

CIM SMASH API Programming Guide

ESX Server 3i version 3.5



You can find the most up-to-date technical documentation on our Web site at

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

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

© 2007 VMware, Inc. All rights reserved. Protected by one or more of U.S. Patent Nos. 6,397,242, 6,496,847, 6,704,925, 6,711,672, 6,725,289, 6,735,601, 6,785,886, 6,789,156, 6,795,966, 6,880,022, 6,944,699, 6,961,806, 6,961,941, 7,069,413, 7,082,598, 7,089,377, 7,111,086, 7,111,145, 7,117,481, 7,149,843, 7,155,558, 7,222,221, 7,260,815, 7,260,820, 7,269,683, 7,275,136, 7,277,998, 7,277,999, 7,278,030, 7,281,102, and 7,290,253; patents pending.

VMware, the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc.

3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book	7
Revision History	7
Intended Audience	7
Document Feedback	7
Technical Support and Education Resources	7
1 About the VMware CIM SMASH API	9
Platform Product Support	9
Protocol Support	9
CIM Version	9
SMASH Version	9
Profiles Supported	9
CIM and SMASH Resources Online	10
2 About the Profiles	11
Profile Registration Profile	11
Base Server Profile	11
Sensors Profile	11
Fan Profile	12
Power Supply Profile	12
CPU Profile	12
Power State Management Profile	12
System Memory Profile	12
Software Inventory Profile	12
Record Log Profile	13
Ethernet Port Profile	13
Role Based Authorization Profile	14
Simple Identity Management Profile	14
IP Interface Profile	15
Indications Profile	15
3 Use Cases	17
Locate a Server With SLP	17
Make a Connection to the CIMOM	17
List Registered Profiles	18
Identify the Server Scoping Instance	19
Query Manufacturer, Model, and Serial Number	20
Query Manufacturer, Model, and Serial Number Using Only the Implementation Namespace	21
Query the BIOS Version	22
Query State of All Sensors	23
Query State of All Sensors Using Only the Implementation Namespace	24
Reboot the Managed Server	25
Subscribe to Indications	26

Glossary 29

Index 31

Tables

Table 1. Revision History	7
Table 1-1. Profile Versions	10
Table 2-1. Record Log Profile Partial Implementation	13
Table 2-2. Ethernet Port Profile Partial Implementation	13
Table 2-3. Role Based Authorization Profile Partial Implementation	14
Table 2-4. Simple Identity Management Profile Partial Implementation	14
Table 2-5. IP Interface Profile Partial Implementation	15

About This Book

This book, the *VMware CIM SMASH Programming Guide*, contains information on VMware's 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 is revised with each release of the Guest SDK or when necessary. A revised version can contain minor or major changes. [Table 1](#) summarizes the significant changes in each version of this guide.

Table 1. Revision History

Revision	Description
20071210	ESX Server 3i version 3.5 release.

To view the most current version of this guide, go to <http://www.vmware.com/support/developer/>.

Intended Audience

The VMware CIM SMASH API Programming Guide is written for designers and implementers of client software that uses standard CIM interfaces to monitor the health of server hardware. This document is also useful to system administrators writing scripts to inventory and help manage their servers.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, send your feedback to:

docfeedback@vmware.com

Technical Support and Education Resources

The following sections describe the technical support resources available to you. You can access the most current versions of other VMware manuals by going to:

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

Online Support

You can submit questions or post comments to the VMware Management APIs (VI SDK, VI Perl, CIM SDK) forum, which is monitored by VMware technical support and product teams. You can access the forum at: <http://www.vmware.com/community/forum.jspa?forumID=393>.

Support Offerings

Find out how VMware support offerings can help meet your business needs. Go to <http://www.vmware.com/support/services>.

VMware Education Services

VMware courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. For more information about VMware Education Services, go to <http://mylearn1.vmware.com/mgreg/index.cfm>.

About the VMware CIM SMASH API

ESX Server 3i version 3.5 includes a CIM Object Manager (CIMOM) that implements a set of server discovery and monitoring features compatible with the SMASH standard. The VMware CIM SMASH 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.

Platform Product Support

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

Protocol Support

The VMware CIM SMASH API supports the following protocols:

- CIM-XML over HTTP
- WS-MAN
- 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 3i is compatible with version 2.14 (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 3i is compatible with version 1.0.0 of the SMASH standard.

Profiles Supported

The VMware CIM SMASH 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.

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 API for ESX Server 3i 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 ["About the Profiles"](#) on page 11.

Table 1-1. Profile Versions

Profile	Version
Base Server Profile	1.0.0a
CPU Profile	1.0.0
Ethernet Port Profile	N/A
Fan Profile	1.0.0
IP Interface Profile	N/A
Power State Management Profile	1.0.0
Power Supply Profile	1.0.0c
Profile Registration Profile	1.0.0
Record Log Profile	N/A
Role Based Authorization Profile	N/A
Sensors Profile	1.0.0
Simple Identity Management Profile	N/A
Software Inventory Profile	1.0.0
System Memory Profile	1.0.0f

CIM and SMASH Resources Online

The following resources related to the CIM and SMASH 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.dmtf.org/standards/published_documents/ (DMTF Publications)

About the Profiles

This chapter contains brief descriptions of the profiles implemented by the VMware CIM SMASH API. For some of the profiles, VMware implements all mandatory elements of the profile specification. In those cases, this document refers to the specification and does not list all the required elements. For other profiles, VMware implements only a subset of the mandatory elements of the profile specification. In such cases, this document lists the elements of the profile that are implemented.

Profile Registration Profile

This profile defines the objects and associations needed to publish to a CIM client the names and versions of profiles registered with the CIMOM. The primary CIM class describing registered profiles is `CIM_RegisteredProfile`. `CIM_RegisteredProfile` is available in the Interop namespace.

VMware provides implementations of all mandatory properties and methods defined by the related DMTF specification for each profile registered in an instance of `CIM_RegisteredProfile`.

This profile is featured in the use case “[List Registered Profiles](#)” on page 18. The Profile Registration profile is specified in http://www.dmtf.org/standards/published_documents/DSP1033.pdf.

Base Server Profile

This profile defines the objects and associations needed to describe a simple managed server, including hardware and software. Related profiles can be used to supplement this model for more complex server configurations.

This profile is featured in the use cases “[Identify the Server Scoping Instance](#)” on page 19 and “[Query Manufacturer, Model, and Serial Number](#)” on page 20.

The Base Server profile is specified in http://www.dmtf.org/standards/published_documents/DSP1004.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

Sensors Profile

This profile defines the objects and associations needed to model the sensors in a managed server. The Sensors profile supplements other profiles describing the managed server, adding the capability to monitor devices with attached sensors.

This profile is featured in the use cases “[Query State of All Sensors](#)” on page 23 and “[Query State of All Sensors Using Only the Implementation Namespace](#)” on page 24.

The Sensors profile is specified in http://www.dmtf.org/standards/published_documents/DSP1009.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

Fan Profile

This profile defines the objects and associations needed to model the cooling fans in a managed server. The Fan profile supplements other profiles describing the managed server, adding the capability to monitor and manage the physical characteristics of the fans and their associated sensors.

The Fan profile is specified in http://www.dmtf.org/standards/published_documents/DSP1013.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

Power Supply Profile

This profile defines the objects and associations needed to model the power supplies in a managed server. The Power Supply profile supplements other profiles describing the managed server, adding the capability to monitor and manage the physical characteristics and possible redundancy of power supplies.

The Power Supply profile is specified in http://www.dmtf.org/standards/published_documents/DSP1015.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

CPU Profile

This profile defines the objects and associations needed to model the CPUs in a managed server. The CPU profile supplements other profiles describing the managed server, adding the capability to manage the physical characteristics and the cache memory of the processors.

The CPU profile is specified in http://www.dmtf.org/standards/published_documents/DSP1022.pdf.

Power State Management Profile

This profile defines the objects and associations needed to model power management of a server. This profile supplements other profiles describing the managed server, adding the capability to model the power management service.

This profile is featured in the use case “[Reboot the Managed Server](#)” on page 25.

The Power State Management profile is specified in http://www.dmtf.org/standards/published_documents/DSP1027.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

System Memory Profile

This profile defines the objects and associations needed to model the memory of a server. This profile supplements other profiles describing the managed server, adding the capability to model the system memory configuration and its physical characteristics.

The System Memory profile is specified in http://www.dmtf.org/standards/published_documents/DSP1026.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

Software Inventory Profile

This profile defines the objects and associations needed to model the software installed on a managed server. The Software Inventory profile supplements other profiles describing the managed server, adding the capability to inventory applications, BIOS, firmware, and device drivers.

The Software Inventory profile is specified in http://www.dmtf.org/standards/published_documents/DSP1023.pdf. VMware implements all mandatory classes, properties, and methods of the profile.

Record Log Profile

VMware provides a partial, experimental implementation for this profile.

The Record Log profile is specified in http://www.dmtf.org/standards/published_documents/DSP1010.pdf. VMware implements the following classes, properties, and methods of the profile.

Table 2-1. Record Log Profile Partial Implementation

Class	Property or Method
CIM_LogRecord	RecordID
CIM_LogRecord	RecordData
CIM_LogRecord	RecordFormat
CIM_LogRecord	ElementName
CIM_LogRecord	LogCreationClassName
CIM_LogRecord	MessageTimeStamp
CIM_LogRecord	Caption
CIM_RecordLog	InstanceID
CIM_RecordLog	MaxNumberOfRecords
CIM_RecordLog	LogState
CIM_RecordLog	OverwritePolicy
CIM_RecordLog	RequestedState
CIM_RecordLog	EnabledState
CIM_RecordLog	OperationalStatus
CIM_RecordLog	HealthState
CIM_RecordLog	ElementName
CIM_RecordLog	Name
CIM_RecordLog	Caption
CIM_RecordLog	Version
CIM_RecordLog	AddTimeStamp
CIM_RecordLog	EraseTimeStamp
CIM_RecordLog	Flags
CIM_RecordLog	OverFlowFlag
CIM_RecordLog	ClearLog()

Ethernet Port Profile

VMware provides a partial, experimental implementation for this profile.

The Ethernet Port profile is specified in http://www.dmtf.org/standards/published_documents/DSP1014.pdf. VMware implements the following classes, properties, and methods of the profile.

Table 2-2. Ethernet Port Profile Partial Implementation

Class	Property or Method
CIM_EthernetPort	PortType
CIM_EthernetPort	NetworkAddresses
CIM_EthernetPort	Capabilities
CIM_EthernetPort	EnabledCapabilities

Table 2-2. Ethernet Port Profile Partial Implementation (Continued)

Class	Property or Method
CIM_EthernetPort	LinkTechnology
CIM_EthernetPort	PermanentAddress

Role Based Authorization Profile

VMware provides a partial, experimental implementation for this profile.

The Role Based Authorization profile is specified in http://www.dmtf.org/standards/published_documents/DSP1039.pdf. VMware implements the following classes, properties, and methods of the profile.

Table 2-3. Role Based Authorization Profile Partial Implementation

Class	Property or Method
CIM_Role	CreationClassName
CIM_Role	Name
CIM_Role	RoleCharacteristics
CIM_Role	CommonName
CIM_Role	ElementName
CIM_MemberOfCollection	GroupComponent
CIM_MemberOfCollection	PartComponent
CIM_MemberOfCollection	GroupComponent
CIM_MemberOfCollection	PartComponent
CIM_Privilege	InstanceID
CIM_Privilege	RepresentsAuthorizationRights
CIM_Privilege	PrivilegeGranted

Simple Identity Management Profile

VMware provides a partial, experimental implementation for this profile.

The Simple Identity Management profile is specified in http://www.dmtf.org/standards/published_documents/DSP1034.pdf. VMware implements the following classes, properties, and methods of the profile.

Table 2-4. Simple Identity Management Profile Partial Implementation

Class	Property or Method
CIM_Identity	InstanceID
CIM_Identity	ElementName
CIM_Account	SystemCreationClassName
CIM_Account	SystemName
CIM_Account	CreationClassName
CIM_Account	Name
CIM_Account	UserID
CIM_Account	UserPassword
CIM_Account	OrganizationName
CIM_Account	ElementName

Table 2-4. Simple Identity Management Profile Partial Implementation (Continued)

Class	Property or Method
CIM_AssignedIdentity	IdentityInfo
CIM_AssignedIdentity	ManagedElement

IP Interface Profile

VMware provides a partial, experimental implementation for this profile.

The IP Interface profile is specified in http://www.dmtf.org/standards/published_documents/DSP1036.pdf. VMware implements the following classes, properties, and methods of the profile.

Table 2-5. IP Interface Profile Partial Implementation

Class	Property or Method
CIM_IPProtocolEndpoint	IPv4Address
CIM_IPProtocolEndpoint	IPv4DefaultGateway
CIM_IPProtocolEndpoint	MACAddress
CIM_IPProtocolEndpoint	Name
CIM_IPProtocolEndpoint	SystemName
CIM_IPProtocolEndpoint	CreationClassName
CIM_IPProtocolEndpoint	SystemCreationClassName
CIM_HostedAccessPoint	Antecedent
CIM_HostedAccessPoint	Dependent

Indications Profile

VMware provides partial support for the CIM Indications feature. Indications are supported for the following:

- LSI Logic RAID Controller
- IPMI SEL

The Indications profile is specified in http://www.dmtf.org/standards/published_documents/DSP1054.pdf.

VMware does not instantiate any static filters. The VMware implementation does not include the FilterCollection class, so you can only subscribe to individual user-created filters.

For instructions on how to subscribe to indications, see “[Subscribe to Indications](#)” on page 26.

Use Cases

These use cases are for CIM clients managing a server that runs VMware ESX Server 3i. Each use case describes a goal to accomplish, steps to accomplish the goal, and a few lines of sample code to demonstrate the steps used in the client.

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 Interop namespace, you can 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 you need 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 17.

For more information about SLP, see the following links.

<http://tools.ietf.org/html/rfc2608>

<http://tools.ietf.org/html/rfc3059>

Make a Connection to the CIMOM

- 1 Collect the connection parameters from the environment.

```
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.

```

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 18.

List Registered Profiles

A CIM client should generally 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 sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17.

- 1 Make a connection to the CIMOM, 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.

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.

function print_profile( instance )
    print '\n' + '[' + 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 way 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 way to locate the objects requires less knowledge of the implementation.

This section shows the first step of locating CIM objects from the Scoping Instance.

The sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17.

- 1 Make a connection to the CIMOM, 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
    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)

```

- Enumerate instances of CIM_RegisteredProfile.

```
use registered_profiles renamed prof

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

- Select the instance corresponding to the Base Server profile.

```
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 Profile'
            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
```

- Traverse the CIM_ElementConformsToProfile association to reach the Scoping Instance.

```
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 '\n' + '[' + 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 ) )
```

Query 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.

The sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17 and on the sample code in [“Identify the Server Scoping Instance”](#) on page 19.

- 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
    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 Scoping Instance.'
    sys.exit(-1)

```

3 Traverse the CIM_ComputerSystemPackage association to reach the CIM_PhysicalPackage instance corresponding to the managed server.

```

instance_names = connection.AssociatorNames( scoping_instance_name, \
        AssocClass = 'CIM_ComputerSystemPackage', \
        ResultRole = 'Antecedent' )
if len( instance_names ) > 1
    print 'Error: %d Physical Packages found for Scoping Instance.' \
        % len ( instance_names )
    sys.exit(-1)

```

4 Print the Manufacturer, Model, and SerialNumber properties.

```

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

```

Sample output looks like this:

```

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

```

Query 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 sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17 and on the sample code in [“Identify the Server Scoping Instance”](#) on page 19.

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

```

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.

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

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

```
function print_info( instance )
    print '\n' + object_type + ' [' + instance.classname + '] -&gt;'
    for prop in [ 'Manufacturer', 'Model', 'SerialNumber' ]
        print ' %30s = %s' % ( prop, instance[prop] )
```

```
for instance in list
    if instance[ 'ElementName' ] == 'Chassis'
        print_info( instance )
        sys.exit(0)
print 'Unable to find Physical Package instance for Chassis.'
```

Sample output looks like this:

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

Query 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.

VMware's implementation of CIM_InstalledSoftware makes the version available in the VersionString property rather than the MajorVersion and MinorVersion properties.

The sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17 and on the sample code in [“Identify the Server Scoping Instance”](#) on page 19.

- 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 = ci.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_InstalledSoftware` 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;';
    for prop in [ 'Manufacturer', 'VersionString' ]
        print ' %30s = %s' % ( prop, instance[prop] )

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

Query 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.

At this time, the VMware implementation does not include `CIM_DiscreteSensor`. Only instances of `CIM_NumericSensor` are available.

This section shows how to get the sensor states from the Interop namespace, traversing associations from the managed server Scoping Instance. For information on getting sensor states using only the Implementation namespace, see [“Query State of All Sensors Using Only the Implementation Namespace”](#) on page 24.

The sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17 and on the sample code in [“Identify the Server Scoping Instance”](#) on page 19.

- 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
    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_NumericSensor` instances on the managed server.

```
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)
```

- 4 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.

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

for instance in instances
    print_info( instance )
```

Sample output looks like this:

```
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
```

Query 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.

At this time, the VMware implementation does not include `CIM_DiscreteSensor`. Only instances of `CIM_NumericSensor` are available.

This topic shows how to get the sensor states from the Implementation namespace, assuming you already know its name. For information on getting sensor state using the standard Interop namespace, see [“Query State of All Sensors”](#) on page 23.

The sample code in this topic depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17.

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

```
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`.

```
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`.

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

```

for instance in instances
    print_info( instance )

```

- 4 Sample output looks like this:

```

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

```

Reboot the Managed Server

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

The sample code in this section depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17.

- 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
    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_AssociatedPowerManagementService association to reach the CIM_PowerManagementService instance on the managed server.

```

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).

```

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 may be left in maintenance mode after power cycling.

Subscribe to Indications

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:

- DMTF's CIM Indications White Paper at <http://www.dmtf.org/standards/documents/CIM/DSP0107.pdf>
- DMTF's Indications Profile specification at http://www.dmtf.org/standards/published_documents/DSP1054.pdf
- 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 sample code in this section depends on the sample code in [“Make a Connection to the CIMOM”](#) on page 17.

- 1 Make a connection to the CIMOM, 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 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' : handlerName )

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

subInst = wbemlib.CIMInstance( 'CIM_IndicationSubscription', \
                               path = subName )
subInst['Filter'] = cfilterName
subInst['Handler'] = handlerName

rsubName = client.CreateInstance( subInst )

```


Glossary

C **CIM (Common Information Model)**

A collection of standards created by the DMTF to provide a shared model for managing systems.

CIMOM (CIM Object Manager)

A service that provides standard CIM management functions over a WBEM connection to a CIM client.

D **DMTF (Distributed Management Task Force)**

An industry-wide consortium of hardware and software vendors with a mission to define interoperability standards.

F **FRU (Field Replaceable Unit)**

A part that can be removed and replaced at the installed site.

I **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.

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`.

P **profile**

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

S **Scoping Instance**

In managed servers, an instance of `CIM_ComputerSystem` that serves the key role in associations with other objects describing a managed server.

SMASH (Systems Management Architecture for Server Hardware)

A collection of CIM profiles and communication protocols selected for datacenter management.

W **WBEM (Web-Based Enterprise Management)**

A standard protocol for passing CIM-XML messages over HTTP.

Index

A

associations

- CIM_ComputerSystemPackage **21**
- CIM_ElementConformsToProfile **20**
- CIM_InstalledSoftware **23**
- CIM_SystemDevice **23**

associations

- CIM_AssociatedPowerManagementService **25**

B

Base Server profile **20**

BIOS **12, 22, 23**

C

CIM version **9**

CIMOM **10, 17**

CIM-XML **9**

connection object, client **18**

CPU **12**

D

DMTF **9, 10**

F

fans **12**

filters, indication **15**

firmware **12**

I

Implementation namespace **18, 19, 21, 24**

indication consumer **26**

indications **26**

Instances

- CIM_RegisteredProfile **19**

instances

- CIM_AlertIndication **27**
- CIM_DiscreteSensor **23, 24**
- CIM_NumericSensor **23, 24**
- CIM_PhysicalPackage **21, 22**
- CIM_PowerManagementService **25**
- CIM_Sensor **24**
- CIM_SoftwareIdentity **23**

Interop namespace **11, 17, 18, 19, 20, 22, 23, 24, 25**

inventory **20**

IPMI SEL **15**

L

LSI Logic **15**

M

maintenance mode **25**

managed server **11, 17, 19, 21, 22, 23, 25**

memory **12**

methods

- RequestPowerStateChange() **25**

N

namespace

- Implementation **18, 19, 21, 24**

- Interop **11, 18, 19, 20, 22, 23, 24, 25**

namespace, Interop **17**

O

OMC **9**

P

power management service **12**

power supplies **12**

profiles

- Base Server **10, 11, 20**

- CPU **10, 12**

- Ethernet Port **10, 13**

- experimental **13, 14, 15**

- Fan **10, 12**

- Indications **15**

- IP Interface **10, 15**

- mandatory elements **11**

- Power State Management **10, 12**

- Power Supply **10, 12**

- Profile Registration **10, 11**

- Record Log **10, 13**

- registered **18**

- Role Based Authorization **10**

- Sensors **10, 11**

- Simple Identity Management **10, 14**

- SMASH **9, 11**

- Software Inventory **10, 12**

- System Memory **10, 12**

profiles, SMASH **10**

properties

- CurrentState **24**

- ElementName **22, 24**

MajorVersion **22**
Manufacturer **21, 22, 23**
Model **21, 22**
PossibleStates **24**
RegisteredName **19**
RegisteredVersion **19**
SerialNumber **21, 22**
VersionString **22, 23**

properties

MinorVersion **22**

R

rebooting **25**
registered profiles **18**
reset **25**

S

Scoping Instance **19, 20, 21, 22, 23, 25**
sensors **11, 23, 24**
server **11**
server, managed **17, 19, 21, 22, 23, 25**
SLP **9, 17**
SMASH profiles **9, 10, 11**
SMWG **9**

T

technical support resources **7**

U

use cases **17**

V

version, CIM **9**

W

WBEM **17**
WS-MAN **9**