Contents

About This Book  5
  Revision History  5
  Intended Audience  6
  Document Feedback  6
  Technical Support and Education Resources  6

1 Introduction  7
  Platform Product Support  7
  Protocol and Version Support  7
    CIM Version  8
    SMASH Version  8
    Profiles Supported  8
    CIM and SMASH Resources Online  9

2 Developing Client Applications  11
  Namespaces  11
  Locate a Server with SLP  12
  Make a Connection to the CIMOM  13
  List Registered Profiles  14
  Identify the Server Scoping Instance  15

3 Using the CIM Object Space  17
  Report Manufacturer, Model, and Serial Number  17
  Report Manufacturer, Model, and Serial Number Using Only the Implementation Namespace  19
  Report the BIOS Version  20
  Report State of All Sensors  21
  Report State of All Sensors Using Only the Implementation Namespace  23
  Report Fan Redundancy  24
  Report CPU Cores and Threads  26
  Report RAID Controller State  29
  Report State of RAID Connections  31
  Report Available Storage Extents  32
  Reboot the Managed Server  34
  Subscribe to Indications  35
Appendix: Troubleshooting Connections  37
  Connections from Client to CIM Server  37
    Using SLP  37
    Using a Web Browser  37
    Using wbemcli  37
    Verifying User Authentication Credentials  37
    Rebooting the Server  37
    Using Correct Client Samples  38
    Using other CIM Client Libraries  38
    Using the WS-Management Library  38
  Connections from CIM Server to Indication Consumer  38
    Opening the Firewall  38

Glossary  39

Index  43
About This Book

This book, the CIM SMASH/Server Management API Programming Guide, contains information about the VMware(R) implementation of System Management Architecture for Server Hardware (SMASH), an industry standard for managing server hardware. This book describes the SMASH profiles implemented by VMware and contains suggestions for using the Common Information Model (CIM) classes to accomplish common use cases.

Revision History

This book, the CIM SMASH/Server Management API Programming Guide, 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</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20071210</td>
<td>ESX Server 3i version 3.5 release.</td>
</tr>
<tr>
<td>20080409</td>
<td>ESX Server 3i version 3.5 Update 1 release.</td>
</tr>
<tr>
<td></td>
<td>Changed title (formerly CIM SMASH API Programming Guide)</td>
</tr>
<tr>
<td></td>
<td>Updated URLs.</td>
</tr>
<tr>
<td></td>
<td>Removed List of Tables.</td>
</tr>
<tr>
<td></td>
<td>Added Physical Asset profile; listed properties for all profiles.</td>
</tr>
<tr>
<td></td>
<td>Updated ElementName of Base Server registered profile.</td>
</tr>
<tr>
<td></td>
<td>Added SMI-S RAID Controller profile.</td>
</tr>
<tr>
<td></td>
<td>Divided chapter 2 into 2 parts, and expanded introductory material.</td>
</tr>
<tr>
<td></td>
<td>Corrected typographical errors.</td>
</tr>
<tr>
<td></td>
<td>Added some illustrations.</td>
</tr>
<tr>
<td>20080703</td>
<td>ESX Server 3.5 Update 2 and ESX Server 3i version 3.5 Update 2 release.</td>
</tr>
<tr>
<td></td>
<td>Replace instance diagrams with expanded versions.</td>
</tr>
<tr>
<td></td>
<td>Add use case for CPU core &amp; threading model.</td>
</tr>
<tr>
<td></td>
<td>Add use case for fan redundancy.</td>
</tr>
<tr>
<td></td>
<td>Add use cases for Host Hardware RAID Controller.</td>
</tr>
<tr>
<td></td>
<td>Added appendix about troubleshooting connections.</td>
</tr>
<tr>
<td></td>
<td>Replaced Profile Reference appendix with a URL.</td>
</tr>
<tr>
<td></td>
<td>Listed indications supported.</td>
</tr>
<tr>
<td></td>
<td>Added ESX Server 3.5.</td>
</tr>
</tbody>
</table>

To view the current version of this guide, go to http://www.vmware.com/support/pubs/sdk_pubs.html.
Intended Audience

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

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 manuals, go to:

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

Online Support

You can submit questions or post comments to the Developer Community: SDKs and APIs forum, which is monitored by VMware technical support and product teams. To access the forum, go to:

http://communities.vmware.com/community/developer

Support Offerings

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/.
VMware ESX Server 3.5 and ESX Server 3i version 3.5 include a CIM Object Manager (CIMOM) that implements a set of server discovery and monitoring features compatible with the SMASH standard. The VMware CIM SMASH/Server Management API allows clients using industry-standard protocols to do the following:

- Enumerate system resources
- Monitor system health data
- Power off host systems for maintenance

The VMware implementation of the SMASH standard leverages the open-source implementation of the Open Management with CIM (OMC) project. OMC is a collaborative effort bringing together developers from a variety of organizations to provide tools and software infrastructure for hardware vendors and others who need a reliable implementation of the Distributed Management Task Force (DMTF) management profiles.

This chapter contains the following topics:

- “Platform Product Support” on page 7
- “Protocol and Version Support” on page 7

### Platform Product Support

The VMware CIM SMASH/Server Management API is supported by ESX Server 3.5 and ESX Server 3i version 3.5. Hardware compatibility for ESX Server is documented in the hardware compatibility guides, available on the VMware Web site. See http://www.vmware.com/support/pubs/vi_pubs.html.

### Protocol and Version Support

The VMware CIM SMASH/Server Management API supports the following protocols:

- CIM-XML over HTTP
- WS-Management
- SLP
CIM Version
The CIM standard is an object model maintained by the DMTF, a consortium of leading hardware and software vendors. This release of ESX Server 3.5 and ESX Server 3i is compatible with version 2.17 (experimental) of the CIM schema.

SMASH Version
The SMASH standard is maintained by the Server Management Working Group (SMWG) of the DMTF. This release of ESX Server 3.5 and ESX Server 3i is compatible with version 1.0.0 of the SMASH standard.

Profiles Supported
The VMware CIM SMASH/Server Management API supports a number of profiles defined by the SMWG. These profiles have overlapping structures, and can be used individually (sometimes) or in combinations to manage a server.

This VMware CIM implementation also includes the Host Hardware RAID Controller profile from the SMI specification developed by SNIA. The implementation uses SMI-S version 1.2.

It's important to be aware of which version of a profile the CIM Object Manager (CIMOM) supports. The following table shows the version of each profile that is implemented by the VMware CIM SMASH/Server Management API for ESX Server 3.5 and ESX Server 3i version 3.5.

Some profiles are only partially implemented by VMware. The implementation does not include all mandatory elements specified in the profile. These profiles are listed with "N/A" in the Version column. For information about which elements are implemented, see http://www.vmware.com/support/developer/cim-sdk/smash/u2/ga/profiledoc/.

Table 1-1. Profile Versions

<table>
<thead>
<tr>
<th>Profile</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Server</td>
<td>1.0.0a</td>
</tr>
<tr>
<td>CPU</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Ethernet Port</td>
<td>N/A</td>
</tr>
<tr>
<td>Fan</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Host Hardware RAID Controller</td>
<td>N/A</td>
</tr>
<tr>
<td>IP Interface</td>
<td>N/A</td>
</tr>
<tr>
<td>Physical Asset</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Power State Management</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Power Supply</td>
<td>1.0.0c</td>
</tr>
<tr>
<td>Profile Registration</td>
<td>1.0.0a</td>
</tr>
<tr>
<td>Record Log</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Role Based Authorization</td>
<td>N/A</td>
</tr>
<tr>
<td>Sensors</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Simple Identity Management</td>
<td>N/A</td>
</tr>
<tr>
<td>Software Inventory</td>
<td>1.0.0</td>
</tr>
<tr>
<td>System Memory</td>
<td>1.0.0f</td>
</tr>
</tbody>
</table>
CIM and SMASH Resources Online

The following resources related to the CIM, SMASH, and SMI standards are available on the World Wide Web:

- http://www.dmtf.org (DMTF Home Page)
- http://www.dmtf.org/standards/cim/ (CIM Standards)
- http://www.snia.org/ (SNIA Home Page)
This chapter provides basic information about writing a CIM client for VMware ESX Server 3.5 and ESX Server 3i version 3.5, presents details of namespaces used by the VMware CIMOM, and presents an outline for writing a simple CIM client. The outline is divided into several steps, illustrated with pseudocode.

The following topics are covered:
- “Namespaces” on page 11
- “Locate a Server with SLP” on page 12
- “Make a Connection to the CIMOM” on page 13
- “List Registered Profiles” on page 14
- “Identify the Server Scoping Instance” on page 15

The CIM client outline presents a recommended general approach to accessing the CIM objects. This approach assumes no advance knowledge of the CIM implementation. If your client is aware of items such as the Service URL and the namespaces used in the VMware implementation, see “Using the CIM Object Space” on page 17 for more information about specific objects in the Implementation namespace.

### Namespaces

To access a CIM object directly, you need to know the namespace in which the object is stored. A managed server can have several CIM namespaces. This document uses the Interop namespace and the Implementation namespace.

Most CIM objects are stored in the Implementation namespace. If you know the URL and the Implementation namespace in advance, you can enumerate objects directly by connecting to that namespace.

The Interop namespace contains a few CIM objects, particularly instances of `CIM_RegisteredProfile`. One of these instances exists for each CIM profile that is fully implemented on the managed server.

`CIM_RegisteredProfile` acts as a repository of information that can be used to identify and access objects in the Implementation namespace. For each registered CIM profile there is an association that you can follow to move from the Interop namespace to the Implementation namespace.

However, the Interop namespace does not contain instances of `CIM_RegisteredProfile` for profiles that are only partially implemented — that is, not all the mandatory properties and methods are present in the VMware implementation. To access unregistered profiles, you need to know the Implementation namespace.
Namespaces can be hard-coded in the client, or they can be obtained from Service Location Protocol (SLP). The following table lists the namespaces used by ESX Server and ESX Server 3i.

Table 2-1. ESX Server and ESX Server 3i Namespaces

<table>
<thead>
<tr>
<th></th>
<th>Interop Namespace</th>
<th>Implementation Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESX Server</td>
<td>root/PG_interop</td>
<td>root/cimv2</td>
</tr>
<tr>
<td>ESX Server 3i</td>
<td>root/interop</td>
<td>root/cimv2</td>
</tr>
</tbody>
</table>

Both the Interop namespace and the Implementation namespace for your managed server can be obtained from SLP. The Interop namespace is more convenient to identify than the Implementation namespace.

The recommended approach described in this document is to use SLP to obtain the Interop namespace and the URL to enumerate CIM_RegisteredProfile, then move to the Scoping Instance of the Base Server profile in the Implementation namespace. The Scoping Instance represents the managed server and is associated with many other objects in the Implementation namespace, providing a reliable point from which to navigate to CIM objects representing any part of the managed server.

**Locate a Server with SLP**

If you do not know the URL to access the WBEM service of the CIMOM on the ESX Server 3i machine, or if you do not know the namespace, use SLP to discover the service and the namespace before your client makes a connection to the CIMOM.

SLP-compliant services attached to the same Ethernet switch respond to a client SLP query with a Service URL and a list of service attributes. The Service URL returned by the WBEM service begins with the service type `service:wbem:https://` and follows with the domain name and port number to connect to the CIMOM.

Among the attributes returned to the client is `InteropSchemaNamespace`. The value of this attribute is the name of the Interop namespace. This namespace is used for most client connections to the CIMOM, as described in “Make a Connection to the CIMOM” on page 13.

For more information about SLP, see the following links:

Make a Connection to the CIMOM

1 Collect the connection parameters from the environment.

```python
use os

function parse_environment()
    ///Check if all parameters are set in the shell environment.///
    VI_SERVER = VI_USERNAME = VI_PASSWORD = VI_NAMESPACE=Null
    ///Any missing environment variable is cause to revert to command-line arguments.///
    try
        return { 'VI_SERVER':os.environ['VI_SERVER'],
                'VI_USERNAME':os.environ['VI_USERNAME'],
                'VI_PASSWORD':os.environ['VI_PASSWORD'],
                'VI_NAMESPACE':os.environ['VI_NAMESPACE'] }
    catch
        return Null

use sys

function get_params()
    ///Check if parameters are passed on the command line.///
    param_host = param_user = param_password = param_namespace = Null
    if len( sys.argv ) == 5
        print 'Connect using command-line parameters.'
        param_host, param_user, param_password, param_namespace = sys.argv[1:5]
        return { 'host':param_host,
                 'user':param_user,
                 'password':param_password,
                 'namespace':param_namespace }
    env = parse_environment()
    if env
        print 'Connect using environment variables.'
        return { 'host':env['VI_SERVER'],
                 'user':env['VI_USERNAME'],
                 'password':env['VI_PASSWORD'],
                 'namespace':env['VI_NAMESPACE'] }
    else
        print 'Usage: ' + sys.argv[0] + ' <host> <user> <password> [<namespace>]'
        print '  or set environment variables: VI_SERVER, VI_USERNAME, VI_NAMESPACE'
        return Null

params = get_params()
if params is Null
    exit(-1)
```

2 Create the connection object in the client.

```python
use wbemlib
connection = Null

function connect_to_host( params )
    ///Connect to the server.///
    connection = wbemlib.WBEMConnection( 'https://' + params['host'],
                                          ( params['user'], params['password'] ),
                                          params['namespace'] )
    return connection

if connect_to_host( params )
    print 'Connected to: ' + params['host'] + ' as user: ' + params['user']
else
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
```

With some client libraries, the previous code creates a connection object in the client but does not send a request to the CIMOM. A request is not sent until a method is called. To verify that such a client can connect to and authenticate with the server, continue to another use case, such as “List Registered Profiles” on page 14.
List Registered Profiles

VMware recommends that CIM clients list the registered profiles before trying to use them for other purposes. If a profile is not present in the registration list (CIM_RegisteredProfile), it usually means that the profile is not implemented or is incompletely implemented.

SMASH profiles are registered in the Interop namespace, even when they are implemented in a different namespace (the Implementation namespace). A client exploring the CIM objects on the managed server can use the associations to move from CIM_RegisteredProfile to the objects in the Implementation namespace.

The CIM_RegisteredProfile class is instantiated with subclasses representing the profiles that are registered in the Interop namespace. Each instance represents a profile that is fully implemented in the Implementation namespace. Figure 2-1 shows a few instances of CIM_RegisteredProfile subclasses.

Figure 2-1. Registered Profile Subclasses in Interop Namespace

The following pseudocode shows one way to identify the profiles registered on the managed server. The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13.

```
1 Connect to the server URL, using the Interop namespace.

use wbemlib
use sys
use connection renamed cnx
connection = Null
params = cnx.get_params()
params['namespace'] = 'root/interop'
if params is Null
    exit(-1)
connection = cnx.connect_to_host( params )
if connection is Null
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
```
2 Enumerate instances of CIM RegisteredProfile.

```python
function get_registered_profile_names( connection )
    ///Get instances of RegisteredProfile.///
    instance_names = connection.EnumerateInstanceNames ( 'CIM_RegisteredProfile' )
    if instance_names is Null
        print 'Failed to enumerate RegisteredProfile.'
        return Null
    else
        return instance_names

instance_names = get_registered_profile_names( connection )
if instance_names is Null
    exit(-1)
```

3 For each instance of CIM RegisteredProfile, print the name and version of the profile.

```python
function print_profile( instance )
    print '
' + ' [' + instance.classname + ']'
    for prop in ( 'RegisteredName', 'RegisteredVersion' )
        print ' %30s = %s' % ( prop, instance[prop] )

for instance_name in instance_names
    instance = connection.GetInstance( instance_name )
    print_profile( instance )
```

**Identify the Server Scoping Instance**

The Scoping Instance of CIM ComputerSystem is an object representing the managed server. Various hardware and software components of the managed server are represented by CIM objects associated with the Scoping Instance.

A client can locate CIM objects using either of the following two ways:

- Enumerate instances in the Implementation namespace, then filter the results by their property values. This requires specific knowledge of the Implementation namespace and the subclassing used by the SMASH implementation on the managed server.
- Locate the Scoping Instance representing the managed server, then traverse selected association objects to find the desired components. This requires less knowledge of the implementation.

This topic describes how to locate the Scoping Instance representing the managed server. Figure 2-2 shows the association between the profile registration instance in the Interop namespace and the Scoping Instance in the Implementation namespace.

**Figure 2-2. Scoping Instance Associated with Profile Registration**

```
CIM_RegisteredProfile::
  OMC_RegisteredBaseServerProfile

root/interop
----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----

root/cimv2

CIM_ComputerSystem::
  OMC_UnitaryComputerSystem

CIM_ElementConformsToProfile::
  OMC_ElementConformsToBaseServerProfile

ConformantStandard
ManagedElement
```

The following pseudocode shows how to traverse the association to arrive at the Scoping Instance. The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13.
1 Make a connection to the CIMOM, using the Interop namespace.

```python
use wbemlib
use sys
use connection renamed cnx
connection = Null

params = cnx.get_params()
params['namespace'] = 'root/interop'
if params is Null
    sys.exit(-1)
connection = cnx.connect_to_host( params )
if connection is Null
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
    sys.exit(-1)
```

2 Enumerate instances of CIM_RegisteredProfile.

```python
use registered_profiles renamed prof

profile_instance_names = prof.get_registered_profile_names( connection )
if profile_instance_names is Null
    return Null
```

3 Select the instance corresponding to the Base Server profile.

```python
function isolate_base_server_registration( connection, instance_names )
    ///Isolate the Base Server registration.///
    for instance_name in instance_names
        instance = connection.GetInstance( instance_name )
        if instance ['RegisteredName'] == 'Base Server'
            return instance_name
    return Null

profile_instance_name = isolate_base_server_registration( profile_instance_names )
if profile_instance_name is Null
    print 'Base Server profile is not registered in namespace ' + namespace
```

4 Traverse the CIM_ElementConformsToProfile association to reach the Scoping Instance.

```python
function associate_to_scoping_instance( connection, profile_name )
    ///Follow ElementConformsToProfile from RegisteredProfile to ComputerSystem.///
    instance_names = connection.AssociatorNames ( profile_name, 
        AssocClass = 'CIM_ElementConformsToProfile', 
        ResultRole = 'ManagedElement' )
    if len( instance_names ) > 1
        print 'Error: %d Scoping Instances found.' % len( instance_names )
        sys.exit(-1)
    return instance_names.pop()  

function print_instance( instance )
    print '
    [' + instance.classname + ']
    for prop in instance.keys()
        print '%30s = %s' % ( prop, instance[prop] )

scoping_instance_name = associate_to_scoping_instance( profile_instance_name )
if scoping_instance_name is Null
    print 'Failed to find Scoping Instance.'
    sys.exit(-1)
else
    print_instance( connection.GetInstance( scoping_instance_name ) )
```
This chapter presents examples that show how you can use the CIM object space to get information and manage a server that runs VMware ESX Server 3.5 or ESX Server 3i version 3.5. These examples are chosen primarily to explain features of the VMware implementation of the profiles, and secondarily to demonstrate common operations.

The following examples are included:

- “Report Manufacturer, Model, and Serial Number” on page 17
- “Report Manufacturer, Model, and Serial Number Using Only the Implementation Namespace” on page 19
- “Report the BIOS Version” on page 20
- “Report State of All Sensors” on page 21
- “Report State of All Sensors Using Only the Implementation Namespace” on page 23
- “Report Fan Redundancy” on page 24
- “Report CPU Cores and Threads” on page 26
- “Report RAID Controller State” on page 29
- “Report State of RAID Connections” on page 31
- “Report Available Storage Extents” on page 32
- “Reboot the Managed Server” on page 34
- “Subscribe to Indications” on page 35

Each example describes a goal to accomplish, steps to accomplish the goal, and a few lines of pseudocode to demonstrate the steps used in the client. Many of the examples build on the basic steps described in “Developing Client Applications” on page 11.

**Report Manufacturer, Model, and Serial Number**

Taking an inventory of systems in your datacenter can be a first step to monitoring the status of the servers. You can also store the inventory data for future use in monitoring configuration changes.

This section shows how to get the physical identifying information from the Interop namespace, traversing associations to the CIM_PhysicalPackage for the Scoping Instance. Figure 3-1 shows the relationships of the CIM objects involved.
Figure 3-1. Locating Physical Package Information from the Scoping Instance

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.

1. Connect to the server URL, using the Interop namespace.
   ```python
   use wbemlib
   use sys
   use connection renamed cnx
   connection = Null

   params = cnx.get_params()
   params['namespace'] = 'root/interop'
   if params is Null
       sys.exit(-1)
   connection = cnx.connect_to_host( params )
   if connection is Null
       print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
       sys.exit(-1)
   ```

2. Locate the Scoping Instance of CIM_ComputerSystem.
   ```python
   use scoping_instance renamed si

   scoping_instance_name = si.get_scoping_instance_name( connection )
   if scoping_instance_name is Null
       print 'Failed to find Scoping Instance.'
       sys.exit(-1)
   ```

3. Traverse the CIM_ComputerSystemPackage association to reach the CIM_PhysicalPackage instance corresponding to the managed server.
   ```python
   instance_names = connection.AssociatorNames( scoping_instance_name, \       AssocClass = 'CIM_ComputerSystemPackage', \       ResultRole = 'Antecedent' )
   if len( instance_names ) &gt; 1
       print 'Error: %d Physical Packages found for Scoping Instance.' % len( instance_names )
       sys.exit(-1)
   ```
4. Print the Manufacturer, Model, and SerialNumber properties.

```python
print '\n' + '[' + instance.classname + ']' + ' ='
print '\n' + object_type + ' [' + instance.classname + '] ->
for prop in [ 'Manufacturer', 'Model', 'SerialNumber' ]
  print ' %30s = %s' % ( prop, instance[prop] )
```

A sample of output looks like the following:

```
Connecting to: server as user: admin
CIM_PhysicalPackage [OMC_Chassis] ->
  Manufacturer = Dell Inc.
  Model = PowerEdge 1900
  SerialNumber = GYZ41D1
```

Report Manufacturer, Model, and Serial Number Using Only the Implementation Namespace

Taking an inventory of systems in your datacenter can be a first step to monitoring the status of the servers. You could also store the inventory data for future use in monitoring configuration changes.

This section shows how to get the physical identifying information from the Implementation namespace, enumerating and filtering CIM_PhysicalPackage for the Scoping Instance. This way is convenient when the namespace is known in advance.

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.

1. Connect to the server URL, using the Implementation namespace, supplied as a parameter. (The actual namespace you will use depends on your installation.)

```python
use wbemlib
use sys
use connection renamed cnx
connection = Null
params = cnx.get_params()
if params is Null
  sys.exit(-1)
connection = cnx.connect_to_host( params )
if connection is Null
  print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
  sys.exit(-1)
```

2. Use the EnumerateInstances method to get all the CIM_PhysicalPackage instances on the server.

```python
object_type = 'CIM_PhysicalPackage'
list = connection.EnumerateInstances( object_type )
```

3. Select the instance whose ElementName property has the value "Chassis", and print the Manufacturer, Model, and SerialNumber properties of the instance.

```python
function print_info( instance )
  print '\n' + object_type + ' [' + instance.classname + '] ->
  for prop in [ 'Manufacturer', 'Model', 'SerialNumber' ]
    print ' %30s = %s' % ( prop, instance[prop] )
  if instance[ 'ElementName' ] == 'Chassis'
    print_info( instance )
sys.exit(0)
print 'Unable to find Physical Package instance for Chassis.'
```
A sample of output looks like the following:

```
Connecting to: server as user: admin
CIM_PhysicalPackage [OMC_Chassis] ->
  Manufacturer = Dell Inc.
  Model = PowerEdge 1900
  SerialNumber = GYZ41D1
```

### Report the BIOS Version

Users might want to query the BIOS version of the managed server as part of routine maintenance by system administrators.

This section shows how to get the BIOS version string by traversing the `CIM_InstalledSoftwareIdentity` association from the server Scoping Instance. VMware does not implement the `CIM_ElementSoftwareIdentity` association, so you need to use `CIM_InstalledSoftwareIdentity` instead. Figure 3-2 shows the relationships of the CIM objects involved.

**Figure 3-2. Locating the BIOS Version from the Scoping Instance**

The VMware implementation of `CIM_SoftwareIdentity` makes the version available in the `VersionString` property rather than the `MajorVersion` and `MinorVersion` properties.

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.
1  Connect to the server URL, using the Interop namespace.
    use wbemlib
    use sys
    use connection renamed cnx
    use registered_profiles renamed prof
    connection = Null

    params = cnx.get_params()
    params[\'namespace\'] = 'root/interop'
    if params is Null
        sys.exit(-1)
    connection = cnx.connect_to_host( params )
    if connection is Null
        print 'Failed to connect to: ' + params[\'host\'] + ' as user: ' + params[\'user\']
        sys.exit(-1)

2  Locate the Scoping Instance of the managed server.

    use scoping_instance renamed si

    scoping_instance_name = si.get_scoping_instance_name( connection )
    if scoping_instance_name is Null
        print 'Failed to find server Scoping Instance.'
        sys.exit(-1)

3  Traverse the CIM_InstalledSoftwareIdentity association to reach the CIM_SoftwareIdentity instances corresponding to the software on the managed server.

    instance_names = connection.Associators( scoping_instance_name, \
        AssocClass = 'CIM_InstalledSoftwareIdentity', \
        ResultRole = 'InstalledSoftware' )

4  Select the CIM_SoftwareIdentity instance representing the BIOS of the managed server, then print the Manufacturer and VersionString properties.

    function print_info( instance )
        print '\n' + '[' + instance.classname + '] ='
        print '\n' + object_type + '[' + instance.classname + ']' + '&gt;' + prop + ' = %s' % (prop, instance[prop])

    for instance in instances
        if instance[\'Name\'] == 'System BIOS'
            print_info( connection.GetInstance( instance_name ) )

---

**Report State of All Sensors**

This information is useful to system administrators who need to monitor system health. This section shows how to locate system sensors, report their current states, and flag any sensors with abnormal states.

The example uses only CIM_NumericSensor instances for simplicity. Discrete sensors can be queried as well by substituting CIM_Sensor for CIM_NumericSensor. Determining what values constitute normal sensor state is hardware-dependent.

This section shows how to get the sensor states by starting from the Interop namespace and traversing associations from the managed server Scoping Instance. Figure 3-3 shows the relationships of the CIM objects involved. For information about getting sensor states using only the Implementation namespace, see “Report State of All Sensors Using Only the Implementation Namespace” on page 23.
The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.

1. Connect to the server URL, using the Interop namespace.

   ```python
   use wbemlib
   use sys
   use connection renamed cnx
   connection = Null
   params = cnx.get_params()
   params['namespace'] = 'root/interop'
   if params is Null
       sys.exit(-1)
   connection = cnx.connect_to_host( params )
   if connection is Null
       print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
       sys.exit(-1)
   ```

2. Locate the Scoping Instance of CIM_ComputerSystem.

   ```python
   use scoping_instance renamed si
   scoping_instance_name = si.get_scoping_instance_name( connection )
   if scoping_instance_name is Null
       print 'Failed to find server Scoping Instance.'
       sys.exit(-1)
   ```

3. Traverse the CIM_SystemDevice association to reach the CIM_NumericSensor instances on the managed server.

   ```python
   target_class = 'CIM_NumericSensor'
   instances = connection.Associators( scoping_instance_name, 
       AssocClass = 'CIM_SystemDevice', 
       ResultClass = target_class )
   if len( instances ) is 0
       print 'Error: No sensors associated with server Scoping Instance.'
       sys.exit(-1)
   ```
For each sensor instance, print the ElementName and CurrentState properties. You can flag any abnormal values you find. Abnormal values depend on the sensor type and its PossibleStates property.

```python
function print_info( instance )
    print \n' + target_class + ' [\' + instance.classname + \'] ='
    if instance['CurrentState'] != 'Normal'
        print '********* SENSOR STATE WARNING *********
    for prop in [ 'ElementName', 'CurrentState' ]
        print ' %30s = %s' % ( prop, instance[prop] )

for instance in instances
    print_info( instance )
```

A sample of output looks like the following:

```
CIM_NumericSensor [OMC_NumericSensor] =
    ElementName = FAN 1 RPM for System Board 1
    CurrentState = Normal

CIM_NumericSensor [OMC_NumericSensor] =
    ElementName = Ambient Temp for System Board 1
    CurrentState = Normal
```

Report State of All Sensors Using Only the Implementation Namespace

This information is useful to system administrators who need to monitor system health. This topic shows how to locate system sensors, report their current states, and flag any sensors with abnormal states.

The example uses only CIM_NumericSensor instances for simplicity. Discrete sensors can be queried as well by substituting CIM_Sensor for CIM_NumericSensor. Determining what values constitute normal sensor state is hardware-dependent.

This topic shows how to get the sensor states from the Implementation namespace, assuming you already know its name. For information about getting sensor state using the standard Interop namespace, see “Report State of All Sensors” on page 21.

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13.

1. Connect to the server URL, using the Implementation namespace, supplied as a parameter. (The actual namespace you will use depends on your installation.)

   ```python
   use wbemlib
   use sys
   use connection renamed cnx
   connection = Null
   
   params = cnx.get_params()
   if params is Null
       sys.exit(-1)
   connection = cnx.connect_to_host( params )
   if connection is Null
       print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
       sys.exit(-1)
   ```

2. Enumerate instances of CIM_NumericSensor.

   ```python
   target_class = 'CIM_NumericSensor'
   instances = connection.EnumerateInstances( target_class )
   if len( instances ) is 0
       print 'Error: No sensors found on managed server.'
       sys.exit(-1)
   ```
3 Iterate over the sensor instances, printing the properties ElementName and CurrentState.

```python
function print_info( instance )
    print '\n' + target_class + ' [' + instance.classname + '] ='
    if instance['CurrentState'] != 'Normal'
        print '********* SENSOR STATE WARNING *********
    for prop in [ 'ElementName', 'CurrentState' ]
        print ' %30s = %s' % ( prop, instance[prop] )
    for instance in instances
        print_info( instance )
```

A sample of output looks like the following:

```
CIM_NumericSensor [OMC_NumericSensor] =
    ElementName = FAN 1 RPM for System Board 1
    CurrentState = Normal
CIM_NumericSensor [OMC_NumericSensor] =
    ElementName = Ambient Temp for System Board 1
    CurrentState = Normal
```

### Report Fan Redundancy

This information is useful to system administrators who need to monitor system health. This section shows how to locate system fans and query the CIMOM for redundant fan relationships.

This section shows how to enumerate the fans by starting from the Interop namespace and traversing associations from the managed server Scoping Instance. Figure 3-4 shows the relationships of the CIM objects involved. If the managed server provides redundant cooling, the redundancy is modeled in the CIMOM by an instance of CIM_RedundancySet that is associated with two (or more) redundant fans.
Figure 3-4. Locating Redundant Fans

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.

1. Connect to the server URL, using the Interop namespace.

   ```python
   use wbemlib
   use sys
   use connection renamed cnx
   connection = Null

   params = cnx.get_params()
   params['namespace'] = 'root/interop'
   if params is Null
       sys.exit(-1)
   connection = cnx.connect_to_host( params )
   if connection is Null
       print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
       sys.exit(-1)
   
   2. Locate the Scoping Instance of CIM_ComputerSystem.

      use scoping_instance renamed si

      scoping_instance_name = si.get_scoping_instance_name( connection )
      if scoping_instance_name is Null
          print 'Failed to find server Scoping Instance.'
          sys.exit(-1)
   ```
3 Traverse the CIM_SystemDevice association to reach the CIM_Fan instances on the managed server.

    target_class = 'CIM_Fan'
    fan_instances = connection.Associators( scoping_instance_name, \
        AssocClass = 'CIM_SystemDevice', \
        ResultClass = target_class )

    if len( fan_instances ) is 0
        print 'Error:  No fans associated with server Scoping Instance.'
        sys.exit(-1)

4 For each fan instance, print the ElementName and DeviceID properties.

    function print_info( instance )
        print '\n' + target_class + ' [' + instance.classname + '] ='
        for prop in [ 'ElementName', 'DeviceID' ]
            print ' %30s = %s' % ( prop, instance[prop] )

    for fan_instance in fan_instances
        print_info( fan_instance )

5 For each fan instance, traverse the CIM_MemberOfCollection association to reach any instances of CIM_RedundancySet.

    target_class = 'CIM_RedundancySet'
    set_instances = connection.Associators( scoping_instance_name, \
        AssocClass = 'CIM_MemberOfCollection', \
        ResultClass = target_class )

6 For each fan instance, print the redundancy status. If the fan is not a member of a redundancy set, the redundancy status is not applicable.

    if len( set_instances ) is 0
        print ' Redundancy status: N/A'
    else
        for instance in set_instances
            name = instance['Name']
            status = instance['RedundancyStatus']
            print ' redundancy set (%s) status = %s' % ( name, (status==2 ? 'Fully Redundant' : 'unknown or degraded')

A sample of output looks like the following:

    CIM_Fan [OMC_Fan] =
        ElementName = FAN 1 RPM
        DeviceID = 48.0.32.99
    redundancy set (117.0.32.0) status = Fully Redundant
    CIM_Fan [OMC_Fan] =
        ElementName = FAN 2 RPM
        DeviceID = 49.0.32.99
    redundancy set (117.0.32.0) status = Fully Redundant

Report CPU Cores and Threads

This information is useful to system administrators who need to monitor system health. This section shows how to enumerate the processor cores and hardware threads in a managed server.

The VMware implementation does not include instances of CIM_ProcessorCapabilities, but cores and hardware threads are modeled with individual instances of CIM_ProcessorCore and CIM_HardwareThread.

This section shows how to locate information about the CPU cores and threads by starting from the Interop namespace, traversing associations from the managed server Scoping Instance. A managed server has one or more processors, each of which has one or more cores with one or more threads. Figure 3-5 shows the relationships of the CIM objects involved. For simplicity, the diagram shows only a single processor with one core and one hardware thread.
Figure 3-5. Locating CPU Cores and Hardware Threads

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.

1. Connect to the server URL, using the Interop namespace.

```python
use wbemlib
use sys
use connection renamed cnx
connection = Null
params = cnx.get_params()
params['namespace'] = 'root/interop'
if params is Null
    sys.exit(-1)
connection = cnx.connect_to_host( params )
if connection is Null
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
sys.exit(-1)
```
2 Locate the Scoping Instance of CIM_ComputerSystem.

    use scoping_instance renamed si

    scoping_instance_name = si.get_scoping_instance_name( connection )

    if scoping_instance_name is Null
      print 'Failed to find server Scoping Instance.'
      sys.exit(-1)

3 Traverse the CIM_SystemDevice association to reach the CIM_Processor instances on the managed server.

    target_class = 'CIM_Processor'
    proc_instance_names = connection.AssociatorNames( scoping_instance_name, 
              AssocClass = 'CIM_SystemDevice', 
              ResultClass = target_class )
    
    if len( proc_instance_names ) is 0
      print 'Error:  No processors associated with server Scoping Instance.'
      sys.exit(-1)

4 For each CIM_Processor instance, print the ElementName, Family, and CurrentClockSpeed properties.

    for proc_instance_name in proc_instance_names
      instance = connection.GetInstance( proc_instance_name )
      print ' %s (Family: %s) (%sMHz)' % 
        ( instance['ElementName'], instance['Family'], instance['CurrentClockSpeed'] )

5 For each CIM_Processor instance, traverse the CIM_ConcreteComponent association to reach the CIM_ProcessorCore instances on the managed server.

    target_class = 'CIM_ProcessorCore'
    core_instance_names = connection.AssociatorNames( proc_instance_name, 
              AssocClass = 'CIM_ConcreteComponent', 
              ResultClass = target_class )
    
    if len( core_instance_names ) is 0
      print 'No processor cores associated with this CPU.'
      sys.exit(-1)

6 For each CIM_ProcessorCore instance, print the ElementName and CoreEnabledState properties.

    for core_instance_name in core_instance_names
      instance = connection.GetInstance( core_instance_name )
      print ' %s %s %s (%s)' % 
        ( instance['ElementName'], instance['CoreEnabledState']
          (instance['CoreEnabledState'] == 'Enabled')? 'Enabled' : 'Disabled' )

7 For each CIM_ProcessorCore instance, traverse the CIM_ConcreteComponent association to reach the CIM_HardwareThread instances on the managed server.

    target_class = 'CIM_HardwareThread'
    thread_instance_names = connection.AssociatorNames( core_instance_name, 
              AssocClass = 'CIM_ConcreteComponent', 
              ResultClass = target_class )
    
    if len( thread_instance_names ) is 0
      print 'No hardware threads associated with this CPU core.'
      sys.exit(-1)

8 For each CIM_HardwareThread instance, print the ElementName property.

    for thread_instance_name in thread_instance_names
      instance = connection.GetInstance( thread_instance_name )
      print ' %s' % instance['ElementName']
A sample of output looks like the following:

CPU1 (Family: 179) (2667MHz)
CPU1 Core 1 (Enabled)
CPU1 Core 1 Thread 1
CPU1 Core 2 (Enabled)
CPU1 Core 2 Thread 1
CPU2 (Family: 179) (2667MHz)
CPU2 Core 1 (Enabled)
CPU2 Core 1 Thread 1
CPU2 Core 2 (Enabled)
CPU2 Core 2 Thread 1

Report RAID Controller State

This information is useful to system administrators who need to monitor system health. This section shows how to report the health state of RAID controllers on the managed server.

You can enumerate the controllers by starting from the Interop namespace and traversing associations from the managed server Scoping Instance. Figure 3-6 shows the relationships of the CIM objects involved.

The CIM_PortController instance is logically identical to an instance of CIM_ComputerSystem subclassed as VMware_Controller. The VMware_Controller instance is the logical entity that is associated with the controller port objects.

Figure 3-6. Locating RAID Controllers

The pseudocode in this topic depends on the pseudocode in “Make a Connection to the CIMOM” on page 13 and the pseudocode in “Identify the Server Scoping Instance” on page 15.
1 Connect to the server URL, using the Interop namespace.

```python
use wbemlib
use sys
use connection renamed cnx
connection = Null
params = cnx.get_params()
params['namespace'] = 'root/interop'
if params is Null
    sys.exit(-1)
connection = cnx.connect_to_host( params )
if connection is Null
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
    sys.exit(-1)
```

2 Locate the Scoping Instance of CIM_ComputerSystem.

```python
use scoping_instance renamed si
scoping_instance_name = si.get_scoping_instance_name( connection )
if scoping_instance_name is Null
    print 'Failed to find server Scoping Instance.'
    sys.exit(-1)
```

3 Traverse the CIM_SystemDevice association to reach the CIM_PortController instances on the managed server. Use the value of the OtherControllerType property to distinguish instances of HBA ports from other port types.

```python
target_class = 'CIM_PortController'
pc_instance_names = connection.AssociatorNames( scoping_instance_name, 
    AssocClass = 'CIM_SystemDevice', 
    ResultClass = target_class )
if len( pc_instance_names ) is 0
    print 'Error:  No port controllers associated with server Scoping Instance.'
sys.exit(-1)
for instance_name in pc_instance_names
    instance = connection.GetInstance( instance_name )
    if ( instance['OtherControllerType'] != 'SAS/SATA' )
        pc_instance_names.delete( instance_name )
```

4 For each port controller instance, traverse the CIM_LogicalIdentity association to reach the matching instance of CIM_ComputerSystem (subclassed as VMware_Controller).

```python
for pc_instance_name in pc_instance_names
    target_class = 'CIM_ComputerSystem'
    controller_instance_names = connection.AssociatorNames( scoping_instance_name, 
        AssocClass = 'CIM_LogicalIdentity', 
        ResultClass = target_class )
```

5 For the resulting controller instance, print the ElementName, Name, and HealthState properties.

```python
for instance_name in controller_instance_names
    instance = connection.GetInstance( instance_name )
    print "%s (%s) health state: %s" % 
        ( instance['ElementName'], 
        instance['Name'], 
        (instance['HealthState']) ? 'OK' : 'Degraded' )
```

A sample of output looks like the following:

```
Controller 0 (SASSira) health state: OK
Controller 1 (SASSira) health state: OK
```
Report State of RAID Connections

This section shows how to report the connections of RAID controller initiators to targets on the managed server. This information is useful to system administrators who need to monitor system health.

The VMware implementation models serial-attached SCSI connections to drives that belong to pooled RAID configurations. The VMware model is similar to, but not identical to, the SMI-S Host Hardware RAID Controller profile published by the SNIA.

You can enumerate the connections of a controller by starting from the instance of CIM_ComputerSystem subclassed as VMware_Controller that represents the RAID controller. From there, you traverse associations to the related CIM_ConnectivityCollection instances. Figure 3-7 shows the relationships of the CIM objects involved. You need to do this for each disk controller on the managed server. See “Report RAID Controller State” on page 29 for information about locating the RAID controllers attached to a managed system.

**Figure 3-7.** Locating Connections Between HBA Initiators and Targets

1 From a given instance of CIM_ComputerSystem, traverse the CIM_SystemDevice association to reach the CIM_LogicalPort instances on the managed server.

```python
target_class = 'CIM_LogicalPort'
port_instance_names = connection.AssociatorNames( controller_instance_name, 
    AssocClass = 'CIM_SystemDevice', 
    ResultClass = target_class )

if len( port_instance_names ) is 0
    print 'Error: No ports associated with controller.'
sys.exit(-1)
```
2 For each logical port instance, traverse the CIM_DeviceSAPImplementation association to reach the matching instance of CIM_SCSIProtocolEndpoint.

```python
for port_instance_name in port_instance_names
    target_class = 'CIM_SCSIProtocolEndpoint'
    init_instance_names = connection.AssociatorNames( port_instance_name, 
        AssocClass = 'CIM_DeviceSAPImplementation', 
        ResultClass = target_class )
```

3 From the instance of CIM_SCSIProtocolEndpoint, traverse the CIM_MemberOfCollection association to reach the instances of CIM_ConnectivityCollection representing the connection between the initiator and its targets.

```python
for init_instance_name in init_instance_names
    target_class = 'CIM_MemberOfCollection'
    conn_instance_names = connection.AssociatorNames( init_instance_name, 
        AssocClass = 'CIM_MemberOfCollection', 
        ResultClass = target_class )
```

4 For the resulting instance of CIM_ConnectivityCollection, print the InstanceID, HealthState, and ConnectivityStatus properties.

```python
for instance_name in conn_instance_names
    instance = connection.GetInstance( instance_name )
    print "Port connection %s  health state: %s  connectivity status: %s" % (
        instance['InstanceID'], 
        (5==instance['HealthState']) ? 'OK' : '??', 
        (2==instance['ConnectivityStatus']) ? 'Up' : '??' )
```

**Report Available Storage Extents**

This section shows how to report the unused disk storage extents available to a managed server. The unused storage extents are those that do not belong to a storage pool implementing a RAID configuration. The information can be useful for configuring the managed servers in a datacenter.

This procedure assumes you have already located the SAS or SATA connections on a managed server. See “Report State of RAID Connections” on page 31 for information about locating the connections of a RAID controller to targets on the managed system.

You can locate disk storage extents by starting from each instance of CIM_ConnectivityCollection and following associations to the disk media attached to the target endpoint. Figure 3-8 shows the relationships of the CIM objects involved.
From a given instance of CIM_ConnectivityCollection, traverse the CIM_MemberOfCollection association to reach the CIM_SCSIProtocolEndpoint instances on the managed server. Use the value of the ElementName property to distinguish the target endpoints from the initiator endpoints.

```python
1F from given instance of CIM_ConnectivityCollection, traverse the CIM_MemberOfCollection association to reach the CIM_SCSIProtocolEndpoint instances on the managed server. Use the value of the ElementName property to distinguish the target endpoints from the initiator endpoints.

target_class = 'CIM_SCSIProtocolEndpoint'
targ_instance_names = connection.AssociatorNames( controller_instance_name, 
             AssocClass = 'CIM_MemberOfCollection', 
             ResultClass = target_class )

if len( targ_instance_names ) is 0
    print 'Error: No targets associated with SCSI connection instance.'
sys.exit(-1)
for instance_name in targ_instance_names
    instance = connection.GetInstance( instance_name )
    if ( instance['ElementName'] not begins 'Target' )
        targ_instance_names.delete( instance_name )

2F for each target instance, traverse the CIM_DeviceSAPImplementation association to reach the disk drive for the target.

for targ_instance_name in targ_instance_names
    target_class = 'CIM_DiskDrive'
disk_instance_names = connection.AssociatorNames( targ_instance_name, 
             AssocClass = 'CIM_DeviceSAPImplementation', 
             ResultClass = target_class )
From **CIM_DiskDrive**, traverse the **CIM_MediaPresent** association to reach the storage extents belonging to that drive. Use the value of the **ElementName** property to identify the storage extents that already belong to a storage pool.

```python
for disk_instance_name in disk_instance_names
    target_class = 'CIM_StorageExtent'
    ext_instance_names = connection.AssociatorNames( disk_instance_name, \
        AssocClass = 'CIM_MediaPresent', \
        ResultClass = target_class )
    for instance_name in ext_instance_names
        instance = connection.GetInstance( instance_name )
        if ( instance['ElementName'] not ends 'ONLINE' )
            ext_instance_names.delete( instance_name )
```

For each instance of **CIM_StorageExtent**, print the **ElementName**, **OperationalStatus**, and **DataRedundancy** properties, along with the extent size (BlockSize * NumberOfBlocks).

```python
for ext_instance_name in ext_instance_names
    instance = connection.GetInstance( ext_instance_name )
    print 'Disk extent: %s' % (instance['ElementName'])
    print ' Operational status: %s' % (instance['OperationalStatus'])
    print ' Redundancy: %s' % (instance['DataRedundancy'])
    print ' Size: %s' % (instance['BlockSize'] * instance['NumberOfBlocks'])
```

### Reboot the Managed Server

This information can be useful when you want to reset the managed server.

The pseudocode in this section depends on the pseudocode in “Make a Connection to the CIMOM” on page 13.

1. Connect to the server URL, using the Interop namespace.
   ```python
   use wbemlib
   use sys
   use connection renamed cnx
   connection = Null
   params = cnx.get_params()
   params['namespace'] = 'root/interop'
   if params is Null
       sys.exit(-1)
   connection = cnx.connect_to_host( params )
   if connection is Null
       print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
       sys.exit(-1)
   ```

2. Locate the Scoping Instance of **CIM_ComputerSystem**.
   ```python
   use scoping_instance renamed si
   scoping_instance_name = si.get_scoping_instance_name( connection )
   if scoping_instance_name is Null
       print 'Failed to find server Scoping Instance.'
       sys.exit(-1)
   ```

3. Traverse the **CIM_AssociatedPowerManagementService** association to reach the **CIM_PowerManagementService** instance on the managed server.
   ```python
   instance_names = connection.AssociatorNames( scoping_instance_name, \
       AssocClass = 'CIM_AssociatedPowerManagementService', \
       ResultRole = 'ServiceProvided' )
   if len( instance_names ) != 1
       print 'Error: Unable to find PowerManagementService.'
       sys.exit(-1)
   service_name = instance_names.pop()"
4. Invoke the `RequestPowerStateChange()` method to request power state 5 (Power Cycle).

```python
requested_state = 5
method_params = {'PowerState': requested_state, 
                 'ManagedElement': scoping_instance_name}
(error_return, output) = connection.InvokeMethod(
                      'RequestPowerStateChange', 
                      service_name, 
                      **method_params)
```

The managed server might be left in maintenance mode after power cycling.

**Subscribe to Indications**

ESX Server 3.5 and ESX Server 3i version 3.5 support the following types of indications:

<table>
<thead>
<tr>
<th>Indication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMC_IpmiAlertIndication</td>
<td>This indication is sent whenever entries are added to the IPMI System Event Log. The indication is also sent whenever a sensor's HealthState property becomes less healthy than previously seen.</td>
</tr>
<tr>
<td>VMware_AlertIndication</td>
<td>This indication is sent for events detected on the LSI Host Hardware RAID controller. All AEN data generates indications.</td>
</tr>
<tr>
<td>VMware_KernelIPChangedIndication</td>
<td>This indication is sent whenever the ESX Server kernel IP address for the host has changed.</td>
</tr>
</tbody>
</table>

To receive CIM indications, you need a running process that accepts indication messages and logs them or otherwise acts on them, depending on your application. You can use a commercial CIM indication consumer to do this. If you choose to implement your own indication consumer, see the following documents:

- CIM indication specifications from your server supplier that are specific to the server model

Whether or not you implement your own indication consumer, the indication consumer needs to operate with a known URL. This URL is used when instantiating the IndicationHandler object.

Similarly, you need to know which indication class to monitor. This information is used when instantiating the IndicationFilter object.

Given the indication consumer URL and the indication class, the following steps show how to instantiate the objects needed to register for indications.

The pseudocode in this section depends on the pseudocode in “Make a Connection to the CIMOM” on page 13.

1. Connect to the server URL, using the Interop namespace.

```python
use wbemlib
use sys
use connection renamed cnx
connection = Null

params = cnx.get_params()
params['namespace'] = 'root/interop'
if params is Null
    exit(-1)
connection = cnx.connect_to_host(params)
if connection is Null
    print 'Failed to connect to: ' + params['host'] + ' as user: ' + params['user']
```
2  Build the URL for the indication consumer.
   
   destination = 'http://' + params['consumer_host'] \\
   + ':' + params['consumerPort'] + '/indications'

3  Create the IndicationHandler instance to represent the consumer.

   handlerBindings = {
   "SystemCreationClassName" : 'OMC_UnitaryComputerSystem', \\
   "SystemName" : clientHost, \\
   "Name" : 'IndicationHandlerName', \\
   "CreationClassName" : 'CIM_IndicationHandlerCIMXML' \\
   }

   handlerName = wbemlib.CIMInstanceName( \\
   "CIM_IndicationHandlerCIMXML", \\
   keybindings=handlerBindings, \\
   namespace='root/interop' )

   handlerInst = wbemlib.CIMInstance( \\
   "CIM_IndicationHandlerCIMXML", \\
   properties = handlerBindings, \\
   path = handlerName )

   handlerInst['Destination'] = destination

   chandlerName = client.CreateInstance( handlerInst )

4  Create the IndicationFilter instance to specify the indication class (such as CIM_AlertIndication).

   filterBindings = {
   "SystemCreationClassName" : 'OMC_UnitaryComputerSystem', \\
   "SystemName" : clientHost, \\
   "Name":  'Org:Local', \\
   "CreationClassName" : 'CIM_IndicationFilter' \\
   }

   filterName = wbemlib.CIMInstanceName( \\
   "CIM_IndicationFilter", \\
   keybindings=filterBindings, \\
   namespace='root/interop' )

   filterInst = wbemlib.CIMInstance( \\
   "CIM_IndicationFilter", \\
   properties = filterBindings, \\
   path = filterName )

   filterInst['SourceNamespace'] = 'root/cimv2'

   filterInst['Query'] = 'SELECT * FROM ' + params['className']

   filterInst['QueryLanguage'] = 'WQL'

   cfilterName = client.CreateInstance( filterInst )

   Use a globally unique organization identifier in place of Org, and an organizationally unique identifier in place of Local.

5  Create the IndicationSubscription association to link the filter with the handler.

   subBindings = { 'Filter': cfilterName, \\
   'Handler' : chandlerName }

   subName = wbemlib.CIMInstanceName( \\
   'CIM_IndicationSubscription', \\
   keybindings = subBindings, \\
   namespace = 'root/interop' )

   subInst = wbemlib.CIMInstance( 'CIM_IndicationSubscription', \\
   path = subName )

   subInst['Filter'] = cfilterName

   subInst['Handler'] = chandlerName

   rsname = client.CreateInstance( subInst )
Appendix: Troubleshooting Connections

This appendix contains information that helps you diagnose and correct problems with connections between a CIM client and a CIM server or with connections between a CIM server and a process that consumes indications.

Connections from Client to CIM Server

This section deals with clients that fail to complete a connection to a CIM Server. The following suggestions help to verify the connection parameters and the health of the CIM server.

Using SLP

Check the connection parameters using an SLP client (available on the World-Wide Web). Run the SLP client on the same subnetwork as the managed server. Verify that the managed server advertises the expected CIM service and the correct URL.

Using a Web Browser

To verify that you can reach the CIM service at the advertised location, connect to the managed server with a Web browser. Use a URL of the form https://<cim-server.mydomain.com>:5989/ (substituting the name of the actual server), and verify that the server is responding on the expected port. Port 5989 is the default port for CIM connections, but your installation might be different.

Using wbemcli


Verifying User Authentication Credentials

If you are certain that the connection parameters are correct, verify the authentication parameters. To complete a connection, you need to authenticate as a user that is known to the managed server.

Connect to the managed server through the console and check that your root password is correct. Then use that password to authenticate as the root user from your client.

Rebooting the Server

If all your connection parameters are correct, but you still cannot complete a connection, reboot the managed server or restart the management agents on the server.
Using Correct Client Samples

If you are using sample clients supplied by VMware, check the documentation to be sure that the samples are intended to work with the CIM server you are trying to connect to. The samples might hard-code parameters such as port and namespace that affect the connection.

For example, the C++ code in the *CIM SDK Programming Guide for ESX Server 3.0* connects to the CIM server included with ESX Server 3.0, but does not connect to the CIM server included with ESX Server 3.5 and ESX Server 3i version 3.5.

Using other CIM Client Libraries

VMware does not test all available CIM client libraries with ESX Server 3.5 and ESX Server 3i version 3.5. If your CIM client cannot connect to the CIM server, try writing a test client for a different library.

For example, [http://sourceforge.net](http://sourceforge.net) has a number of CIM client libraries available.

Using the WS-Management Library

If you are unable to find a satisfactory client library to make a WBEM connection, use WS-Management. ESX Server 3.5 and ESX Server 3i version 3.5 include a WS-Man server to support CIM operations.

VMware recommends using the Web Services for Management Perl Library for WS-Man clients. This library is included with the VMware Infrastructure Perl Toolkit version 1.6 or higher. See [http://www.vmware.com/support/pubs/sdk_pubs.html](http://www.vmware.com/support/pubs/sdk_pubs.html) for more information about the Perl Toolkit.

Connections from CIM Server to Indication Consumer

This section deals with situations in which your client is able to connect to a CIM server and subscribe to indications, but is unable to receive indications.

Opening the Firewall

ESX Server 3.5 ships with a software firewall that is configured by default to block outgoing connection requests. When an indication is triggered, the producer cannot open a connection to the consumer unless the target port is opened in the firewall.

This problem does not apply to ESX Server 3i version 3.5, which does not have a firewall for outgoing connections.

To open an outgoing port in the firewall

In the service console, use the following command to open a port for an HTTP connection between the indication producer and the indication consumer:

```
esxcfg-firewall -o <port_number>,tcp,out,http
```

To close an outgoing port in the firewall

In the service console, use the following command to close an outgoing HTTP port in the firewall:

```
esxcfg-firewall -c <port-number>,tcp,out,http
```
Glossary

A  API (application programming interface)
   A set of functions that allows you to access a service programmatically.

B  BIOS (basic input/output system)
   Firmware that controls machine startup and manages communication between the CPU and other devices, such as the keyboard, monitor, printers, and disk drives.

C  CIM (Common Information Model)
   A collection of standards created by the DMTF to provide a shared model for managing systems. A set of object-oriented schemas, defined by the DMTF (Distributed Management Task Force), that provides a shared model for managing systems. The CIM schemas are 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 service that provides standard CIM management functions over a WBEM connection to a CIM client. A CIMOM stores class definitions and populates requests for CIM operations with information returned from specific data providers.
   See also CIM (Common Information Model).

   CIM-XML
   A WBEM protocol defined by the DMTF for transporting XML over HTTP.
   See also DMTF (Distributed Management Task Force).

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

E  extrinsic method
   A CIM method that is defined only for a specific CIM class.
   See also intrinsic method.

F  FRU (field replaceable unit)
   A component that can be removed and replaced at the installed site.

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 API (application programming interface), stub.
**Implementation namespace**
A logical location that contains most of the CIM classes and objects. The name is generally specific to the organization that supplies the CIM implementation, such as acme/cimv2. The name of the Implementation namespace can be discovered from associations with the Interop namespace.

**indication**
In the context of a system modeled by CIM, an asynchronous notification of an event, such as a state change or the deletion of an object monitored on behalf of a CIM client.

**indication consumer**
A process that receives CIM indication messages on a known port. In some systems, the consumer might be the same client process that subscribed to the indications.
See also indication.

**indication producer**
A process that creates and sends CIM indication messages to report events for which one or more CIM clients have subscribed.
See also indication.

**initiator**
In the context of storage subsystems, a host controller that communicates with devices on a storage network.
See also target.

**Interop namespace**
A logical location containing a few key CIM classes and objects that have associations to the larger Implementation namespace. The Interop namespace generally has a well-known name, such as root/interop.

**intrinsic method**
A CIM method that can apply to any CIM class or object.
See also extrinsic method.

**IPMI (Intelligent Platform Management Interface)**
A specification for system management interfaces that operate independently of system software. IPMI controllers in a managed system can interact with CIM providers to make system management functions available to CIM clients.

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

**profile**
A standardized collection of CIM classes selected and organized to model a particular management area.

**provider**
A module associated with a CIMOM that provides information in response to CIM client requests. The CIMOM dispatches service requests to one or more providers and returns an aggregated response to a CIM client.

**SAN (storage area network)**
A large-capacity network of storage devices that can be shared by a number of servers.

**Scoping Instance**
In managed servers, an instance of CIM_ComputerSystem that serves the key role in associations with other objects describing a managed server.
SAS (Serial-attached SCSI)
A high-performance storage communication technology.

SATA (Serial Advanced Technology Attachment)
A standard, based on serial signaling technology, for connecting computers and hard drives. Also called “Serial ATA.”

SEL (system event log)
An ongoing record of operational events occurring on a managed system.

SLP (Service Location Protocol)
A standard for advertising services available to a network.

SMASH (Systems Management Architecture for Server Hardware)
A collection of CIM profiles and communication protocols selected for datacenter management.

SMBIOS (System Management BIOS)
A DMTF standard for application access to system BIOS information.
See also DMTF (Distributed Management Task Force), BIOS (basic input/output system).

SMI-S (Storage Management Initiative Specification)
A standard defined by SNIA for using a CIM client to manage a SAN.
See also CIM (Common Information Model), SNIA (Storage Networking Industry Association).

SMWG (Server Management Working Group)
A DMTF committee responsible for the SMASH standard.
See also DMTF (Distributed Management Task Force), SMASH (Systems Management Architecture for Server Hardware).

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

SOAP (Simple Object Access Protocol)
A lightweight XML-based communication protocol that provides the messaging framework for Web services. SOAP specifies a standard way to encode parameters and return values in XML, and standard ways to pass them using common network protocols like HTTP (Web) and SMTP (email).

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.

T target
In the context of storage subsystems, a device controller that responds to commands on a storage network.
See also initiator.

W WBEM (Web-Based Enterprise Management)
A standard for message-based system management over HTTP.

WS-Man (Web Services-Management)
A standard for system management using SOAP over HTTP.
See also SOAP (Simple Object Access Protocol).
Index

A
associations
CIM_AssociatedPowerManagementService 34
CIM_ComputerSystemPackage 18
CIM_ConcreteComponent 28
CIM_DeviceSAPImplementation 32, 33
CIM_ElementConformsToProfile 16
CIM_ElementSoftwareIdentity 20
CIM_IndicationFilter 36
CIM_IndicationSubscription 36
CIM_InstalledSoftwareIdentity 20, 21
CIM_LogicalIdentity 30
CIM_MediaPresent 34
CIM_MemberOfCollection 26, 32, 33
CIM_SystemDevice 22, 26, 28, 30, 31
authentication credentials 37

B
Base Server profile 12, 16
BIOS 20, 21

C
CIM object space 17
CIM server 37
CIM version 8
CIMOM 8, 12
CIM-XML 7
client libraries, CIM 38
connection object, client 13
connections
  CIM client to CIM server 37
  CIM server to indication consumer 38
  network 37
connections, troubleshooting 37
console 37
controllers, RAID 29, 31, 35
cores
  See processor cores
CPU 26

D
diagnosis 37
dlient applications, developing 11
DMTF 7, 8, 9

E
ESX Server 3.0 38
ESX Server 3.5 38
ESX Server 3i version 3.5 38
esxcfg-firewall utility 38
extents
  storage 32, 33, 34
firewall 38

H
hardware threads 26

I
Implementation namespace 11, 14, 15, 19
indication consumer 35, 38
indication producer 38
indications 35
initiators 31, 32, 33
instances
  CIM_AlertIndication 36
  CIM_ComputerSystem 22, 25, 28, 29, 30, 31, 34
  CIM_ConnectivityCollection 31, 32, 33
  CIM_DiskDrive 34
  CIM_Fan 26
  CIM_HardwareThread 26, 28
  CIM_LogicalPort 31
  CIM_NumericSensor 21, 22, 23
  CIM_PhysicalPackage 18, 19
  CIM_PortController 29, 30
  CIM_PowerManagementService 34
  CIM_Processor 28
  CIM_ProcessorCore 26, 28
  CIM_RedundancySet 24, 26
  CIM_RegisterProfile 11, 12, 15
  CIM_SCSIProtocolEndpoint 32, 33
  CIM_Sensor 21, 23, 24
  CIM_SoftwareIdentity 20, 21
  CIM_StorageExtent 34
  ProcessorCapabilities 26
  VMware_Controller 29, 30, 31
inventory 11, 12, 14, 16

Interop namespace 11, 12, 14, 16
M
maintenance mode 35
managed server 11, 15, 17, 18, 34
management agents 37
manufacturer 17, 19
methods, extrinsic
  RequestPowerStateChange() 35
model number 17, 19

N
namespaces 11
  Implementation 11, 14, 15, 19
  Interop 11, 12, 14, 16

O
OMC 7
online resources, CIM and SMASH 9

P
Perl Toolkit, VMware Infrastructure 38
platform product support 7
ports
  CIM server 37
  controller 29, 30
  network 12, 38
processor cores 26
profiles
  Base Server 8, 12, 16
  CPU 8
  Ethernet Port 8
  Fan 8
  IP Interface 8
  Power State Management 8
  Power Supply 8
  Profile Registration 8
  Record Log 8
  registered 11, 14
  Role Based Authorization 8
  Sensors 8
  Simple Identity Management 8
  SMASH 8, 11
  Software Inventory 8
  System Memory 8
  versions 8
properties
  BlockSize 34
  ConnectivityStatus 32
  CoreEnabledState 28
  CurrentClockSpeed 28
  CurrentState 23, 24
  DataRedundancy 34
  DeviceID 26
  ElementName 19, 23, 24, 26, 28, 30, 33, 34
  Family 28
  HealthState 30, 32, 35
  InstanceID 32
  MajorVersion 20
  Manufacturer 19, 21
  MinorVersion 20
  Model 19
  Name 30
  NumberOfBlocks 34
  OperationalStatus 34
  OtherControllerType 30
  PossibleStates 23
  RegisteredName 15
  RegisteredVersion 15
  SerialNumber 19
  VersionString 20, 21
protocol and version support 7

R
RAID 32
RAID controllers 29, 31, 35
rebooting 34, 37
redundancy, fans 24
registered profiles 11, 14
  listing 14
reset 34

S
sample clients 38
SAS 32
SATA 32
Scoping Instance 12, 15, 16, 21
SEL
  See System Event Log
sensors 21, 23
serial number 17, 19
server, managed 11, 15, 17, 18, 34
service console 38
Service URL 12
shell operations 37
SLP 7, 12, 37
SMASH profiles 8
SMASH version 8
SMI-S 9, 31
SMWG 8
SNIA 9, 31
state
  RAID connections 31
  RAID controller 29
  sensors 21, 23
  storage extents 32, 33, 34
Index

subnetwork 37
subscribing to indications 35
System Event Log 35

T
targets 31, 32, 33
technical support resources 6
threads
  See hardware threads
troubleshooting 37

U
URL
  CIM server 11, 12, 14, 37
  indication consumer 35, 36
  Service 12

V
version
  BIOS 20
  CIM 7
  profiles 8

W
WBEM 12
wbemcli utility 37
Web Services for Management Perl Library 38
WS-Management 7