware/ssl/rui.cert` and `/etc/vmware/ssl/rui.key` are fixed. These files are always used to connect to vSphere servers.

**Setting the CIM Server to Start Automatically**

The setup script configures the management appliance to start the CIM server automatically when the appliance starts up. To disable this behavior, see Appendix B, “CIM Server Setup Utility,” on page 69.
Starting the CIM Servers Manually

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

The WS-Management service is provided by openwsmand, from the OpenWSMAN project. openwsmand is responsible for receiving WS-Management requests and translating them to CIM-XML requests for the CIMOM.

When you run the setup utility, it configures both CIM servers to start when the management appliance starts up. If you need to check the status or restart the CIMOM, use the following procedures. The WS-Management service can be managed in the same way, substituting wsman for pegasus.

To check the status of the Pegasus CIMOM
1. Log on to a shell in the management appliance, using either a local or a remote connection. The appliance supports Secure Shell (SSH) connections.
2. Switch to the /etc/init.d directory.
3. Use the command sudo ./pegasus status to determine the current operational status of the Pegasus CIMOM in the appliance.

To start the Pegasus CIMOM
1. Log on to a shell in the management appliance, using either a local or a remote connection. The appliance supports Secure Shell (SSH) connections.
2. Switch to the /etc/init.d directory.
3. Use the command sudo ./pegasus start to start the Pegasus CIMOM in the appliance.

In some cases, such as when you have changed the configuration, you may need to restart the CIMOM when it is already running.

To restart the Pegasus CIMOM
1. Log on to a shell in the management appliance, using either a local or a remote connection. The appliance supports Secure Shell (SSH) connections.
2. Switch to the /etc/init.d directory.
3. Use the command sudo ./pegasus restart to restart the Pegasus CIMOM in the appliance.

Other Configuration Settings

A few additional settings can be changed in the management appliance configuration for debugging or performance reasons. These settings are described in Appendix C, “Advanced Server Configuration Settings,” on page 71.

Connecting Your Client to the CIMOM

Before you connect a client application to the CIMOM, you must have the following information available:

- Hostname or IP address of the management appliance. The CIM appliance does not support SLP advertisement.
- Port number in the appliance on which the CIM server is listening. See “Configuring the Firewall” on page 15 for information about supported TCP ports.
- SSL context information (if applicable).
Namespace for the VMware CIM Storage Management API. The namespace used by VMware is `vmware/esxv2`.

For WS-Management connections, you need the schema URIs. The URI you use depends on the object prefix:

- For CIM_* objects, use `http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/`. 
This chapter explains the schema used by the VMware CIM Storage Management API. In addition to conforming to CIM standards in general, the VMware CIM Storage Management API is based on SMI-S standards for storage device modeling. Where the SMI-S profile fails to model ESX/ESXi 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
- “Introduction to the VMware CIM Storage Management API Schema” on page 24
- “Sample System Environments” on page 26
- “Using the CIM Storage Management API Schema” on page 40

Relationship to Industry Standards

The schema for the VMware CIM Storage Management API 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 Virtualization profile, 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 Storage Management API implements version 2.17 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
CIM Intrinsic Methods Implemented by VMware

The In-band Virtualization 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 In-band Virtualization profile because it is well adapted to model the storage virtualization technology used by VMware.

In-band Virtualization 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.

A storage virtualizer (Figure 3-1) 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/ESXi (Figure 3-2) closely approximates a storage virtualizer as described in the In-band Virtualization profile specified by SMI-S version 1.02. ESX/ESXi handles Fibre Channel, parallel SCSI, block mode, or iSCSI storage systems. ESX/ESXi 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 Storage Management API Schema

The VMware schema, based on SMI-S version 1.02, implements a set of subprofiles of the In-band Virtualization profile to model virtual storage and its underlying physical storage in an ESX/ESXi environment. These include the core profile and related subprofiles that allow the user to browse the VMware storage environment.

In addition to the In-band Virtualization profile, VMware implements the Software subprofile for this release. The Software subprofile contains version information for the running software.

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

Exceptions to Standard Schema

The VMware extension closely approximates the In-band Virtualization profile, 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 In-band Virtualization profile in one important respect.

In a typical storage virtualization environment, the storage virtualizer has a Fibre Channel connection to the client. Therefore, the model used in the In-band Virtualization profile includes an FCPort object attached to the SCSIProtocolController for the client. The SMI-S schema models the connections as shown in Figure 3-3.
In a VMware ESX/ESXi environment, a virtualizer client connection is not a physical connection. The connection is a logical relationship between the ESX/ESXi host and a virtual machine that runs on the host. ESX/ESXi virtualizes disks as a parallel SCSI connection. No Fibre Channel connection exists, either physical or virtual. Consequently, the VMware extension models the connection with an SPIPort object instead of an FCPort object, as shown in Figure 3-4.

**Figure 3-3. The SMI-S Model for a SCSI Controller Connection**

**Figure 3-4. The VMware Model for a SCSI Controller Connection**
Figure 3-4 includes a series of objects on the right side that represent a virtual machine. The SMI-S does not model storage clients. The VMware schema extension, however, does model virtual machines (which are storage clients with respect to ESX/ESXi) because they are an important part of the ESX/ESXi environment.

The objects on the right, in conjunction with the two objects at the bottom center of Figure 3-4, are marked with asterisks to distinguish them from standard objects shared with the SMI-S. The objects representing the virtual machine and the virtual initiator port derive from classes used by the SMI-S. These objects are associated in patterns very similar to those of the specification.

Figure 3-4 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. The VMware extension schema uses the DiskPartition object because ESX/ESXi always creates a VMFS within a partition on the target LUN.

Introduction to the VMware CIM Storage Management API Schema

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

Types of Clients

Clients of the CIM Storage Management API can 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 the client 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/ESXi 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 follows the extended paths and retrieves additional information about virtual machines and their relationships to host resources.

Conventions Used in This Document

This document uses the following textual conventions to help clarify the meaning of the text:

- 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 must 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, you specify “VMWARE_VirtualInitiatorSCSIProtocolController”, but to invoke a method on the parent class, you 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 might 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 might be labeled:
  
  VirtualInitiatorSCSIProtocolController

  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. This is done solely to make the illustrations more compact. For example,

  VirtualInitiatorSCSIProtocolController

  might appear as follows:

  VirtualInitiator
  SCSIProtocolController

- 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 In-band Virtualization profile 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. Consider these areas experimental, because future standards developments might 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 illustrations are included to aid in visualizing the entities modeled in the instance diagrams.

Traversing Associations

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 so by traversing the association instance that relates them (see Figure 3-5).
Several approaches are 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 method 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 the same time.

- 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 approach takes two steps, but the total network traffic is less. This approach is a good choice if you are interested in only 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 approach 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 more than one reference is returned) and invoke AssociatorNames on that reference to obtain another reference or set of references, until you reach the object of interest. This approach is especially useful for traversing a series of one-to-one associations.

### Sample System Environments

These illustrations show how the CIM Storage Management API 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 27
- “Virtual Machine with Virtual Disk and Snapshot on Local Storage” on page 28
- “Virtual Machine with Single Virtual Disk on a LUN in a Storage Array” on page 30
- “Virtual Machine Using Raw Device Mapping in a Storage Array” on page 32
- “Two Virtual Machines on Different Servers Accessing Shared LUN” on page 34
- “Multipath SAN Environment with Two Storage Arrays” on page 37

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/ESXi host. After the block diagram, an instance diagram 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 3-6.

**Figure 3-6.** Logical View of a Virtual Machine with Single Virtual Disk on Local VMFS Storage

This environment is represented to a CIM client as shown in Figure 3-7.

**Figure 3-7.** Schema Model of a Virtual Machine with Single Virtual Disk on Local VMFS Storage

*Figure 3-7* can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 3-6.
- The objects and their associations in the lower left part of Figure 3-7 represent the virtual machine and its virtual host bus adapter (HBA). For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42.

- The four objects and their associations in the lower right part of Figure 3-7 represent the virtual disk attached to the virtual HBA. For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43.

- The StoragePool object represents a VMFS. For more detail on this part of the model, see “Virtual Disks on a VMFS” on page 45.

- The three objects and their associations in the upper right part of Figure 3-7 represent the direct-attached host storage. For more detail on this part of the model, see “VMFS Volumes Sharing a Directly Attached Disk” on page 47.

**NOTE** The VMware CIM Storage Management API models iSCSI with the same objects as direct-attached storage. If your host has networked storage (except for Fibre Channel storage), it cannot be distinguished from the direct-attached storage. For more detail on Fibre Channel storage, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49.

**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 3-8.

**Figure 3-8.** Logical View of a Virtual Machine with Virtual Disk and Snapshot on Local Storage

This environment is represented to a CIM client as shown in Figure 3-9. The two disk files are associated as synchronized storage.
The four objects and their associations along the left side of Figure 3-9 represent the virtual machine and its virtual HBA. For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42.

The four objects and their associations in the middle of the lower part of Figure 3-9 represent the file or set of files implementing the base virtual disk. For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43.

The four objects and their associations in the lower right part of Figure 3-9 represent 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. see the section “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43 for information on working with virtual disk objects.

The StoragePool object represents the VMFS. For more detail on this part of the model, see “VMFS Volumes Sharing a Directly Attached Disk” on page 47.
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 3-10.

**Figure 3-10.** Logical View of a Virtual Machine with Single Virtual Disk on a LUN in a Storage Array

This environment is represented to a CIM client as shown in Figure 3-11.
Figure 3-11. Schema Model of a Virtual Machine with Single Virtual Disk on a LUN in a Storage Array

Figure 3-11 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 3-10.

- The four objects and their associations along the left side of Figure 3-11 represent the virtual machine and its virtual HBA. For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42.

- The four objects and their associations in the middle of the lower part of Figure 3-11 represent the file or set of files implementing the base virtual disk. For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43.

- The four objects and their associations in the upper right part of Figure 3-11 represent the Fibre Channel HBA and the LUN that holds the VMFS. For more detail on this part of the model, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49.

- The StoragePool object in the center represents the VMFS. For more detail on this part of the model, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49.
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 3-12.

Figure 3-12. Logical View of a Virtual Machine Using Raw Device Mapping in a Storage Array

This environment is represented to a CIM client as shown in Figure 3-13.
The four objects and their associations in the lower right part of Figure 3-13 represent the virtual machine and its virtual HBA. For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42.

The SCSIProtocolController and the FCPort objects at the top of Figure 3-13 represent the host’s Fibre Channel HBA. For more detail on this part of the model, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49.

The four objects and their associations in the lower left part of Figure 3-13 represent the virtual disk (implemented with a mapping file, in this case). For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43.

The StorageExtent, DiskPartition, and StoragePool objects in the upper left part of Figure 3-13 represent a partitioned LUN with a VMFS. For more detail on this part of the model, see “VMFS Volumes Sharing a Directly Attached Disk” on page 47. The mapping file is always stored on a VMFS.

The StorageExtent object in the middle near the top of Figure 3-13 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 3-14.

**Figure 3-14.** Logical View of Two Virtual Machines on Different Servers Accessing Shared LUN

In this situation, the CIM 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. Neither host can return information 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 3-15 and Figure 3-16 show the parts of the environment visible to each of the two hosts.
**Figure 3-15. Schema Model of Two Virtual Machines on Different Servers Accessing Shared LUN, As Viewed by the First Host**

Figure 3-15 is the part of the virtual storage environment visible to one host. This set of objects and associations can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 3-14, “Logical View of Two Virtual Machines on Different Servers Accessing Shared LUN,” on page 34. The shaded boxes on the right side of Figure 3-15 represent the objects visible to the client when connected to the second host.

- The three objects and their associations at the top of Figure 3-15 represent the host and its Fibre Channel HBA. For more detail on the Fibre Channel HBA model, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49.
- The four objects and their associations in the lower left of Figure 3-15 represent the virtual machine running on the first host, including its virtual HBA. For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42.
The four objects and their associations in the middle lower part of Figure 3-15 represent the virtual disk belonging to the virtual machine on this host. For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43.

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, see “VMFS Volumes Sharing a Directly Attached Disk” on page 47.

Similarly, the StoragePool object represents the shared VMFS volume, which is also visible to both hosts. Figure 3-16 shows the part of the virtual storage environment visible to the second host. Figure 3-16 contains the same kinds of objects and associations as Figure 3-15. 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 must compare their UUIDs. Getting the UUID of a VMFS volume is described in the section “Virtual Disks on a VMFS” on page 45.
Figure 3-16. Schema Model of Two Virtual Machines on Different Servers Accessing Shared LUN, As Viewed by the Second Host

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. The illustration shows 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 3-17.
This environment is represented to a CIM client as shown in Figure 3-18.

**NOTE** The CIM Storage Management API does not model Fibre Channel ports for the storage arrays.
Figure 3-18. Schema Model of a Multipath SAN Environment with Two Storage Arrays

Figure 3-18 can be viewed as a set of groupings of objects, in which each group represents a part of the logical view illustrated in Figure 3-17.

- The four objects and their associations in the lower left of Figure 3-18 represent a virtual disk. For more detail on this part of the model, see “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43. 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 3-18 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 3-18 represent the virtual machine and its virtual HBA. For more detail on this part of the model, see “Virtual Host Bus Adapters on a Virtual Machine” on page 42. 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 3-18 represent a partitioned LUN with a VMFS. For more detail on this part of the model, see “VMFS Volumes Sharing a Directly Attached Disk” on page 47. 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 part of the model, see “VMFS Volumes Sharing a Fibre Channel LUN” on page 49. Across from them to the right are the corresponding objects and associations for the host’s other HBA.

NOTE 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 Storage Management API 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 26.

If your discovery approach begins with the host, you must 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 40
- “Virtual Host Bus Adapters on a Virtual Machine” on page 42
- “Virtual Disks Connected to a Virtual Host Bus Adapter” on page 43
- “Virtual Disks on a VMFS” on page 45
- “VMFS Volumes Sharing a Directly Attached Disk” on page 47
- “VMFS Volumes Sharing a Fibre Channel LUN” on page 49
- “VMFS Spanning Two LUNs” on page 51
- “Raw Device Mapping” on page 53
- “System Devices of a Host” on page 54

Each subsection presents a diagram of the CIM Storage Management API model for a typical small part of an ESX/ESXi 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 Storage Management API 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 3-19.
Typical Uses

To get the name of the ESX/ESXi system, a simple way is to enumerate all instances of the ESXComputerSystem class. Only one ESXComputerSystem is available to a given client connection. Example 3-1 shows this way to find the host object.

**Example 3-1.**

```cim
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 3-2 shows this way to find the host object.

**Example 3-2.**

```cim
Array<CIMInstance> cs;
CIMInstance theEsxInstance = NULL;
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);
    }
}
```

To find all the virtual machines managed by the ESX/ESXi 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 3-3 determines the number of virtual machines.
Example 3-3.

```c
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 Storage Management API models for virtual storage can be viewed as two groups of objects: virtual host bus adapters (HBAs) and virtual disks. Figure 3-20 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 3-20. Virtual Host Bus Adapters on a Virtual Machine**

![Diagram showing virtual host bus adapters on a virtual machine.]

### 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, with one reference for each virtual HBA.

**Example 3-4.**

```c
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, you must traverse the associations to the ProtocolEndpoint objects:

**Example 3-5.**

```java
// 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 3-6.**
  ```java
  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 3-7.**
  ```java
  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."

**Virtual Disks Connected to a Virtual Host Bus Adapter**

The CIM Storage Management API models virtual storage in two parts: virtual HBAs and virtual disks. **Figure 3-21** 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 ProtocolEndpoint object of the virtual HBA
- A SPIPort object

**NOTE** This way of modeling virtual storage is a departure from the SMI-S standard. The VMware extension schema departs from the standard in this case because ESX/ESXi 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.
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, with one reference for each virtual disk attached to the virtual HBA.

**Example 3-8.**

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

To retrieve information about a specific virtual disk, you must 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 3-9.**

```
Array<CIMObjectPath> vport;
vport = client.associatorNames(NAMESPACE, theVdEndpointRef, "CIM_DeviceSAPImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdPortRef = vport[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 3-10.

```csharp
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 3-11.

```csharp
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 3-12.

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

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

Example 3-13.

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

**Virtual Disks on a VMFS**

Figure 3-22 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. For this reason, virtual disk files 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 difference 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 preallocated 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 might exceed the size reported to the virtual machine.
Figure 3-22. Virtual Disks on a VMFS

**Typical Uses**

The following are some typical uses for these objects and associations.

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

1. 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 3-14.**
   ```
   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 3-15.**
   ```
   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 3-16.**
   ```
   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 3-17.**
   ```
   CIMProperty prop = theStoragePoolInstance.getProperty(
                   theStoragePoolInstance.findProperty("InstanceID"));
   String uuid = prop.getValue().toString();
   ```
VMFS Volumes Sharing a Directly Attached Disk

Figure 3-23 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 3-23. VMFS Volumes Sharing a Directly Attached Disk

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, with one reference for each VMFS visible to the ESX/ESXi host.

Example 3-18.

```csharp
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, with one reference for each controller attached to the ESX/ESXi host.

Example 3-19.

```csharp
Array<CIMObjectPath> ctlr;
ctlr = client.associators(NAMESPACE, theESXInstance.getPath(),
   "CIM_SystemDevice",
   "CIM_SCSIProtocolController");
```
Use the client’s accessor method to get the `ElementName` properties of the `SCSIProtocolController` objects.

**Example 3-20.**

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

Locate the desired controller in the list.

**Example 3-21.**

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

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, with one reference for each LUN attached to the controller.

**Example 3-22.**

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

For each `StorageExtent` object, use the `GetProperty` method with the object reference and the property name `Name`.

Each invocation produces a unit ID.

**Example 3-23.**

```java
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();
}
```

Locate the desired LUN in the list.

**Example 3-24.**

```java
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, with one for each partition of the LUN.
Example 3-25.

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

8 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 3-26.

```csharp
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]);
    }
}
```

9 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 3-27.

```csharp
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 3-28.

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

**VMFS Volumes Sharing a Fibre Channel LUN**

Figure 3-24 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 3-23, “VMFS Volumes Sharing a Directly Attached Disk,” on page 47) is the presence of an FCPPort object that models the Fibre Channel characteristics of the SCSI controller, including its WWPN.
Figure 3-24. VMFS Volumes Sharing a Fibre Channel LUN

```
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).

   Example 3-29.
   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).

   Example 3-30.
   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];
```

NOTE  A VMFS volume can span more than one LUN in general. This example assumes a single LUN.
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 3-31.

```java
Array<CIMObjectPath> ctlr;
ctlr = client.associatorNames(NAMESPACE, theLunRef,
                             "CIM_ProtocolControllerAccessesUnit");
// Many-to-one mapping produces a 1-element array.
CIMObjectPath theCtlrRef = ctlr[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 3-32.

```java
Array<CIMObject> fc;
fcp = client.associators(NAMESPACE, theCtlrRef,
                        "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 3-33.

```java
CIMProperty prop = theFCPortInstance.getProperty(
    theFCPortInstance.findProperty("PermanentAddress"));
String theAddress = prop.getValue().toString();
```

**VMFS Spanning Two LUNs**

Figure 3-25 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.
Typical Uses

The following are some typical uses for these objects and associations.

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 3-34.**
   ```csharp
   Array<CIMObjectPath> part;
   part = client.associatorNames(NAMESPACE,
                                theStoragePoolInstance.getPath(),
                                "CIM_ConcreteComponent");
   ```

2. 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 3-35.**
   ```csharp
   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 3-36.**

```java
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 3-26 shows the association of objects modeling a raw device mapping. VMWARE_RDMStorageVolume, 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, see “Virtual Machine Using Raw Device Mapping in a Storage Array” on page 32.) 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** 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.

**Figure 3-26. Raw Device Mapping**

```

```

**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 theAssociators 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 3-37.**

```java
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 3-38.**
   ```c++
   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 3-39.**
   ```c++
   uint64 rdmSizeBytes = numBlocks * blockSize;
   ```

**System Devices of a Host**

Figure 3-27 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, but the other illustrations omit some of the `SystemDevice` associations.

**Figure 3-27. System Devices of a Host**

![Diagram of System Devices of a Host]

**Typical Uses**

The following are some typical uses for these objects and associations.

**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 3-40.

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

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

Example 3-41.

```java
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

1 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 3-42.

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

2 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 3-43.

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

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

Example 3-44.

```java
Array<String> name;
for (int i = 0; i < fchba.size(); i++) {
    CIMProperty prop = fchba[i].getProperty(
        fchba[i].findProperty("ElementName"));
    name[i] = prop.getValue();
}
```
The following examples are included as starting points to help you assemble the basic building blocks presented in the chapter “VMware CIM Schema” on page 19. These examples show how to connect to the Pegasus CIMOM in the management appliance and how to send and receive CIM operation requests and responses in the context of a simple utility program.

Several approaches to navigating the VMware 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 Storage Management API effectively.

All examples use the C++ programming language and the OpenPegasus CIM client library. Your choice of language and library might 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 67. Please read the restrictions before using this sample code.

This chapter contains the following samples:

- “Connecting to the Pegasus CIMOM” on page 58
- “Retrieving Information about an ESX/ESXi Host” on page 59
- “Enumerating StorageExtent (LUN) Objects Starting from the ESXComputerSystem” on page 60
- “Listing Virtual Storage Available to Virtual Machines” on page 61
- “Enumerating Virtual Machines Starting from StoragePool (VMFS) Objects” on page 63
Connecting to the Pegasus CIMOM

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

```cpp
#include <Pegasus/Common/Config.h>
#include <Pegasus/Client/CIMClient.h>
#include <iostream>
using namespace Pegasus;
using namespace std;
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/ESXi Host

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

```cpp
#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/ESXi 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> esxs;
    esxs = client.enumerateInstances(NAMESPACE,
            "VMWARE_ESXComputerSystem");
    // Can only get 1 ESXComputerSystem from each connection.
    CIMInstance theESXInstance = esxs[0];
    return (theESXInstance);
}

void PrintESXServerInfo(const CIMObject &theESXInstance) {
    CIMProperty prop = theESXInstance.getProperty(
            theESXInstance.findProperty("ElementName");
    String EsxName = prop.getValue().toString();
    cout << "ESX 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/ESXi 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.

```cpp
#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/ESXi 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);
}
```
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.

```cpp
#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/ESXi 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> esxs;
    esxs = client.enumerateInstances(NAMESPACE,
        "VMWARE_ESXComputerSystem");
    // Can only get 1 ESXComputerSystem from each connection.
    CIMInstance theESXInstance = esxs[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> vctlr;
    vctlr = client.associatorNames(NAMESPACE, theVMInstance.getPath(),
        "VMWARE_VirtualInitiatorControllerSystemDevice");
    // For each virtual controller, navigate to its virtual disks.
    for (int i = 0; i < vctlr.size(); i++) {
        Array<CIMObjectPath> viport;
        viport = client.associatorNames(NAMESPACE, vctlr[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_DeviceSAPImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViEndpointRef = viendpt[0];
Array<CIMObjectPath> vgendpt;
vgendpt = client.associatorNames(NAMESPACE, theViEndpointRef, "CIM_ActiveConnection");

// For each virtual disk, navigate to the FileStorageVolume object.
for (int j = 0; j < vgendpt.size(); j++) {
    FindOneVirtualDisk(client, vgendpt[j]);
}

void FindOneVirtualDisk(const CIMClient &client,
    const CIMObjectPath &theVdEndpointRef) {
    Array<CIMObjectPath> vdport;
    vdport = client.associatorNames(NAMESPACE, theVdEndpointRef, "CIM_DeviceSAPImplementation");

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

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

VMware, Inc. 63
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> vдендpt;
vдендpt = client.associatorNames(NAMESPACE, theVdPortRef,
    "CIM_DeviceSAPImplementation");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theVdEndptRef = vдендpt[0];

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

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

Array<CIMObjectPath> виенtrl;
виенtrl = client.associatorNames(NAMESPACE, theViPortRef,
    "CIM_ProtocolControllerForPort");
// One-to-one mapping produces a 1-element array.
CIMObjectPath theViCtlrRef = виенtrl[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;
};
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. Use Rights: Subject to the restrictions below, you may download and make a reasonable number of copies of the SDK contents for your use solely for the purpose of creating software that communicates with VMware Software (“Developer Software”). For the code designated as “distributable code” in the SDK documentation and posted at www.vmware.com, you may use and merge all or portions of the “distributable code” with your Developer Software. Any merged portion of the “distributable code” is subject to this SDK Agreement. For the code designated as “sample code” in the SDK documentation and posted at www.vmware.com, you may modify or create derivative works of all or portions of the “sample code”. You are permitted to re-distribute the “distributable code” and the modified or derivative works of the “sample code” only as part of your Developer Software for non-commercial or commercial use.

3. Restrictions: You agree that you will not (1) use the SDK to create, 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 as expressly permitted in Section 2; (4) distribute, sell, lease, rent, lend, or sublicense any part of the SDK to any third party except as expressly permitted in Section 2 or; (5) use the SDK to (a) create, design or develop software or service to circumvent, enable, modify or provide access, permissions or rights which would violate the technical restrictions of VMware Software, any additional licensing terms provided by VMware via product documentation, email notification and/or policy change on VMware website, and/or the terms of the End User License Agreements of VMware products; (b) disable, remove, over-ride or modify the display of any VMware product End User License Agreements to the end customers; and (c) 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.

The restrictions in this Section 3 shall not apply if and to the extent they contradict mandatory local law (including, but not limited to, law implementing the EC Software Directive) provided that, before you exercise any rights that you believe to be entitled to based on mandatory law, you provide VMware with 30 days prior written notice at SDK_Legal@vmware.com and provide all reasonably requested information to allow VMware to assess your claim and, at VMware's sole discretion, to provide alternatives that reduce any adverse impact on VMware's intellectual property or other rights.

4. VMware retains ownership of the SDK, including without limitation all copyrights and other intellectual property rights therein. You may not remove, delete or modify any of VMware copyright statements in the SDK.
5. You may not use VMware's name, trademarks or service marks in your Developer Software's names or in a way that suggests your Developer Software is certified or endorsed by VMware. Your use of VMware product names and trademarks shall comply with the Branding Guidelines posted at www.vmware.com.

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 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. Indemnification: 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 or derivative works of the "sample code", distribution or use of your Developer Software or breach of this SDK Agreement.

10. 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.

If you have any questions about this SDK Agreement, please direct all correspondence to SDK_Legal@vmware.com.
This appendix provides detailed usage information for the CIM server setup utility in the management appliance. This appendix includes the following sections:

- “Running the Setup Script” on page 69
- “Command Line Options” on page 69
- “Usage Example” on page 70

Running the Setup Script

The setup script configures the CIM server in the management appliance. You must run the setup script to configure the CIM server before its first use. You need to run the setup script with root privileges. Log in as the vi-admin user, then use `sudo` to run the script.

You can run the setup script in one of two ways. If you supply the `--silent` option, the setup script applies the other command-line options. If you omit the `--silent` option, the setup script ignores any other command-line options and prompts for all necessary input.

Command Line Options

The setup script supports the following command line options:

`--server <hostname>`

Specifies the vCenter server or ESX server to which the management appliance will be bound. If you bind the appliance to a single ESX server, the CIM server collects data from that server only. If you bind the appliance to a vCenter server, the CIM server collects data from all ESX servers managed by that vCenter server.

`--username <username>`

Specifies a login account on the server to which the appliance will be bound.

`--password <password>`

Specifies the password for the login account on the server to which the appliance will be bound.

`--addchkconfig <pegasus|wsman|all|none>`

(optional) Specifies that the CIM server should launch when the management appliance starts up. The suboptions are:

- `pegasus`
  - Launch the CIM-XML service on startup.

- `wsman`
  - Launch the WS-Management service on startup.
all
  Launch both CIM-XML and WS-Management services on startup.

none
  Do not launch any CIM services on startup.

If the --addchkconfig option is not specified, the setup script sets the value to all.

--startup <pegasus|wsman|all|none>
  (optional) Specifies that the CIM server should launch immediately after the management appliance is configured. The suboptions are:

  pegasus
    Launch the CIM-XML service immediately.

  wsman
    Launch the WS-Management service immediately.

  all
    Launch both CIM-XML and WS-Management services immediately.

  none
    Do not launch any CIM services at this time.

If the --startup option is not specified, the setup script sets the value to all.

--silent
  Specifies that the setup script should not prompt for more information. If any necessary options are missing and --silent is specified, the script fails.

Usage Example

  sudo /etc/vmware/vmware-smis-setup.pl --silent --server myhost.mydomain.com --username localuser --password secret

This example runs the setup script in batch mode, binding the management appliance to a machine named myhost, with a login name of localuser. These credentials will be used for all future CIM queries.

The setup script logs all actions to the file /var/log/vmware/vmware-smis-setup.log.
Advanced Server Configuration Settings

Some installations require additional configuration that is not described in “Setting Up Your Development Environment” on page 13. This appendix describes configuration options used for troubleshooting or for working with very large datacenters.

This appendix contains the following topics:

- “CIM Server Configuration Files” on page 71
- “Adjusting the CIM Server Cache Refresh Interval” on page 73
- “Adjusting the HTTP Response Buffer Size for vCenter Server” on page 74

CIM Server Configuration Files

To configure the CIM server in the management appliance, you normally use the /etc/vmware/vmware-smis-setup.pl setup utility. The setup utility addresses the most common configuration needs. However, some configuration settings can only be set manually. These are settings you would use only for debugging or for better performance in unusual datacenter environments.

The CIM server configuration is stored in two configuration files. One file contains configuration parameters for the Pegasus CIMOM. The other file contains setup parameters for the SMI-S API that is served by the CIMOM. This appendix describes the changes you can make in both configuration files.

Pegasus Configuration File

The Pegasus configuration file is installed by default at /var/pegasus/cimserver_current.conf. This file contains the configuration of the currently running instance of the Pegasus CIMOM. Do not change this file directly.

To change the Pegasus configuration, edit the file /var/pegasus/cimserver_planned.conf and restart the Pegasus CIMOM, as described in “Starting the CIM Servers Manually” on page 17.

Some of the Pegasus configuration parameters are described in Table C-1. See http://wiki.opengroup.org/ for more information.

Table C-1. Pegasus configuration parameters

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableHttpConnection</td>
<td>(true or false) CIMOM supports unsecured connections</td>
<td></td>
</tr>
<tr>
<td>enableHttpsConnection</td>
<td>(true or false) CIMOM supports secured (SSL) connections</td>
<td></td>
</tr>
<tr>
<td>enableAssociationTraversal</td>
<td>(true or false) CIMOM supports association traversal methods</td>
<td></td>
</tr>
<tr>
<td>enableNamespaceAuthorization</td>
<td>(true or false) CIMOM requires user authorization for namespace access</td>
<td>Configure user authorization using cimauth utility</td>
</tr>
</tbody>
</table>
Table C-1. Pegasus configuration parameters (Continued)

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableIndicationService</td>
<td>(true or false) CIMOM supports CIM indications</td>
<td></td>
</tr>
<tr>
<td>enableSubscriptionsForNonprivilegedUsers</td>
<td>(true or false) CIMOM allows nonprivileged users to subscribe to CIM indications</td>
<td></td>
</tr>
<tr>
<td>enableRemotePrivilegedUserAccess</td>
<td>(true or false) CIMOM allows access by privileged users on remote systems</td>
<td></td>
</tr>
<tr>
<td>logLevel</td>
<td>(TRACE, INFORMATION, WARNING, SEVERE, or FATAL) Log file verbosity</td>
<td>Default = INFORMATION</td>
</tr>
<tr>
<td>sslCertificateFilePath</td>
<td>For secured connections to the datacenter server, the file containing the CIMOM's certificate</td>
<td></td>
</tr>
<tr>
<td>slp</td>
<td>(true or false) CIMOM issues SLP advertisements</td>
<td></td>
</tr>
<tr>
<td>enableAuthentication</td>
<td>(true or false) CIMOM authenticates non-root users</td>
<td></td>
</tr>
<tr>
<td>forceProviderProcesses</td>
<td>(true or false) Providers run in separate processes</td>
<td></td>
</tr>
</tbody>
</table>

VMware Configuration File

The VMware-specific CIM configuration parameters for the management appliance are stored in the /etc/vmware/vmware-cim-config.xml file. You can change some of these parameters using the /etc/vmware/vmware-smis-setup.pl setup utility. Use the setup utility whenever possible.

If you modify the configuration file, you must restart the CIM server, as described in “Starting the CIM Servers Manually” on page 17.

The configuration parameters are described in Table C-2. The configuration is structured as an XML file. Each parameter is identified by its ancestry and its node name.

Table C-2. VMware CIM configuration file

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;vmacore&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;ssl&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;libraryPath&gt;</td>
<td>Path to directory containing openssl library files</td>
<td>Do not change this.</td>
</tr>
<tr>
<td>&lt;log&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;directory&gt;</td>
<td>Path to directory containing log files for CIM providers</td>
<td>Change this only if advised by VMware Support.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>Name of log file for CIM providers</td>
<td>Change this only if advised by VMware Support.</td>
</tr>
<tr>
<td>&lt;outputToConsole&gt;</td>
<td>Whether to copy log entries to the system console (true or false)</td>
<td>Change this only if advised by VMware Support.</td>
</tr>
<tr>
<td>&lt;level&gt;</td>
<td>Logging verbosity (panic, error, warning, info, verbose, or trivia)</td>
<td>Increase log level for debugging, or decrease to save disk space.</td>
</tr>
<tr>
<td>&lt;cim_provider&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;contextPersistence&gt;</td>
<td>Lifetime (in seconds) of user’s login context in CIMOM</td>
<td>Change this only if advised by VMware Support.</td>
</tr>
<tr>
<td>&lt;refreshInterval&gt;</td>
<td>How often (in seconds) CIMOM should do a full cache refresh. Default is every 6 hours. Minimum is 1 hour.</td>
<td>See “Adjusting the CIM Server Cache Refresh Interval” on page 73.</td>
</tr>
<tr>
<td>&lt;viURL&gt;</td>
<td>URL of server to which this appliance is bound</td>
<td>Use the setup utility to change this value.</td>
</tr>
</tbody>
</table>
Adjusting the CIM Server Cache Refresh Interval

The CIM server maintains a cache of information about vSphere storage resources available to the server. The cache is used to provide improved response times, especially for large datacenters managed by vCenter.

The default time interval for refresh vSphere data is six hours. After each interval, the CIM server collects a new snapshot of vSphere data.

For very large datacenters, it might be necessary to increase the refresh interval. The time needed to refresh the cache depends on the following factors:

- The total number of virtual disks on all datastores.
- The number of virtual machines that are powered on.

For datacenters containing hundreds of hosts or datastores, and thousands of virtual machines, the cache refresh operation can take longer than six hours. In such cases, you should increase the refresh interval to exceed the time that the CIM server takes to fill the entire cache.

To determine the time that the server takes to fill the cache

1. Examine the CIM log files. You can find the log files in the /var/log/vmware/ directory. The log file names have the pattern `vmware-cim-<number>.log`, where `<number>` represents a digit from 0 to 9.
2. Locate the log file that has the earliest modification timestamp.
3. Search the file for the string "Successfully logged as ". The timestamp of the record that contains the string shows when the earliest refresh interval started.
4. Search the remainder of the file for the string "Mgr: successfully initialized". The timestamp of the record that contains the string shows when the earliest refresh interval completed.

If the file does not contain the string "Mgr: successfully initialized", the refresh interval has continued into the next log file. Locate the next log file (by modification time stamp) and continue searching for the string "Mgr: successfully initialized". If necessary, search in successive log files until you find the string representing the end of the refresh interval. The timestamp of the record that contains the string shows when the earliest refresh interval completed.
5. Subtract the earlier timestamp from the later timestamp, giving the time taken to complete the cache refresh.
6. When you know the time it takes the CIM server to complete a cache refresh for your datacenter, you can adjust the refresh interval to be slightly larger than that time.

To change the refresh interval

1. Edit the file /etc/vmware/vmware-cim-config.xml.
2. Locate the cim_provider element in the XML tree.
3. Locate the refreshInterval element within the cim_provider element.
4. Change the value of the refreshInterval element. The value is expressed in seconds.
5. Save your changes to the file /etc/vmware/vmware-cim-config.xml.
6 Restart the CIM server. See the instructions in “To start the Pegasus CIMOM” on page 17.

When you restart the CIM server, it starts a new cache refresh. After the new cache is filled, you can use the procedure “To determine the time that the server takes to fill the cache” on page 73 to determine the time it took to complete the refresh and verify that your new refresh interval is large enough.

If you are unsure about changing the refresh interval, call VMware Technical Support for assistance.

Adjusting the HTTP Response Buffer Size for vCenter Server

VMware vCenter 4 limits the size of the buffer it uses for HTTP responses. By default, the size limit is 100MB. In some situations, the buffer needs to be set to a larger size.

For very large datacenters, HTTP responses can exceed the default size limit. For example, in a datacenter that has more than 200 hosts, 2000 virtual machines and 200 datastores, HTTP responses from the vCenter server might exceed the default response buffer size. When this happens, vCenter chunks the data into separate responses.

SOAP clients that use HTTP chunking are unaffected by this limitation. vSphere 4 clients handle chunked responses correctly, but earlier clients do not. Most earlier clients are not affected because they do not request enough data to exceed the buffer size.

However, the VMware Storage Management CIM server also does not handle chunked responses correctly. When the response exceeds the buffer size, the vCenter server returns empty or incomplete responses. The client often reports an error message such as "Cache: Error Checking for Updates. Done - Incomplete header received" in these situations.

If your datacenter exceeds 200 hosts, 2000 virtual machines and 200 datastores, or if your client receives errors related to incomplete responses, VMware recommends that you increase the HTTP response buffer size to 200MB to avoid exceeding the response buffer limit.

To prevent chunked responses, increase the size of the vCenter response buffer.

To change the buffer size

1 On the vCenter server machine, open the vpxd.config file with a text editor.

   cd C:\Documents and Settings\All Users\Application Data\VMware\VMware VirtualCenter
   notepad vpxd.config

2 Within the <vmacore> node, add an <http> node, if it is not already present.

   The configuration file should look similar to this, in part:

   <config>
      ... 
      <vmacore>
         <threadPool>
            <TaskMax>30</TaskMax>
         </threadPool>
         <http>
            <maxBufferedResponseBytes>104857600</maxBufferedResponseBytes>
         </http>
      </vmacore>
   </config>

3 Save your changes and exit the text editor.

4 Right-click My Computer and choose Manage.

5 Double-click Services and Applications in the Name column on the right.

6 Double-click Services in the Name column on the right.

7 Right-click VMware VirtualCenter Server in the Name column on the right, and choose Restart.
A  API
Application programming interface. A specified set of functions that allows one to access a module or service programmatically.

C  CIM (Common Information Model)
A set of object-oriented schemas, defined by the DMTF (Distributed Management Task Force), that is not bound to any particular implementation. CIM defines how managed elements in a networked environment are represented as a common set of objects and relationships that multiple users can view, share, and control.

CIMOM (CIM Object Manager)
A component that stores class definitions and populates requests for CIM operations with information returned from specific data providers. See also CIM (Common Information Model).

D  DMTF (Distributed Management Task Force)
The organization responsible for the CIM standard. An industry-wide consortium of hardware and software vendors with a mission to define interoperability standards.
See also CIM (Common Information Model).

G  growable disk
A type of virtual disk in which the disk space is not preallocated to its full size. The disk files begin small and grow as data is written to the disk. See also preallocated disk.

H  HBA (host bus adapter)
A device that connects one or more peripheral units to a computer and manages data storage and I/O processing (often for Fibre Channel, IDE, or SCSI interfaces). An HBA can be physical (attached to a host) or virtual (part of a virtual machine).

host
A computer that uses virtualization software to run virtual machines. Also called the “host machine” or “host computer.” The physical computer on which the virtualization (or other) software is installed.

I  IDL (interface definition language)
A human-readable syntax used to specify an API. An API described in the IDL can be compiled into stubs on a client machine. See also stub.

IBVP (In-Band Virtualizer Profile)
The standard profile that the VMware Migration Server adapts to describe ESX server. The CIM SDK version 2 is based on the IBVP.
ISV (independent software vendor)
A company that develops and sells software for use on other companies’ platforms. Includes systems management vendors, imaging and provisioning vendors, storage management vendors, and so on.

L
LUN (logical unit number)
An identifier for a disk volume in a storage array.

M
MOF (Managed Object Format)
A file format for the CIM IDL that describes model classes.
See also CIM (Common Information Model), IDL (interface definition language).

P
Pegasus
An open source implementation of the DMTF CIM and WBEM standards. Pegasus is the CIMOM used by ESX Server. Pegasus also provides a client library used to write CIM client applications.
See also CIM (Common Information Model), CIMOM (CIM Object Manager), DMTF (Distributed Management Task Force).

physical disk
In hosted products, a hard disk in a virtual machine that is mapped to a physical disk drive or partition on the host machine. A virtual machine's disk can be stored as a file on the host file system or on a local hard disk. When a virtual machine is configured to use a physical disk, vCenter Server directly accesses the local disk or partition as a raw device (not as a file on a file system). See also virtual disk.

preallocated disk
A type of virtual disk in which all disk space for the virtual machine is allocated at the time the disk is created.

R
RDM (raw device mapping)
A mechanism that enables a virtual machine to have direct access to a LUN on the physical storage subsystem (Fibre Channel or iSCSI only). At the same time, the virtual machine has access to the disk that is using a mapping file in the VMFS name space.

S
SDK (software development kit)
A set of tools for programmers who are developing software for a particular platform. A VMware SDK might include an API, an IDL, client stubs, sample code, and documentation.

server
(1) A system capable of managing and running virtual machines. (2) A process capable of accepting and executing instructions from another process.

SMI-S (Storage Management Initiative-Specification)
A standard containing the CIM profile (IBVP) used by the VMware Migration Server.
See also CIM (Common Information Model), IBVP (In-Band Virtualizer Profile).

SNIA (Storage Networking Industry Association)
The trade organization responsible for the SMI-S profiles.
See also SMI-S (Storage Management Initiative-Specification).

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

stub
A local procedure that implements the client side of a remote procedure call. The client calls the stub to perform a task. The stub packages the parameters, sends them over the network to the server, and returns the results to the client.
**UUID (Universally Unique Identifier)**
A number used to uniquely identify some object or entity. The UUID is either assigned by VMware vSphere (in the case of virtual machines) or is hardware-assigned (in the case of SCSI LUNs). vCenter Server tries to ensure that the UUIDs of all virtual machines being managed are unique, changing the UUIDs of conflicting virtual machines if necessary.

**Virtual Disk**
A file or set of files that appears as a physical disk drive to a guest operating system. These files can be on the host machine or on a remote file system.
See also growable disk, physical disk, preallocated disk.

**Virtual Machine**
A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. Multiple virtual machines can operate on the same host system concurrently.

See the guidelines for using the acronym VM in place of virtual machine.

**VMFS (Virtual Machine File System)**
A file system that is optimized for storing virtual machines. One VMFS partition is supported per SCSI storage device or LUN. 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 (a subset of SGML) designed for structured Web documents. XML enables users to define their own tags that encode information with meaningful structure and semantics.
Index

A
associations
  traversing 25

B
"Block Server" property value 40, 41
BlockSize property 45, 49, 54

C
Certificates, Installing 15, 16
CIM schema 7
  DMTF 19
  VMware 22, 24, 40
CIM server, starting 16, 17
CIM_prefix 24
"CIM_ProtocolControllerAccessesUnit" parameter value 48
"CIM_SCSIProtocolController" parameter value 42, 47
"CIM_StorageVolume" parameter value 54
"CIM_SystemDevice" parameter value 42, 47, 54
CIMOM 11, 12, 17
CIM-XML 15, 16
client 12, 13
client toolkits 13
clients 7, 15, 17, 24
configuration, management appliance 13, 17, 71, 72
conventions
  illustrations 25
  text 24
credential store 13

D
Dedicated property 40, 41
determining the amount of physical storage space in use 46
DeviceID property 53
directly attached disk, shared 47
disks
  physical 30, 47, 49, 50
  virtual 27, 28, 30, 43, 44, 45, 46
Distributed Management Task Force 8
DMTF CIM schema 19

E
ElementName property 48, 55
extrinsic methods 20

F
features 7
  Fibre Channel 49, 50
  Fibre Channel SCSI controllers 55
firewall 15
flat virtual disks 45

H
HBAs, virtual 42, 43
host bus adapters, virtual 42, 43, 44
hosts 40
HTTP 12, 15, 17

I
IDL 19
indications, CIM 15
industry standards
  relationship of VMware CIM SDK 19
  SMI-S 20
InstanceID property 46
intrinsic methods 20, 26

L
local storage 27, 28
LUNs 30, 49, 50, 51, 52
  shared 34
LUNs, shared 34, 49

M
management appliance 11, 12, 13, 17
methods, extrinsic 20
methods, intrinsic 20, 26
MOF 11
multipath SAN environment 37

N
Name property 48
namespaces 15, 18
new features 7
NumberOfBlocks property 45, 49, 54

O
OpenPegasus 8, 17
OVF 13

P
Pegasus 8, 12