

vSphere Web Services SDK Programming Guide

Modified 15 MAY 2023

VMware vSphere 8.0

VMware ESXi 8.0

vCenter Server 8.0

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

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

Copyright © 2011-2023 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

About This Book 14

1 About the vSphere Web Services SDK 17

- vSphere Web Services SDK 17
- SDK Developer Setup for the Web Services SDK 18
- SDK Samples for the Web Services SDK 18
- UML Diagrams Used in the Web Services SDK Programming Guide 18

2 vSphere Web Services API Programming Model 20

- vSphere Client-Server Architecture 20
- Web Services API as a Web Service 21
 - WSDL Files and the Client-Side Proxy Interface 21
 - Network Access to the vSphere Web Service 22
 - Language-Specific Classes and Methods 23
 - Mapping XML Data Types to Java or JSON Data Types 23
- Access to Managed Objects 24
- Access to vSphere Server Data 25
 - Obtaining Information from a Server 25
 - Working with Data Structures 26
 - Accessing Property Values 26
 - Nested Properties and Property Paths in Composite Data Structures 27
 - xsd:anyType Arrays 27
 - Indexed Array and Key-Based Array Properties 30
 - Unset Optional Properties 30
 - Escape Character in Name and Path Properties 31

3 Client Applications for the Web Services API 32

- vCenter Server Connections 32
- Establishing a Single Sign-On Session with a vCenter Server 33
- LoginByToken to vCenter Server By Using Java 33
 - vCenter Server Single Sign-On Session Using Java 34
 - HTTP and SOAP Header Handlers in Java 34
 - Creating the HTTP Connection in Java 36
 - Using LoginByToken in Java 37
- Establishing a Session with Username and Password Credentials 40
- Overview of a Java Sample Application for the Web Services SDK 40
 - Build a Simple vSphere Client Application for the Web Services SDK 40
 - Java Client Example for the Web Services SDK 41

Web Server Session Token	44
Accessing the vSphere Web Services HTTP Endpoint with JAX-WS	45
Accessing the vSphere Server from a Web Services Client	47
Closing the Connection from a Web Services Client	48
Using the Java Samples as Reference	48
Multiple Versions of the vSphere API	49
Java Sample Applications in the Web Services SDK	50
Java Samples in the Web Services SDK	50
4 Client Applications Using the JSON Protocol	51
Client Requirements for the JSON Protocol	51
Outlining a Client for the JSON Protocol	52
Building JSON Request URLs	53
Building JSON Request Bodies	54
Retrieving the Service Instance with the JSON Protocol	55
Authenticating a JSON Client with the Session Manager	56
Basic Authentication for the JSON Protocol	57
SAML Token Authentication for a JSON Client	58
Mapping Methods for the JSON Protocol	59
Accessing Managed Object Properties with the JSON Protocol	59
Using Polymorphic Types with the JSON Protocol	64
Python Client Example for the JSON Protocol	65
5 Datacenter Inventory	69
Inventory Overview for the Web Services SDK	69
Inventory Hierarchies and ServiceInstance	69
Folders in the Hierarchy	70
ESXi Inventory Hierarchy	71
Accessing Inventory Objects	71
Creating Inventory Objects	72
Privileges Required for Inventory Management	73
Privileges	73
Permissions	73
Managed and Standalone ESXi Hosts	74
6 Property Collector	76
Introduction to the PropertyCollector	76
Data Retrieval	77
Inventory Traversal and Property Selection	77
Gathering Data with a ViewManager Object	78
vSphere Data Objects for Property Collection	78

vSphere Methods for Property Collection	79
Using the PropertyCollector with RetrievePropertiesEx	80
Retrieve a Property from a Specified Managed Object	80
Retrieve Properties from a Set of Managed Objects with a View	83
Simple Property Collector Program in Java for Web Services SDK	86
Inventory Traversal	91
TraversalSpec Traversal	91
Traverse the Inventory By Using the Property Collector	93
Inventory Traversal Example in Java	94
SelectionSpec Traversal	99
Client Data Synchronization (WaitForUpdatesEx)	105
Property Filters	106
WaitForUpdatesEx	106
Server Data Transmission	108
PropertyCollector Performance	109
SearchIndex	109

7 Authentication and Authorization 110

Objects for Authentication and Authorization Management	110
Authentication and Authorization for ESXi and vCenter Server	111
ESXi User Model	112
vCenter Server User Model	113
vSphere Security Model	113
Setting Up Users, Groups, and Permissions	115
Obtaining User and Group Information from UserDirectory	115
RetrieveUserGroups Method	116
Managing ESXi Users with HostLocalAccountManager	116
Methods Available for Local Account Management	117
Create a Local User Account on an ESXi System	117
Managing Roles and Permissions with AuthorizationManager	117
Using Roles to Consolidate Sets of Privileges	119
Modifying Sample Roles to Create New Roles	120
Granting Privileges Through Permissions	120
Authenticating Users Through SessionManager	123
Using VMware Single Sign On for vCenter Server Sessions	124
Using the Credential Store for Automated Login	124
Credential Store Libraries	124
Credential Store Methods	125
Credential Store Backing File	125
Credential Store Samples	126
Specifying Roles and Users with the Credential Store	127

Managing Licenses with LicenseManager 127

8 Hosts 129

- Host Management Objects 129
- Retrieving Host Information 130
- Configuring and Reconfiguring Hosts 131
- Managing the Host Lifecycle 131
 - Reboot and Shutdown 131
 - Using Standby Mode 131
 - Disconnecting and Reconnecting Hosts 132
- Querying and Changing the Host Time 133
- Querying Virtual Machine Memory Overhead 133

9 Storage 134

- Storage Management Objects 134
- Introduction to Storage 135
 - How Virtual Machines Access Storage 136
 - Datastores 137
- Choosing the Storage API to Use 138
 - Managed Objects for Working with Storage 139
- Configuring Disk Partitions 140
- Multipath Management 140
- Configuring iSCSI Storage 141
 - Configure the VMkernel To Support Software iSCSI 142
 - Configure iSCSI Initiators 142
- Creating and Managing Datastores 144
 - Accessing Datastores 145
 - Creating and Modifying a VMFS Datastore 145
 - Removing and Updating Datastores 147
 - Managing VMFS Datastores with HostStorageSystem 147
- Managing VMFS Volume Copies (Resignaturing) 148
 - Resignaturing Volumes with `ResignatureUnresolvedVmfsVolume_Task` 149
- Managing Diagnostic Partitions 149
 - Retrieving Diagnostic Partition Information 150
 - Create a Diagnostic Partition 150
- Sample Code Reference 151

10 vSphere Networks 152

- Virtual Switches 152
 - Port Groups 152
 - Virtual Machine Network Interfaces 153

VMkernel Network Interfaces	153
Physical Network Adapter (pnict)	154
Using a Distributed Virtual Switch	154
Distributed Virtual Switch Configuration	155
Backup, Rollback, and Query Operations	156
VMware Standard Virtual Switch	157
Configuring a Standard Virtual Switch	158
vNetwork Standard Switch Environment	158
Setting Up Networking with vSS	159
Defining the Host Network Policies	162
NIC Teaming	162
Setting Up IPv6 Networking	163
Adding Networking Services	164
Sample Code Reference	165

11 Virtual Machine Configuration 166

VirtualMachine Management Objects and Methods	166
Creating Virtual Machines and Virtual Machine Templates	167
Creating a Virtual Machine Using VirtualMachineConfigSpec	167
Creating Virtual Machine Templates	169
Cloning a Virtual Machine	169
Converting a Template to a Virtual Machine	170
Accessing Information About a Virtual Machine	170
Configuring a Virtual Machine	171
Name and Location	173
Hardware Version	173
Boot Options	174
Operating System	174
CPU and Memory Information	174
Networks	177
Fibre Channel NPIV Settings	178
File Locations	179
Adding Devices to Virtual Machines	179
Performing Virtual Machine Power Operations	181
Registering and Unregistering Virtual Machines	182
Customizing the Guest Operating System	183
Installing VMware Tools	184
Upgrading a Virtual Machine	184

12 Virtual Machine Management 185

Virtual Machine Migration	185
---------------------------	-----

Cold Migration	186
Migration with vMotion	186
Using Storage vMotion	186
Snapshots	187
Creating a Snapshot	187
Reverting to a Snapshot	188
Deleting a Snapshot	188
Linked Virtual Machines	188
Linked Virtual Machines and Disk Backings	188
Creating a Linked Virtual Machine	189
Instant Clone Virtual Machines	191
Removing Snapshots and Deleting Linked Virtual Machines	197
Relocating a Virtual Machine in a Linked Virtual Machine Group	198
Promoting a Virtual Machine's Disk	198
Performing Advanced Manipulation of Delta Disks	199

13 Virtual Machine Guest Operations 201

Authenticating with the Guest Operating System	201
Running Guest OS Operations	201
Guest Operating System Customization	205
Guest Customization for Stopped Virtual Machines	207
Guest Customization Using cloud-init	207
Prepare a Virtual Machine Template for Raw Data cloud-init Customization	208
Prepare a cloud-init Metadata File	209
Prepare a cloud-init Userdata File	210
Initiate Raw Data cloud-init Customization with the vSphere Web Services API	211
Guest Network Customization for Instant Clone Virtual Machines	212
Installing the Guest Customization Engine	213
Disconnecting Virtual NICs	214
Customizing Guest Network Settings for Running Virtual Machines	214
Reconnecting Virtual NICs in a Running Virtual Machine	216
Restarting the Guest Network After Customization	216
Recovering from Guest Network Customization Errors	216
Application-Dependent Customization	217
Resetting the Network Stack in a Running Virtual Machine	217
Resetting the Network Stack in a Linux Virtual Machine	217
Resetting the Network Stack in a Windows Virtual Machine	218

14 Virtual Machine Security 220

How Virtual Machine Encryption Protects a Datacenter	220
What Keys are Used	221

What Is Encrypted	221
What Is Not Encrypted	222
Who Can Perform Cryptographic Operations	222
How Can I Perform Cryptographic Operations	222
vSphere Virtual Machine Encryption Components	223
Key Management Server	223
vCenter Server	224
ESXi Hosts	224
Encryption Process Flow	224
Prerequisites and Required Privileges for Encryption Tasks	225
Cryptography Privileges and Roles	225
Host Encryption Mode	226
Encrypted vSphere vMotion	226
API Methods for vSphere Virtual Machine Encryption	226
vSphere API Methods for KMS Management	227
API Methods to Prepare an ESXi Host	228
vSphere API Methods for Cryptographic Operations	228
SPBM API Methods for Encryption	230
Workflows for vSphere Virtual Machine Encryption	230
Set Up the Key Management Server Cluster	230
Create an Encryption Storage Policy	233
Create an Encrypted Virtual Machine	236
Clone an Encrypted Virtual Machine	237
Encrypt an Existing Virtual Machine or Disk	237
Decrypt an Encrypted Virtual Machine or Disk	238
Encrypt with Different Keys for VM and Disk	239
Reencrypt (Rekey) Encrypted Virtual Machines	240
Query Crypto Key In-Use Status	241
Encrypted vSphere vMotion	242
Virtual Disk Manager	242
Best Practices for Virtual Machine Encryption	242
Configuring Advanced Options	244
Physical Platform Resources	244

15 Virtual Applications 248

About Virtual Applications	248
Management Overview	249
Direct and Linked Children	249
OVF Packages	250
Creating a VirtualApp	250
Managing VirtualApp Children	251

Exporting a Virtual Application	252
VirtualApp and OvfManager Methods	254
VirtualApp Data Structures	254
OvfManager Data Structures	255
Example of Generating an OVF Package	256
Importing an OVF Package	258
Virtual Application Life Cycle	259
Powering a Virtual Application On or Off	259
Unregistering a Virtual Application	260
Suspending a Virtual Application	260
Destroying a Virtual Application	260
16 Resource Management	261
Resource Management Objects	261
Introduction to Resource Management	262
Resource Allocation	262
Resource Pool Hierarchies	263
Cluster Overview	263
Creating and Configuring Resource Pools	263
Configuring Reservation and Limit for Resource Pools	264
Configuring Priority Shares for Resource Pools	268
Deleting Child Resource Pools	279
Moving Resource Pools or Virtual Machines Into a Resource Pool	279
Introduction to vSphere Clusters	280
VMware DRS	280
VMware HA	280
VMware HCI	281
Creating and Configuring Clusters	281
Creating a Cluster	282
Adding a Host to a Cluster	283
Reconfiguring a Cluster	283
Managing DRS Clusters	284
Managing HA Clusters	284
Primary and Secondary Hosts	285
Failure Detection and Host Network Isolation	285
Using VMware HA and DRS Together	286
17 Tasks and Scheduled Tasks	287
Creating Tasks	287
Session Persistence	288
Cancelling a Task	288

Using TaskInfo to Determine Task Status	288
Monitoring TaskInfo Properties	289
Accessing and Manipulating Multiple Tasks	291
Task Monitoring Example Using the Listview Object	291
Gathering Data with a TaskManager Interface	304
Understanding the ScheduledTaskManager Interface	306
Scheduling Tasks	307
Cancelling a Scheduled Task	310
Using a TaskHistoryCollector	311
Managing the HistoryCollector	312
Sample Code Reference	312

18 Events and Alarms 313

Event and Alarm Management Objects	313
Understanding Events	314
Managing Events with EventManager	314
Event Data Objects	315
Formatting Event Message Content	316
Creating Custom Events	316
Using an EventHistoryCollector	318
Creating an EventHistoryCollector Filter	319
Managing the HistoryCollector	319
Using Alarms	319
Obtaining a List of Alarms	320
Creating an Alarm	320
Defining Alarms Using the AlarmSpec Data Object	321
Specifying Alarm Trigger Conditions with AlarmExpression	322
Specifying Alarm Actions	323
Deleting or Disabling an Alarm	324
Sample Code Reference	325

19 vSphere Performance 326

vSphere Performance Data Collection	326
PerformanceManager Objects and Methods	329
Retrieving vSphere Performance Data	330
Performance Counter Example (QueryPerf)	331
Large-Scale Performance Data Retrieval	343
Using the QueryPerf Method as a Raw Data Feed	343
Comparison of Query Methods	344
Retrieving Summary Performance Data	345
Performance Counter Metadata	345

PerfCounterInfo	345
Performance Intervals	346
ESXi Server Performance Intervals	347
vCenter Server Performance Intervals	348
vSphere Performance and Data Storage	348
Modifying Historical Intervals	348
Modifying Performance Counter Collection Levels	349
Sample Code Reference	350

20 Diagnostics and Troubleshooting 352

Troubleshooting Best Practices	352
Overview of Configuration Files and Log Files	353
ESXi Log File	354
Virtual Machine Log Files	354
vCenter Server Log Files	355
Modifying the Log Level to Obtain Detailed Information	356
Setting the Log Level on ESXi Systems	356
Generating Logs	357
Setting the Log Level on vCenter Server Systems	357
Using DiagnosticManager	358
Using the MOB to Explore the DiagnosticManager	359
Generating Diagnostic Bundles	360
Export Diagnostic Data By Using the vSphere Client	360

21 Managed Object Browser 362

Using the MOB to Explore the Object Model	362
Accessing the MOB	362
Using the MOB to Navigate the vSphere Object Model	363
Using the MOB to Invoke Methods	364
Passing Primitive Datatypes to Method	364
Passing Arrays of Primitives to Methods	364
Passing Complex Structures to Methods	365
Using the MOB Along With the API Reference	369

22 HTTP Access to vSphere Server Files 370

Introduction to HTTP Access	370
URL Syntax for HTTP Access	371
Datastore Access (/folder)	371
Host File Access (/host)	372
Update Package Access (/tmp)	373
Privilege Requirements for HTTP Access	373

23 Sample Program Overview 375

Java Sample Programs (JAXWS Bindings) 375

About This Book

The *vSphere Web Services SDK Programming Guide* provides information about developing applications using the VMware® vSphere Web Services SDK.

VMware provides different APIs and SDKs for various applications and goals. The vSphere Web Services SDK targets developers who create client applications for managing VMware® vSphere components available on VMware ESXi and VMware vCenter Server systems.

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

Revision History

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

Table 1-1. Revision History

Revision Date	Description
15 MAY 2023	Minor updates: <ul style="list-style-type: none">■ Corrected Retrieving the Service Instance with the JSON Protocol which previously referred to PropertyCollector as a singleton, which is not strictly correct.■ Added clarifications regarding PropertyCollector updates, in Client Data Synchronization (WaitForUpdatesEx).
14 APR 2023	vSphere 8.0 Update 1 <ul style="list-style-type: none">■ Added Chapter 4 Client Applications Using the JSON Protocol.
05 DEC 2022	Minor cleanup.
06 OCT 2022	vSphere 8.0 <ul style="list-style-type: none">■ Removed C# and .NET material after deprecation.■ Removed more system configuration parameters that have been moved to ConfigStore. These settings can no longer be edited directly with a web browser.■ Updates for accuracy including Security chapter. Alt text for graphics.
05 OCT 2021	vSphere 7.0 Update 3 <ul style="list-style-type: none">■ Added raw cloud-init configuration to Guest Operations chapter.■ Added PTP time service capability to Hosts chapter.

Table 1-1. Revision History (continued)

Revision Date	Description
19 MAY 2021	<ul style="list-style-type: none"> ■ Added information about configuring advanced host options. ■ Added information about configuring direct access to physical devices on the host. ■ A number of system configuration parameters moved to ConfigStore. These settings can no longer be edited directly with a web browser.
12 APR 2021	VMware has rebranded the My VMware portal as VMware Customer Connect. We have updated this document accordingly.
9 MAR 2021	vSphere 7.0 Update 2 <ul style="list-style-type: none"> ■ Added a simpler use case in the Property Collector chapter. Also made minor corrections and clarifications. ■ Rerorganized part of the Tasks chapter to break up the very long ListView example program into sections. Also made minor corrections and clarifications.
06 OCT 2020	vSphere 7.0 Update 1 <ul style="list-style-type: none"> ■ In Inventory Traversal and Property Selection, add best practice to destroy PropertyCollector views after use. ■ In Task Monitoring Example Using the ListView Object, cross-reference information about chunking of results from PropertyCollector. ■ At VMware, we value inclusion. To foster this principle within our customer, partner, and internal community, we are replacing some of the terminology in our content. We have updated this guide to remove instances of non-inclusive language.
02APR2020	vSphere 7.0 Added scalable shares feature. Reworked and expanded fixed shares and other resource anagement material. Added Virtual Machine Guest Operations chapter, including guest customization for instant clone virtual machines.
20AUG2019	vSphere 6.7 Update 3 Bug fixes. Added section on crypto key query for VM encryption.
16OCT2018	vSphere 6.7 Update 1 Added section on Hyperconverged Infrastructure (HCI) clusters.
17APR2018	vSphere 6.7 - GA Added Instant Clone material to VM Management chapter. Minor updates elsewhere.
12APR2017	Removed appendix with list of permissions. Was not maintained.
15NOV2016	vSphere 6.5 - GA Added chapter about vSphere virtual machine encryption. Removed description of deprecated CIM Storage Management API. Updated information about session cookie management.
04SEP2015	Updated information about migrating VMs with vMotion across data centers.
12MAR2015	vSphere 6.0 - Rewrote “Exporting a Virtual Application” section in Virtual Applications chapter.
19SEP2013	vSphere 5.5 – Added a C# example of using LoginByToken; clarified limitation for HA clusters.

Table 1-1. Revision History (continued)

Revision Date	Description
10SEP2012	vSphere 5.1 – Added information about using the SessionManager.LoginByToken method; added information about distributed virtual switches.
24AUG2011	vSphere 5.0 - Revised performance manager chapter. Added information about: unset properties, using vCenter to access host data, and using the QueryConfigOption to add devices; emphasized ListView instead of TaskManager; clarified limits and limitations of Linked Virtual Machines; updated samples in chapters 3,5,14, and 16; replaced information about Axis bindings with JAX-WS; and updated paths to samples supplied with SDK.
13JUL2010	Restructured manual and added chapters about host, storage, and networking. Revised property collector chapter and added appendix about HTTP access.
07MAY2009	vSphere Web Services SDK 4.0 Programming Guide.

Intended Audience

This book is intended for anyone who needs to develop applications using the vSphere Web Services SDK. Developers typically create client applications using Java and targeting VMware vSphere. An understanding of Web Services technology and some programming background is required.

Document Feedback

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

About the vSphere Web Services SDK

1

VMware vSphere supports robust, fault-tolerant virtualized applications, networking, and storage. vSphere offers many optional components and modules such as VMware High Availability and VMware VMotion. The VMware vSphere Web Services SDK gives Web services developers programmatic access to vSphere components.

This chapter includes the following topics:

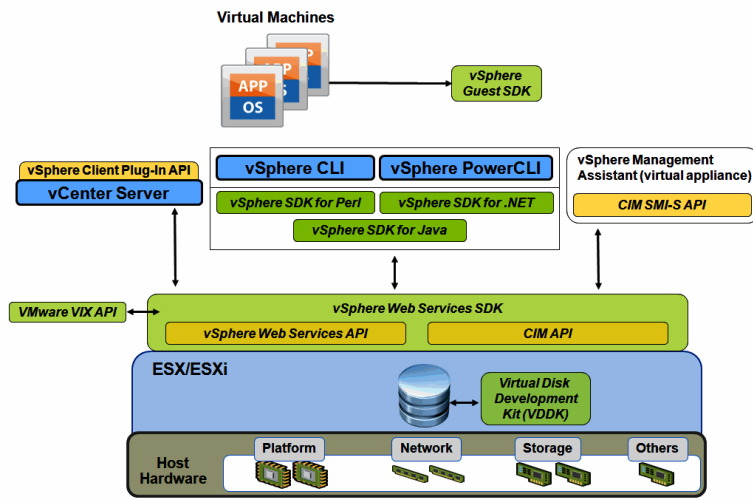
- [vSphere Web Services SDK](#)
- [SDK Developer Setup for the Web Services SDK](#)
- [SDK Samples for the Web Services SDK](#)
- [UML Diagrams Used in the Web Services SDK Programming Guide](#)

vSphere Web Services SDK

The vSphere Web Services SDK is the most comprehensive of the available management APIs. The SDK works against both ESXi and vCenter Server systems.

As a Web Services SDK, the SDK is language neutral. The SDK includes stubs and examples for Java and a comprehensive documentation set including an *API Reference* generated from the source.

Figure 1-1. vSphere APIs and CLIs



SDK Developer Setup for the Web Services SDK

Before you can start developing applications with the vSphere Web Services SDK, you must download the software and set up your system. The *vSphere Web Services SDK Developers Setup Guide* has complete instructions for Java client development and discusses a simplified security configuration for development environments.

SDK Samples for the Web Services SDK

The SDK includes a set of Java samples that illustrate much of the SDK features:

The samples include a set of utility applications that are used by the sample code.

The code fragments in this guide are in part based on the Java sample applications, but present code that does not require utility applications to run.

See [Chapter 23 Sample Program Overview](#) for lists of samples and a brief explanation of what each sample does.

UML Diagrams Used in the Web Services SDK Programming Guide

This guide uses UML (unified modeling language) diagrams to illustrate the API objects and their relationships.

The guide includes class diagrams and instance diagrams. [Figure 1-2. Legend for UML Class Diagrams](#) shows the UML notation used for managed objects and data objects. The diagrams use a tilde (~) if an object has no properties or methods. Ellipses (...) means some properties or methods are omitted.

Figure 1-2. Legend for UML Class Diagrams

Managed Object Type
property name : datatype
...
method name (parameter1, parameter2, ...) : return type
...

Data Object Type
property name : datatype
...
~

vSphere Web Services API Programming Model

2

The vSphere Web Services API is implemented as a language-neutral Web service. The API is based on a remote procedure call mechanism that client applications use to access services and components on ESXi and vCenter Server systems.

This chapter includes the following topics:

- [vSphere Client-Server Architecture](#)
- [Web Services API as a Web Service](#)
- [Access to Managed Objects](#)
- [Access to vSphere Server Data](#)

vSphere Client-Server Architecture

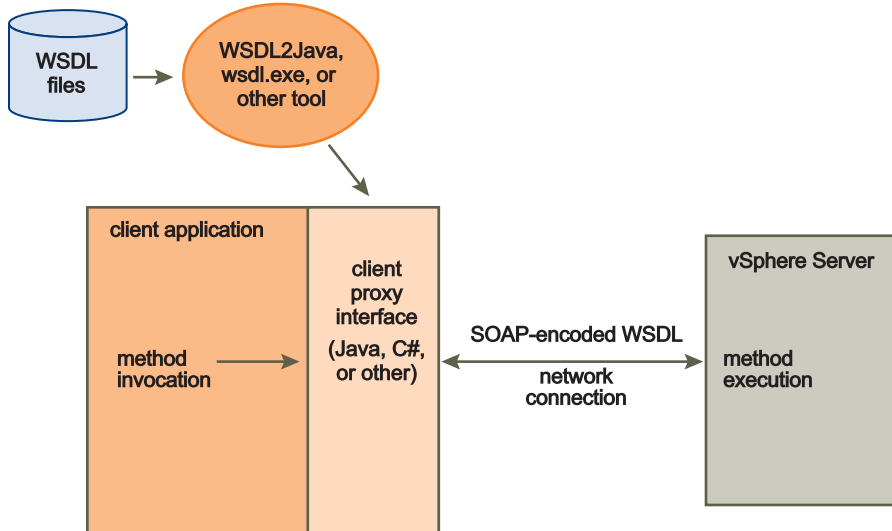
VMware vSphere client applications participate in a distributed architecture that uses an asynchronous communications model. This architecture is based on server-side managed objects, client-side managed object references, and data objects.

- Managed objects exist on a vSphere server (ESX/ESXi or vCenter Server system). They represent vSphere services and components. Services include `PropertyCollector`, `SearchIndex`, `PerformanceManager`, and `ViewManager`. Components include inventory objects such as `VirtualMachine`, `Datastore`, and `Folder`.
- Managed object references are client application references to server-side managed objects. Your client application uses `ManagedObjectReference` objects when it invokes operations on a server. A `ManagedObjectReference` is guaranteed to be unique and persistent during an object's lifetime. The reference persists after an object has moved within the inventory, across sessions, and across server restarts. If you remove an object, for example, a virtual machine, from the inventory, and then put it back, the reference changes.
- Data objects contain information about managed objects. Your client application sends data objects to and receives data objects from a vSphere server. Examples are the different specification and capability objects such as `VirtualMachineConfigSpec` and `HostCapability`.

The client proxy provides a language-specific interface proxy. The proxy facilitates remote method invocation, organization of object data, and other aspects of distributed, object-oriented, applications programming. Your client application calls proxy interface methods. The client proxy uses SOAP (Simple Object Access Protocol) to exchange WSDL messages with a vSphere server.

[Figure 2-2. Client-Server Communication Through a Client Proxy Interface](#) is a representation of a client application that uses the client proxy interface to call a method. The client proxy interface is based on the WSDL definitions.

Figure 2-2. Client-Server Communication Through a Client Proxy Interface



To use the VMware client proxy interface, you must import the vSphere Web Services API client libraries in to your client application using the following Java statement.

```
import com.vmware.vim25.*;
```

Important The vSphere Web Services SDK includes Java client-side proxy code that was generated using the JAX-WS toolkit. If the versions of Java and JAX-WS on your development platform are the same as those used to generate the proxy interface shipped in the SDK, you do not have to generate client-side proxy code from the WSDL. See the *Developer's Setup Guide* for information about how to configure a development environment for the vSphere Web Services SDK.

Network Access to the vSphere Web Service

Your client application can use the vSphere Web Services API to communicate with vSphere servers over HTTPS (HTTP over an encrypted Secure Sockets Layer connection) at port 443. HTTPS is the default protocol.

You can configure the server to support HTTP. Use HTTP access only for test or development environments, not for production. For information about how to configure the server to support HTTP access, see the *vSphere Web Services SDK Developer's Setup Guide*.

Language-Specific Classes and Methods

The SOAP tools generate language-specific classes and methods that match the WSDL definitions. The tools also produce objects and methods that are not in the WSDL files.

- **Generated objects.** The additional objects provide access to the vSphere Web Service to establish the client-server connection (`VimService`, `AppUtil`) and declare the methods defined for the vSphere API (`VimPortType`, `VimService`).
- **Generated methods.** The additional methods are accessor (getter) and mutator (setter) methods for properties. For Java, the method names are constructed by adding `get` and `set` prefixes to a property name, and changing the first character of the property name to upper case.

The following table identifies client proxy definitions for the vSphere Web Services SDK WSDL.

Element Access	Java
Access to vSphere Web service (HTTPS/HTTP)	<code>VimService</code> class
Access to vSphere API methods	<code>VimPortType</code> class
Access to vSphere API properties	<code>getPropertyName</code> and <code>setPropertyName</code> methods defined for data objects
vSphere API data objects	Data objects in the vSphere API (see the <i>vSphere API Reference</i>) defined as objects in the proxy interface

The following code fragments show getter and setter method definitions for the `AfterStartupTaskScheduler.minute` property.

Java

```
public int getMinute() {
    return minute; }
public void setMinute(int minute) {
    this.minute = minute; }
```

You can extrapolate the getter and setter methods that are available in the client proxy interface from the *vSphere API Reference*. For example, the `ScsiLun` data object has a `displayName` property. For the Java API, you can use a `setDisplayDisplayName` method to assign a string value to the property, and obtain the string value by using the `getDisplayDisplayName` method. The vSphere Web Services SDK includes Java sample code that illustrates how to use the proxy interfaces. See [Chapter 3 Client Applications for the Web Services API](#).

Mapping XML Data Types to Java or JSON Data Types

In this guide, the UML class and object diagrams use the primitive data type names such as string and integer, without the XML Schema definition namespace prefix (`xsd:`). The *vSphere API*

Reference contains the complete data type name, such as `xsd:string`. The data types map to the primitive data types of the programming language used for the client application.

The following table lists some of the more common XML primitive data type mappings.

XML Schema	Java	Open API
<code>xsd:base64binary</code>	<code>byte[]</code>	<code>string</code> (format: byte)
<code>xsd:boolean</code>	<code>boolean</code>	<code>boolean</code>
<code>xsd:byte</code>	<code>byte</code>	<code>integer</code> (min: -128, max: 127)
<code>xsd:dateTime</code>	<code>java.util.Calendar</code>	<code>string</code> (format: date-time)
<code>xsd:decimal</code>	<code>java.math.BigDecimal</code>	
<code>xsd:double</code>	<code>double</code>	<code>number</code> (format: double)
<code>xsd:float</code>	<code>float</code>	<code>number</code> (format: float)
<code>xsd:int</code>	<code>int</code>	<code>integer</code> (format: int32)
<code>xsd:string</code>	<code>java.lang.String</code>	<code>string</code>

Access to Managed Objects

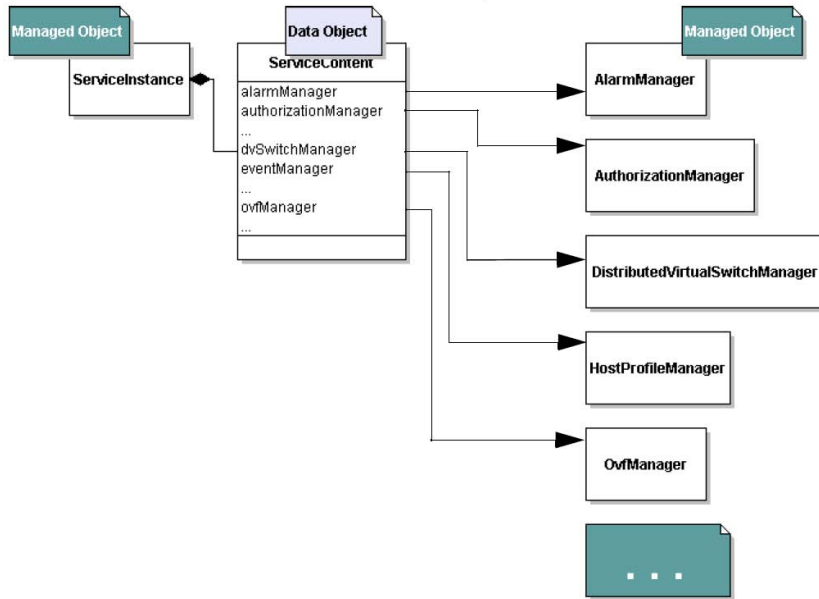
Your client application obtains access to managed objects through the `ServiceInstance` managed object and its associated `ServiceContent` data object. The `ServiceContent` data object contains managed object references to services and manager entities, and to the root folder of the inventory.

The `ServiceInstance` managed object is the root object of the inventory on both ESX/ESXi and vCenter Server systems. The server creates the `ServiceInstance`, and creates the manager entities that provide services in the virtual environment. Examples of manager entities are `LicenseManager`, `PerformanceManager`, and `ViewManager`.

The `ServiceInstance` is the primary point of access to the server inventory. Your client application starts by connecting to a server and creating a reference to the `ServiceInstance`. After you have connected to the server, you can call the `ServiceInstance.RetrieveServiceContent` method to a `ServiceContent` data object. `ServiceContent` provides access to the vSphere managed object services. See [Overview of a Java Sample Application for the Web Services SDK](#) for an example of connecting to a server and using the `ServiceInstance` reference to retrieve the `ServiceContent` object.

Figure 2-3. ManagedObjectReference Data Object shows the object model for the `ServiceInstance` and `ServiceContent` objects. The figure shows some of the `ServiceContent` managed object references and the target objects of the references. Each managed object reference identifies a specific managed object on the server with its type and a value. (The `value` property is an opaque string.)

Figure 2-3. ManagedObjectReference Data Object



Access to vSphere Server Data

To obtain information about the virtual infrastructure, you retrieve managed object properties. Managed object properties can be simple data types, such as integer or string data, or they can be complex types such as data objects that contain sets of properties.

Obtaining Information from a Server

With a reference to a managed object, you can obtain information about the state of the server-side inventory objects and populate client-side data objects based on the values. You can use one of the following approaches:

- Use an accessor (getter) method. The client proxy interface provides accessor methods for each data object property. You can use these accessor methods to obtain the values of the object. See [Language-Specific Classes and Methods](#) for information about client proxy interface accessor methods.
- Use a `PropertyCollector` to navigate to a selected point on the server and obtain values from specific properties. See [Chapter 6 Property Collector](#) for more information about `PropertyCollector`.

- Use the `SearchIndex` managed object to obtain a managed object reference to the managed entity of interest. The `SearchIndex` can return managed object references to specific managed entities—`ComputeResource`, `Datacenter`, `Folder`, `HostSystem`, `ResourcePool`, `VirtualMachine`—given an inventory path, IP address, or DNS name.

Important You can use API methods to operate on managed objects in the vSphere inventory. A method that updates properties in one managed object may also update properties in other managed objects. The Server performs asynchronous updates to the inventory. There is no guarantee that the inventory will be completely updated when the method returns to the caller. Use the `PropertyCollector` method `WaitForUpdatesEx` to obtain property changes.

Working with Data Structures

Properties contain information about the server-side objects at a given point in time. The value of a property can be of one of the following types:

- Simple data types, such as a string, boolean, or integer (or other numeric) data type. For example, the `ManagedEntity` managed object has a `name` property that takes a string value.
- Arrays of simple data types or data objects. For example, a `HostSystem` managed object contains an array of managed object references (a type of data object) to virtual machines hosted by that physical machine. As another example, the `SessionManager` managed object has a `sessionList` property that is an array of `UserSession` data objects.
- Enumerated types (enumeration, enum) of predefined values. The values can be a collection of simple data types or data objects. For example, a virtual machine's power state can be one of three possible string values—`poweredOn`, `poweredOff`, or `suspended`.

The type of a property is often a string, but the property actually expects one of the values an enumeration encapsulates. For example, when you set `VirtualMachineConfigSpec.guestid` you can specify one of the elements of the `VirtualMachineGuestOSIdentifier` as a string.

- Complex (or composite) data types. For example, the `HostProfileConfigInfo` object contains data objects, an array of data objects, and an array of strings.

Accessing Property Values

To use the composite data structures and arrays that contain Server data:

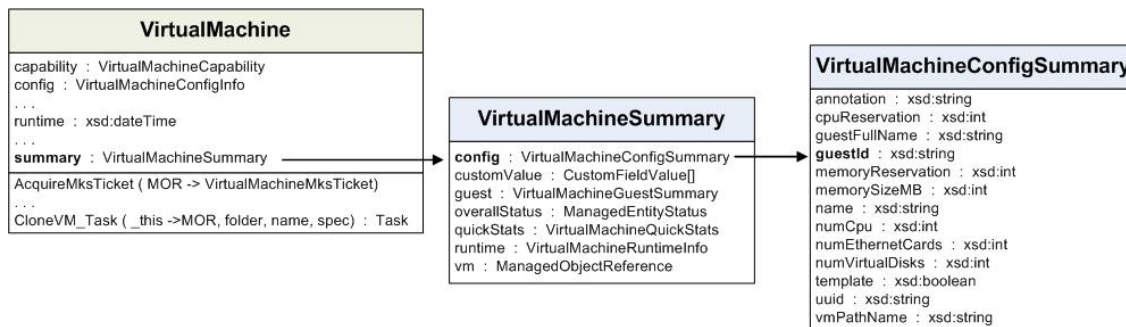
- Use dot notation to access nested properties in composite data structures.
- Cast unconstrained property values (`xsd:anyType`) to array types.
- Use keys or index values as appropriate to access array values.

Nested Properties and Property Paths in Composite Data Structures

vSphere Data objects can include properties that are defined as composite data types, such as data objects. The embedded data objects can also contain properties that are data objects. Properties can nest to several levels.

For example, the following figure shows a UML class diagram of the `VirtualMachine` managed object, which has a `runtime` property that is defined as an `xsd:dateTime` data type. `VirtualMachine` also has a `summary` property that is a `VirtualMachineSummary` data object. The `VirtualMachineSummary` data object contains a `config` property that is a `VirtualMachineConfigSummary` data object.

Figure 2-4. VirtualMachine Managed Object and Nested Properties



To refer to a nested property, use dot notation to separate the object names in the sequence that defines the path to the property. Your code must handle the type referenced at the end of the sequence.

For example, you can compare the property referenced by the path `summary.config.guestId` (a string value) to the property referenced in the path `summary.config` (the complete `VirtualMachineSummary` data object).

The following table shows examples of property references and the corresponding data types for some of the properties of the `VirtualMachine` managed object shown in [Figure 2-4](#).

VirtualMachine Managed Object and Nested Properties.

Reference	Data Type
<code>summary</code>	<code>VirtualMachineSummary</code> data object
<code>summary.config</code>	<code>VirtualMachineConfigSummary</code> data object
<code>summary.config.guestID</code>	string

xsd:anyType Arrays

The vSphere Web Services API uses `xsd:anyType` unconstrained type declarations. A vSphere client must map values of `xsd:anyType` to explicit data types. An `xsd:anyType` value can represent a single data value or it can represent an array. The WSDL for the vSphere Web

Services API defines array types for all of the data values that a vSphere client can send or receive as arrays. The array types use the prefix “ArrayOf”. An example of an array type is `ArrayOfString` for string values.

When a client sends data to a vSphere Server, the client must use explicit datatypes. For example, a client can define a `MethodAction` for a `ScheduledTask`. The vSphere Web Services API defines arguments to the action (the `MethodActionArgument.value` property) as type `xsd:anyType`. If the action takes an array argument, the client must set the corresponding `MethodAction.argument[]` to the appropriate `ArrayOf...` type.

When a client receives `xsd:anyType` data from a vSphere server, it must cast the data to an explicit type. For example, the `PropertyCollector` method `RetrievePropertiesEx` returns a set of `ObjectContent` data objects. The `ObjectContent.propSet` property is a list of `DynamicProperty` objects that contains the requested property values. Each `DynamicProperty` object contains a name-value pair. The value property (`DynamicProperty.val`) is of type `xsd:anyType`. It can represent a single object or an array of objects.

When the returned value is a single object such as an `Event`, `ManagedObjectReference`, or `String`, you can cast it directly to a variable of the appropriate type. However, when the value is an array of objects you cannot cast the `anyType` value directly to an array variable.

When the `PropertyCollector` returns array data, it sends it as an `xsd:anyType` value. The language-specific bindings contain definitions for array objects such as `ArrayOfEvent`, `ArrayOfManagedObjectReference`, and `ArrayOfString`, and corresponding “get” methods. To extract the actual array from a property of type `xsd:anyType`, cast `DynamicProperty.val` to the appropriate array type and use the matching get method – for example, `getEvent()`, `getManagedObjectReference()`, or `getString()`.

The following sections provide some examples of how to cast returned values for a few of the array types.

Cast an `xsd:anyType` Array to an Event Array

When the `PropertyCollector` returns array data representing an array of `Event` objects, the array has the type `xsd:anyType`. To use the data in your application, you must first cast it to a specialized type for `Event` arrays.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

Procedure

- 1 Use the `DynamicProperty.getVal()` method to retrieve the `anyType` property value.
- 2 Cast the `anyType` value to a value of type `ArrayOfEvent`.
- 3 Use the corresponding get method to assign the result of the cast operation to a list variable.

Example

```

/*
 * Handling arrays of Event objects.
 * Cast the return value to ArrayOfEvent and use getEvent().
 */
List[] eventList = ((ArrayOfEvent) dynamicProp.getVal()).getEvent();

```

Cast an `xsd:anyType` Array to an Array of Managed Object References

When the `PropertyCollector` returns array data representing an array of managed objects, the array has the type `xsd:anyType`. To use the data in your application, you must first cast it to a specialized type for arrays of managed object references.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

Procedure

- 1 Use the `DynamicProperty.getVal()` method to retrieve the `anyType` property value.
- 2 Cast the `anyType` value to a value of type `ArrayOfManagedObjectReference`.
- 3 Use the corresponding get method to assign the result of the cast operation to a list variable.

Example

```

/*
 * Handling arrays of ManagedObjectReference objects.
 * Cast the return value to ArrayOfManagedObjectReference and use getManagedObjectReference().
 */
List[] morList =
    ((ArrayOfManagedObjectReference) dynamicProp.getVal()).getManagedObjectReference();

```

Cast an `xsd:anyType` Array to an Array of String

When the `PropertyCollector` returns array data representing an array of strings, the array has the type `xsd:anyType`. To use the data in your application, you must first cast it to a specialized type for arrays of strings.

This example code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL.

Procedure

- 1 Use the `DynamicProperty.getVal()` method to retrieve the `anyType` property value.
- 2 Cast the `anyType` value to a value of type `ArrayOfString`.
- 3 Use the corresponding get method to assign the result of the cast operation to a list variable.

Example

```
/*
 * Handling arrays of strings.
 * Cast the return value to ArrayOfString and use getString().
 */
List[] stringList = ((ArrayOfString) dynamicProp.getVal()).getString();
```

Indexed Array and Key-Based Array Properties

The VMware vSphere data structures include array properties, which can be indexed arrays or key-based arrays.

- **Indexed arrays** are accessed by using an index integer. Indexed arrays are used for arrays of data types whose positions in the array do not change. For example, the `roleList` property of the `AuthorizationManager` managed object is an array of authorization roles. Adding a new role to the array does not change the position of existing elements in the array.
- **Key-based arrays** are used for information whose position is subject to change. A key-based array (same basic concept as a Perl hash or a Python dictionary) uses a unique, unchanging value as a key to access an element's value.

Typically, the key is a string, but integers can also be used. For example, `Event` arrays use integers as keys.

The vSphere management object model uses key-based arrays to track managed object references. The contents of a key-based array property are accessed by the value of either the key property or, in the case of a managed object reference, its value property. The value of these fields is unique across all of the components of an array.

Nested properties can also refer to entries in a key-based array. For example, `a.b.c["xyz"]` refers to the property `c` that has the key value of `xyz`.

Unset Optional Properties

Many of the Data Objects in the vSphere Web Services SDK have optional properties that may be set by your client application or by a Server process or event. If you retrieve a data object that has an optional property that is unset, the Server will not return a value for the optional property. If you call an accessor function to retrieve the property value, the value returned by the function depends on the programming language that you are using.

For example, if you are programming in Java, the value you will receive for an unset property is `null`.

[Figure 2-5. Data Object - HostFirewallInfo Properties](#) shows part of the Properties table for the `HostFirewallInfo` data object in the *vSphere Web Services SDK API Reference*. When you look at properties in the *vSphere Web Services SDK API Reference*, you can see that optional properties are marked with a red asterisk.

In this example, that the `defaultPolicy` property is always returned, but the `ruleset` property will be returned as a `null` value if it has not been set.

Figure 2-5. Data Object - HostFirewallInfo Properties

Properties		
NAME	TYPE	DESCRIPTION
defaultPolicy	HostFirewallDefaultPolicy	Default firewall policy.
ruleset*	HostFirewallRuleset[]	List of configured rulesets.
Properties inherited from DynamicData		
dynamicProperty , dynamicType		
* Need not be set		

Since Data Objects are part of many different constructs, there is no standard scenario for when an optional property should be set, what will happen if an optional property is left unset, or what you should do if a null value is returned.

Escape Character in Name and Path Properties

The percent sign (%) is used as an escape character to embed special characters in strings. For example, %2f (or %2F) is interpreted as the slash (/) character. To include a percent sign as a literal in a string, use %%. The path to the inventory starts from the root folder (`ServiceContent.rootFolder` property), denoted by the slash character.

Character	Description	Representation in URL
%	Percent sign	%25
/	Slash	%2F, %2f
\	Backslash	%5C, %5c
-	Dash	%2D, %2d
.	Dot	%2E, %2e
"	Double quotation mark	%2B, %2b

Client Applications for the Web Services API

3

Any client application written for the Web Services API must perform certain basic functions. These include making a connection to the server, authenticating and creating a session, and closing the connection.

This chapter includes the following topics:

- [vCenter Server Connections](#)
- [Establishing a Single Sign-On Session with a vCenter Server](#)
- [LoginByToken to vCenter Server By Using Java](#)
- [Establishing a Session with Username and Password Credentials](#)
- [Overview of a Java Sample Application for the Web Services SDK](#)
- [Accessing the vSphere Server from a Web Services Client](#)
- [Closing the Connection from a Web Services Client](#)
- [Using the Java Samples as Reference](#)
- [Multiple Versions of the vSphere API](#)
- [Java Sample Applications in the Web Services SDK](#)

vCenter Server Connections

Every vCenter Server client application must connect to the server and pass user account credentials to authenticate to the server. After the connection has been established, the client application can use vSphere services to access the virtual environment.

vSphere uses SSL certificates, HTTP tokens, and vCenter Single Sign-On tokens to authenticate a client and support a persistent connection between the client and vCenter Server. The following table provides an overview of these elements.

Security Element	Description
SSL certificates	vSphere Servers use standard X.509 version 3 (X.509v3) certificates to encrypt session information sent over Secure Socket Layer (SSL) protocol connections. In a production environment, client applications verify the vSphere Server certificate during the connection sequence. The examples in this chapter and the examples in the vSphere Web Services SDK accept all certificates.
HTTP tokens	A vSphere Server uses an HTTP token to identify a client session. The Server provides the HTTP token in its response to a client connection request. Subsequent messages between the client and the Server include the HTTP token in the HTTP header.
Client authentication vCenter Single Sign On token	vSphere supports vCenter Single Sign-On. A vCenter client can obtain a vCenter Single Sign-On token from a vCenter Single Sign-On Server and use that token to login to a vCenter Server.
Client authentication username/ password	Username/password authentication for client-server connections. A client can present user credentials either directly to vCenter Server to establish a session, or to the vCenter Single Sign-On Service in exchange for a SAML token.

Establishing a Single Sign-On Session with a vCenter Server

vSphere uses single sign-on to provide a single point of authentication for clients. vSphere includes the vCenter Single Sign-On Server. To use vCenter Single Sign-On, your client obtains a SAML token (Security Assertion Markup Language) from the vCenter Single Sign-On Server and passes the token to the vCenter Server in the login request. The token represents the client and contains claims that support client authentication. Components in the vSphere environment perform operations based on the original authentication. For information about obtaining a vCenter Single Sign-On token from the vCenter Single Sign-On Server, see *vCenter Single Sign On Programming Guide*.

To use single sign on, your client calls the `LoginByToken` method. Your client must send a SAML token to the vCenter Server by embedding the token in the SOAP header for the `LoginByToken` request. During the login sequence, your client must save and restore the HTTP session cookie. The vCenter Single Sign-On SDK contains sample code that demonstrates how to use the `LoginByToken` method.

The following sections describe examples of using the `LoginByToken` method to establish a vCenter Single Sign On session with a vCenter Server.

LoginByToken to vCenter Server By Using Java

The following example is based on the `LoginByTokenSample.java` file contained in the vCenter Single Sign On SDK. The SDK contains Java code that supports HTTP and SOAP header manipulation.

vCenter Server Single Sign-On Session Using Java

After you obtain a SAML token from the vCenter Single Sign-On Server, you can use the vSphere Web Services API method `LoginByToken` to establish a vCenter Single Sign-On session with a vCenter Server. At the beginning of the session, your client is responsible for the following tasks:

- Insert the vCenter Single Sign-On token and a timestamp into the SOAP header of the `LoginByToken` message.
- Maintain the vCenter session cookie. During the login sequence, the Server produces an HTTP session cookie to support the persistent connection. Your client must save this cookie and re-introduce it at the appropriate times.
- If at a later time your client invokes the `LoginByToken` method, or other login method, the Server issues a new session cookie in response. You must have a cookie handler in place to save the cookie for subsequent requests.

The example program uses these general steps:

Procedure

- 1 Call the `RetrieveServiceContent` method to establish an HTTP connection with the vCenter Server and get the Session Manager managed object reference.
- 2 Call the `LoginByToken` method to authenticate the vCenter session. To send the token to the vCenter Server, the client uses a handler to embed the token and a time stamp in the SOAP header for the message. The client uses an HTTP header handler method to extract the session cookie from the vCenter Server response.
- 3 Restore the session cookie for future requests. To identify the session started with the `LoginByToken` method, the client uses a handler to embed the session cookie in the HTTP header.

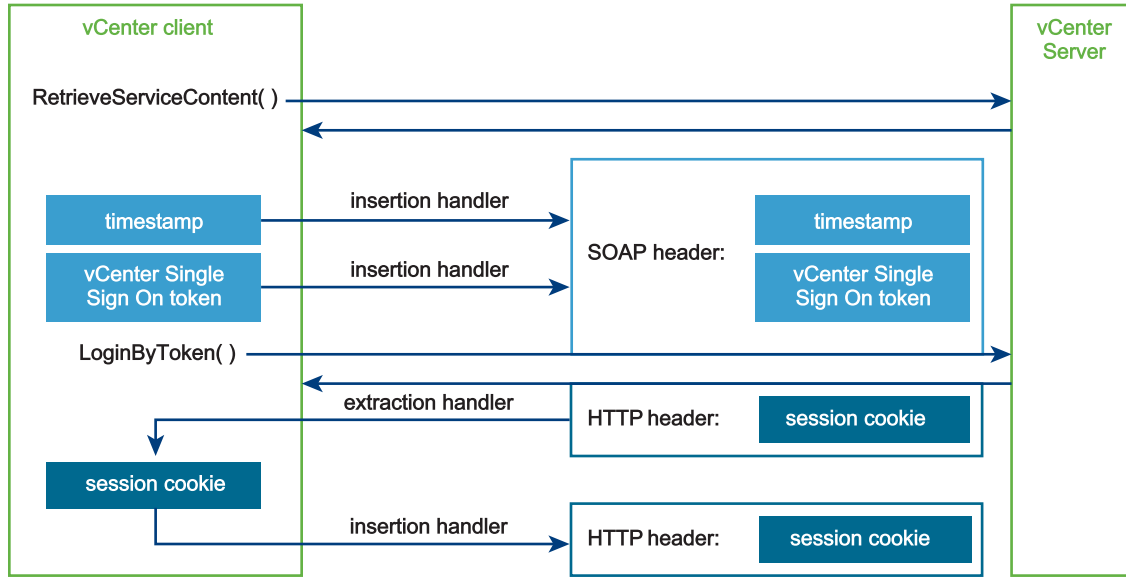
HTTP and SOAP Header Handlers in Java

To use a vCenter Single Sign On token to login to a vCenter Server, the example uses header handlers to manipulate the HTTP and SOAP header elements of the login request. After establishing a handler, subsequent requests automatically invoke the handler.

- Insertion handlers put the vCenter Single Sign On token and a timestamp into the SOAP header into the HTTP header of the login request.
- An extraction handler obtains the HTTP session cookie provided by the vCenter Server. After setting up the handler, a call to the `LoginByToken` method will invoke the handler to extract the cookie from the Server response.

The following figure shows the use of handlers to manipulate header elements when establishing a vCenter Single Sign On session with a vCenter Server.

Figure 3-1. Starting a vCenter Session



Important Every call to the vCenter Server will invoke any message handlers that have been established. The overhead involved in using the SOAP and HTTP message handlers is not necessary after the session has been established. The example saves the default message handler before setting up the SOAP and HTTP handlers. After establishing the session, the example will reset the handler chain and restore the default handler.

The example code also uses multiple calls to the `VimPortType.getVimPort` method to manage the request context. The `getVimPort` method clears the HTTP request context. After each call to the `getVimPort` method, the client resets the request context endpoint address to the vCenter Server URL. After the client has obtained the session cookie, it will restore the cookie in subsequent requests.

Sample Code for a Java Client to the Web Services SDK

The code examples in the following sections show how to use the `LoginByToken` method with a holder-of-key security token. The code examples are based on the sample code contained in the vCenter Single Sign On SDK. The files are located in the Java samples directory (`SDK/ssoclient/java/JAXWS/samples`):

- **LoginByToken sample:**

```
samples/com/vmware/vsphere/samples/LoginByTokenSample.java
```

- **Header cookie handlers:**

```
samples/com/vmware/vsphere/soaphandlers/HeaderCookieHandler.java
```

```
samples/com/vmware/vsphere/soaphandlers/HeaderCookieExtractionHandler.java
```

- SOAP header handlers. These are the same handlers that are used in the vCenter Single Sign-On example in *vCenter Single Sign On Programming Guide*. The SOAP handler files are contained in the vCenter Single Sign-On SDK and are located in the SSO client `soaphandlers` directory:

```
SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers
```

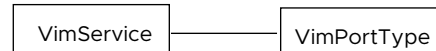
Creating the HTTP Connection in Java

The code fragment in this section establishes an HTTP session with the vCenter Server and saves the HTTP session cookie.

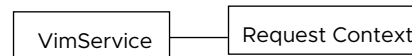
The following sequence describes these steps and shows the corresponding objects and methods.

- 1 Use the `getHandlerResolver` method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unnecessary after login. `VimService.getHandlerResolver()`

- 2 Get the VIM port. The VIM port provides access to the vSphere API methods, including the `LoginByToken` method.



- 3 Set the request context endpoint address to the vCenter Server URL.



- 4 Retrieve the `ServiceContent`. This method establishes the HTTP connection.



The following example shows Java code that saves the session cookie.

Example: Saving the vCenter Server Session Cookie

```

/*
 * The example uses a SAML token (obtained from a vCenter Single Sign On Server)
 * and the vCenter Server URL.
 * The following declarations indicate the datatypes; the token datatype (Element) corresponds
 * to the token datatype returned by the vCenter Single Sign On Server.
 *
 * Element token;          -- from vCenter Single Sign On Server
 * String vcServerUrl;     -- identifies vCenter Server
 *
 * First, save the default message handler.
 */

HandlerResolver defaultHandler = vimService.getHandlerResolver();

/*
 * Create a VIM service object.
 */
  
```

```

vimService = new VimService();

/*
 * Construct a managed object reference for the ServiceInstance.
 */
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

/*
 * Get the VIM port for access to vSphere API methods. This call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Get the request context and set the connection endpoint.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Retrieve the ServiceContent. This call establishes the HTTP connection.
 */
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);

```

Using LoginByToken in Java

The code fragment in this section sets up the message handlers and calls the `LoginByToken` method to get the session cookie. The following sequence describes the steps and shows the corresponding objects and methods.

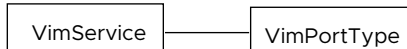
- 1 Create a new `HeaderHandlerResolver`. Then set the message security handlers for cookie extraction and for inserting the SAML token and credentials in the SOAP header.

```

HeaderHandler Resolver
— HeaderCookieExtractionHandler (session cookie)
— TimestampHandler
— SamlTokenHandler (SAML token)
— WsSecurityUserCertificateSignatureHandler (key, certificate, ||

```

- 2 Get the VIM port.



- 3 Set the connection endpoint in the HTTP request context.



- | | | |
|-------|---|--|
| 4 | Call the <code>LoginByToken()</code> method. The method invocation activates the handlers to insert the elements into the message headers. The method returns a session cookie that identifies the newly created session. | <code>VimPortType.LoginByToken()</code> |
| <hr/> | | |
| 5 | Extract the cookie and save it for later use. | <code>HeaderCookieExtractionHandler.getCookie()</code> |

The following examples shows Java code that calls the `LoginByToken()` method.

Example: Using LoginByToken

```

/*
 * Create a handler resolver and add the handlers.
 * Create a cookie extraction handler and add it to the handler resolver.
 */
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
handlerResolver.addHandler(cookieExtractor);
handlerResolver.addHandler(new TimestampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
handlerResolver.addHandler(new WsSecuritySignatureAssertionHandler(
                                userCert.getPrivateKey(),
                                userCert.getUserCert(),
                                Utils.getNodeProperty(token, "ID")));
vimService.setHandlerResolver(handlerResolver);

/*
 * Create a handler resolver.
 * Set the VIM service handler resolver.
 */
vimService.setHandlerResolver(handlerResolver);

/*
 * Get the Vim port; this call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Retrieve the request context and set the server URL.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

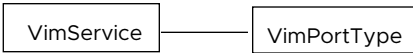
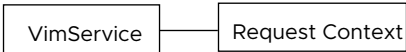
/*
 * Call LoginByToken.
 */
UserSession us = vimPort.loginByToken(serviceContent.getSessionManager(), null);

/*
 * Save the HTTP cookie.
 */
String cookie = cookieExtractor.getCookie();

```

Restoring the vCenter Server Session Cookie in a Java Client

After you log in, you must restore the standard vCenter Server session context. The code fragment in this section restores the default message handler and the session cookie. As the cookie handler has been replaced by the default handler, the client resets the session cookie by calling request context methods to access the context fields directly. The following sequence describes these steps and shows the corresponding objects and methods.

1	Restore the default message handler. The handlers used for <code>LoginByToken()</code> are not used in subsequent calls to the vSphere API.	<code>VimService.setHandlerResolver()</code>
2	Get the VIM port.	 <pre> graph LR VimService[VimService] --- VimPortType[VimPortType] </pre>
3	Set the connection endpoint in the HTTP request context.	 <pre> graph LR VimService[VimService] --- RequestContext[Request Context] </pre>
4	Set the HTTP request header (vCenter Server session cookie).	<code>RequestContext.get()</code> <code>RequestContext.put()</code>

The following example shows Java code that restores the vCenter Server session. This code requires the vCenter Server URL and the cookie and default handler that were retrieved before login. See [Sample Code for a Java Client to the Web Services SDK](#).

Example: Restoring the vCenter Server Session

```

/*
 * Reset the default handler. This overwrites the existing handlers, effectively removing
 * them.
 */
vimService.setHandlerResolver(defaultHandler);
vimPort = vimService.getVimPort();

/*
 * Restore the connection endpoint in the request context.
 */
// Set the validated session cookie and set it in the header for once,
// JAXWS will maintain that cookie for all the subsequent requests

Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Reset the cookie in the request context.
 */
Map<String, List<String>> headers =
    (Map<String, List<String>>) ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
    headers = new HashMap<String, List<String>>();
}

```

```
headers.put("Cookie", Arrays.asList(cookie));  
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
```

Establishing a Session with Username and Password Credentials

You can specify username and password credentials to establish a session with a vCenter Server. The following steps describe how a client application specifies username and password credentials for access to a vCenter Server.

Procedure

- 1 Create a connection to the vSphere server Web service.
- 2 Instantiate a local proxy object for reference to `ServiceInstance`. Use this local proxy object to retrieve the `ServiceContent` object from the server. `ServiceContent` contains a reference to the root folder for the inventory and references to the managed objects that provide the vSphere services.
- 3 Instantiate a local proxy object for access to vSphere API methods.
- 4 Log in to the server using appropriate credentials (user account, password, and optionally the locale).
- 5 Access server-side objects to retrieve data and perform management operations.
- 6 Close the connection.

Overview of a Java Sample Application for the Web Services SDK

This section includes an example of a complete client application that demonstrates the basic client capability. The sample client application prints out the product name, server type, and product version to demonstrate that it is connected and able to retrieve information from the server.

While [Java Test Client Application](#) is a complete client application that demonstrates the basic client capability, it uses a slightly different format than the Java sample files in the `SDK\` directory. This example, and the Java samples that are included with your vSphere Web Service SDK, have been compiled using JAX-WS bindings.

Most of the vSphere Web Services SDK samples do not handle exceptions, and they accept all security certificates. Use the samples as examples for extracting the types of data you want to view, but do not use these security or exception techniques in your production applications.

Build a Simple vSphere Client Application for the Web Services SDK

This simple client application accepts command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

To build a simple vSphere client application in Java, use the following steps.

Procedure

- 1 Import the vSphere Web Services API libraries:

```
import com.vmware.vim25.*;
```

- 2 Import the necessary Java (and JAX-WS connection, bindings, and SOAP) libraries:

```
import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;
```

- 3 Create the `TestClient` class:

```
public class TestClient {
```

- 4 Include the class variable declarations/definitions. Use a `TrustManager` class to accept all certificates, as shown in [Accessing the vSphere Web Services HTTP Endpoint with JAX-WS](#) . This is not appropriate for a production environment. Production code should implement certificate support.
- 5 Use the vSphere Web Services APIs to create the connection, as shown in [Accessing the vSphere Server from a Web Services Client](#).
- 6 Retrieve data from the vSphere or vCenter Server. In this example, we are just going to print out the product name, server type, and product version to prove that the client is connected and working correctly.

```
System.out.println(serviceContent.getAbout().getFullName());
System.out.println("Server type is " + serviceContent.getAbout().getApiType());
System.out.println("API version is " + serviceContent.getAbout().getVersion());
```

- 7 Use the `VimPort` object to close the connection, as shown in [Closing the Connection from a Web Services Client](#). Always close your server connections to maintain security.

Java Client Example for the Web Services SDK

This example shows the complete sample client application code, without the explanatory steps. The example opens a connection with the server, retrieves the service content, uses the session manager managed object reference to log in, displays information about the server, and closes the connection.

Example: Java Test Client Application

```
import com.vmware.vim25.*;
import java.util.*;
```

```

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

public class TestClient {

    // Authentication is handled by using a TrustManager and supplying
    // a host name verifier method. (The host name verifier is declared
    // in the main function.)
    private static class TrustAllTrustManager
        implements javax.net.ssl.TrustManager,
            javax.net.ssl.X509TrustManager {

        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return null;
        }

        public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }

        public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
            return true;
        }

        public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
            String authType)
            throws java.security.cert.CertificateException {
            return;
        }

        public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
            String authType)
            throws java.security.cert.CertificateException {
            return;
        }
    }
}

// end class TrustAllTrustManager

public static void main(String[] args) {
    try {

        // Server URL and credentials.
        String serverName = args[0];
        String userName   = args[1];
        String password   = args[2];
        String url = "https://" + serverName + "/sdk/vimService";
        // Variables of the following types for access to the API methods
        // and to the vSphere inventory.
        // -- ManagedObjectReference for the ServiceInstance on the Server
        // -- VimService for access to the vSphere Web service
        // -- VimPortType for access to methods
        // -- ServiceContent for access to managed object services
        ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
    }
}

```

```

VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;

// Declare a host name verifier that will automatically enable
// the connection. The host name verifier is invoked during
// the SSL handshake.
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};
// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;

// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
// Set the default host name verifier to enable the connection.
HttpURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server.
// The client API will include the Server's HTTP cookie in its
// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),

```

```

        userName,
        password,
        null);

    // print out the product name, server type, and product version
    System.out.println(serviceContent.getAbout().getFullName());
    System.out.println("Server type is " + serviceContent.getAbout().getApiType());
    System.out.println("API version is " + serviceContent.getAbout().getVersion());

    // close the connection
    vimPort.logout(serviceContent.getSessionManager());
} catch (Exception e) {
    System.out.println(" Connect Failed ");
    e.printStackTrace();
}
}
} //end main()
} // end class TestClient

```

// See [Obtaining a Session Token - Code Fragments from VMPromoteDisks.java](#) for more details.

Compiling the Java Test Client in the Web Services SDK

Use the following command to compile the code for the Java Test Client Application, after you have saved it as a `.java` file:

```
c:>javac -classpath path-to-vim25.jar TestClient.java
```

Use the following command to run the compiled class (binary) file:

```
c:>java -classpath path-to-vim25.jar TestClient web-service-url user-name user-
password
```

Web Server Session Token

As with other Web services, the vSphere Web service maintains session state for each client connection by using a token in the HTTP header to identify the session. The vSphere server returns a session token to the client in its response to the client connection request. Subsequent messages between client and server automatically include the token.

Each of the stand-alone samples in the `SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\` uses the JAX-WS `TrustAllTrustCertificates` class, as discussed in [Obtaining a Session Token - Code Fragments from VMPromoteDisks.java](#) to ignore certificates, obtain a session token, and then connect to the server.

Caution We do not recommend that you trust all certificates in a production environment. Instead, you can look at the sample code to see how the JAX-WS libraries are used when making the connection, but set up an SSL policy that allows connection only with trusted certificates.

The logic for getting a cookie and putting it in the header looks like this:

```

//cookie logic
List cookies = (List) headers.get("Set-cookie");
cookieValue = (String) cookies.get(0);

```

```
StringTokenizer tokenizer = new StringTokenizer(cookieValue, ";");
cookieValue = tokenizer.nextToken();
String path = "$" + tokenizer.nextToken();
String cookie = "$Version=\"1\"; " + cookieValue + "; " + path;

// set the cookie in the new request header
Map map = new HashMap();
map.put("Cookie", Collections.singletonList(cookie));

((BindingProvider) vimPort).getRequestContext().put(
    MessageContext.HTTP_REQUEST_HEADERS, map);
```

Accessing the vSphere Web Services HTTP Endpoint with JAX-WS

The steps for accessing any HTTP endpoint with JAX-WS bindings include the vSphere Web Services SDK Server URL, vSphere server object, and variables.

These steps are listed at the beginning of [Obtaining a Session Token - Code Fragments from VMPromoteDisks.java](#).

Procedure

- 1 Create a `TrustManager` class to handle certificate checking.

In this example we use a `TrustManager` class to accept all certificates. This is not appropriate for a production environment. Production code should implement certificate support.

```
private static class TrustAllTrustManager
    implements javax.net.ssl.TrustManager,
        javax.net.ssl.X509TrustManager {

    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(
        java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
        String authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
        String authType)
```

```
        throws java.security.cert.CertificateException {
            return;
        }
    }
}
```

2 Include the Server URL and credentials as arguments in the main method:

```
public static void main(String[] args) {
    try {
        String serverName = args[0];
        String userName   = args[1];
        String password   = args[2];
        String url = "https://" + serverName + "/sdk/vimService";
```

3 Declare variables of the following types for access to vSphere server objects:

- `ManagedObjectReference` for the `ServiceInstance`.
- `VimService` object for access to the Web service.
- `VimPortType` object for access to all of the methods defined in the vSphere API.
- `ServiceContent` for access to the managed object services on the server.

The following Java code fragment shows these variable declarations:

```
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;
```

4 Declare a host name verifier that will automatically enable the connection. The host name verifier is invoked during the SSL handshake.

```
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};
```

5 Instantiate the trust manager object.

```
// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
```

6 Create the SSL context

```
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
```

7 Create the session context

```
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
```

- 8 Initialize the contexts; the session context takes the trust manager.

```
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
```

- 9 Use the default socket factory to create the socket for the secure connection

```
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

- 10 Set the default host name verifier to enable the connection.

```
HttpURLConnection.setDefaultHostnameVerifier(hv);
```

Accessing the vSphere Server from a Web Services Client

The steps that use the vSphere Web Services API to create the connection are:

Procedure

- 1 Create a managed object reference for the `ServiceInstance` object on the server.

```
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
```

- 2 Create a `VimService` object to obtain a `VimPort` binding provider. The `BindingProvider` object provides access to the protocol fields in request/response messages. Retrieve the request context which will be used for processing message requests.

The `VimServiceLocator` and `VimPortType` objects provide access to vSphere servers. The `getVimPort` method returns a `VimPortType` object that provides access to the vSphere API methods.

```
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
```

- 3 Store the Server URL in the request context and specify `true` to maintain the connection between the client and server. The client API will include the server's HTTP cookie in its requests to maintain the session. If you do not set this to `true`, the server will start a new session with each request.

```
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
```

- 4 Retrieve the `ServiceInstance` content (the `ServiceContent` data object) and log in to the server.

```
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
```

```

        userName,
        password,
        null);
isConnected = true;

```

Closing the Connection from a Web Services Client

Use the `VimPort` object again to close the connection. Always close your server connections to maintain security.

```

        vimPort.logout(serviceContent.getSessionManager());
    } catch (Exception e) {
        System.out.println(" Connect Failed ");
        e.printStackTrace();
    }
} //end main()
} // end class TestClient

```

Using the Java Samples as Reference

The following code fragment from the `SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\vm\`

`VMPromoteDisks.java` sample shows another implementation of the server connection. Review the stand-alone Java samples that are shipped with your *vSphere Web Services SDK*, and use similar code to get a session token for your client application.

Example: Obtaining a Session Token - Code Fragments from `VMPromoteDisks.java`

```

.
.
.
private static String cookieValue = "";
private static Map headers = new HashMap();
.
.
.
private static void trustAllHttpsCertificates()
    throws Exception {

    javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
    javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
    trustAllCerts[0] = tm;
    javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
    javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
    sslsc.setSessionTimeout(0);
    sc.init(null, trustAllCerts, null);
    javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
}
...

```



```

private static void connect()
    throws Exception {

    HostnameVerifier hv = new HostnameVerifier() {
        public boolean verify(String urlHostName, SSLSession session) {
            return true;
        }
    };
    trustAllHttpsCertificates();
    HttpURLConnection.setDefaultHostnameVerifier(hv);

    SVC_INST_REF.setType(SVC_INST_NAME);
    SVC_INST_REF.setValue(SVC_INST_NAME);

    vimService = new VimService();
    vimPort = vimService.getVimPort();
    Map<String, Object> ctxt =
        ((BindingProvider) vimPort).getRequestContext();

    ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
    ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

    serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
    headers =
        (Map) ((BindingProvider) vimPort).getResponseContext().get(
            MessageContext.HTTP_RESPONSE_HEADERS);
    vimPort.login(serviceContent.getSessionManager(),
        userName,
        password, null);
    isConnected = true;

    propCollectorRef = serviceContent.getPropertyCollector();
    rootRef = serviceContent.getRootFolder();
}
...

```

Multiple Versions of the vSphere API

When a client application connects to a Web service running on a vSphere server (ESXi or vCenter Server system), the server detects the version of the API that was used to develop the client and makes available only those operations supported by the client.

Client applications convey information about the API version used in the SOAP messages that they send to a vSphere server. These SOAP messages include a `versionID` in the `soapAction` attribute. The details are handled transparently by the SOAP toolkit and the client proxy code. The server adjusts its behavior based on the client's version information, exposing the API version that the client supports to the client.

Starting with vSphere 4.0, information about the supported API versions is contained in an XML file, `vimServiceVersions.xml`, located on the server (see [Service-Versions File \(vimServiceVersions.xml\)](#)). You can access this file with the URL `https://server_hostname/sdk/vimServiceVersions.xml`.

Example: Service-Versions File (vimServiceVersions.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- Copyright 2008-2010 VMware, Inc. All rights reserved. -->
- <namespaces version="1.0">
  - <namespace>
    <name>urn:vim25</name>
    <version>5.0</version>
  - <priorVersions>
    <version>2.5u2</version>
    <version>2.5</version>
  </priorVersions>
</namespace>
- <namespace>
  <name>urn:vim2</name>
  <version>2.0</version>
</namespace>
</namespaces>
```

If you are developing a client application that must support multiple server versions at the same time (ESXi 5.0 and ESXi 5.5, for example), you must obtain information about the API versions that are supported on the server and provide logic in your code to use or not use features, based upon the version information.

One approach to targeting multiple versions of the API from the same client application code is to check for the existence of the server versions file on the server. If you do not find a `vimServiceVersions.xml` file on the server, the server is older than ESX/ESXi 4.x, vCenter Server 4.x.

Java Sample Applications in the Web Services SDK

The vSphere Web Services SDK includes sample applications, written in Java, that demonstrate features of the vSphere API and object model. Most of the samples do not handle exceptions, and they accept all security certificates. So use the applications as examples for extracting the types of data you want to view, but do not use the helper classes, trust store methods or exception handling techniques in your production environment.

Java Samples in the Web Services SDK

The Java samples in your vSphere Web Services SDK include `.java` files that you can compile and then run using any Java editor or IDE. The samples accept command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

Client Applications Using the JSON Protocol

4

You can choose to create client applications for the Web Services API by using the JSON protocol. This protocol offers functionality similar to the SOAP protocol, but provides a more REST-like interface and uses a separate service endpoint.

A JSON protocol client must perform certain basic functions, such as authenticating a session, retrieving data, and ending the session. This chapter outlines how to code basic functions, using Python to drive the JSON protocol.

This chapter includes the following topics:

- [Client Requirements for the JSON Protocol](#)
- [Outlining a Client for the JSON Protocol](#)
- [Building JSON Request URLs](#)
- [Building JSON Request Bodies](#)
- [Retrieving the Service Instance with the JSON Protocol](#)
- [Authenticating a JSON Client with the Session Manager](#)
- [Mapping Methods for the JSON Protocol](#)
- [Accessing Managed Object Properties with the JSON Protocol](#)
- [Using Polymorphic Types with the JSON Protocol](#)
- [Python Client Example for the JSON Protocol](#)

Client Requirements for the JSON Protocol

A client for the JSON protocol of the API needs a mid-level capability to build HTTP request messages, send the requests over the network, and manage server responses. Many languages have libraries to abstract the TCP/IP layers while also providing access to URLs, headers, and bodies.

For example, some popular libraries include:

Python 3

`urllib.request` module

Java

`java.net.HttpURLConnection` or `java.net.HTTP` module (for Java 11 and later)

Go

`net/http` package

Rust

`http` module

Perl

`LWP` module

Swift

`URL`, `URLRequest`, `URLSession`, and `URLSessonDataTask` interfaces

C++

`libcurl` or `curlpp`

Command line and graphical tools for passing HTTP messages include:

- `curl`
- `wget`
- Postman
- Insomnia
- Hoppscotch

To use the JSON protocol, you will build URLs that embed the names of managed objects, properties, and methods. You will build request bodies that specify the parameters for method calls. You will read response bodies and response headers, and you will include a session identifier in your request headers.

Outlining a Client for the JSON Protocol

A client to work with the JSON protocol generally contains the following parts, at minimum.

- The basic elements of the URL, including the address of the vCenter Server instance, the service endpoint specifier, and the version specifier.
- The credentials with which you authenticate your client session.
- Code to GET managed object properties, such as the `ServiceInstance` content, which contains managed object references to singletons, including the `SessionManager`.
- Code to POST requests to invoke methods on managed objects, such as the `Login` method.

- A request to fetch the `ServiceInstance` content and extract the `SessionManager` managed object reference.
- A request to authenticate the client session with the `SessionManager`.
- A request to end the client session.

The following sections use fragments of Python code to show how you can build these parts of the request.

Building JSON Request URLs

The service endpoint for the JSON API is determined by the vCenter Server address and the API version that the client prefers for the response. These data form part of the service URL.

All requests in the JSON API specify a managed object. Some requests retrieve properties of a managed object. Other requests invoke a method on a managed object.

The parts of the service URL are as follows:

- 1 `scheme = 'https://'`
- 2 `domain = vCenter Server IP address or fully qualified domain name`
- 3 The URL path contains the following parts:
 - a `endpoint = '/sdk/vim25'`
 - b `version = '8.0.1.0'`
 - c A managed object reference, generated by the server, which has two subparts, `type` and `value`, where:
 - 1 `type` is the name of a type of managed object, such as `Folder` or `VirtualMachine` or `ServiceInstance`
 - 2 `value` is a unique identifier (within this vCenter Server instance) for a specific object of the type named, such as `group-d1` or `vm-015` or `ServiceInstance`.

The object notation must be serialized as two strings separated by a slash (/). For example: `'Folder/group-d1'`.

```
def serialize_moref_for_request_url(moref):
    return('{}{}'.format(moref['type'], moref['value']))
```

- d A property name or method name, such as `childType` or `CreateVM`

An example request URL to retrieve the type or types of managed objects that a folder can contain::

```
>>> scheme = 'https://'
>>> domain = 'vcenter.example.com'
>>> endpoint = '/sdk/vim25'
>>> version = '8.0.1.0'
```

```
>>> motype = 'Folder'
>>> moid = 'group-d1'
>>> moref = motype + '/' + moid
>>> property = 'childType'
>>> url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + property
>>> print(url)
https://vcenter.example.com/sdk/vim25/8.0.1.0/Folder/group-d1/childType
```

An example request URL to cancel a long-running operation:

```
...
>>> motype = 'Task'
>>> moid = 'task-124'
>>> moref = motype + '/' + moid
>>> method = 'Cancel'
>>> url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + method
>>> print(url)
https://vcenter.example.com/sdk/vim25/8.0.1.0/Task/task-124/Cancel
```

Building JSON Request Bodies

Methods are invoked with the JSON protocol by sending a POST request with the method name in the URL and the method parameters in the body of the request.

The request body for a method invocation is a single JSON object containing the method parameters as named objects or arrays. For instance, a request body for the `MoveIntoFolder_Task` method might look like this:

```
{'list': [
  {'_typeName': 'ManagedObjectReference',
   'type': 'VirtualMachine',
   'value': 'vm-23'},
  {'_typeName': 'ManagedObjectReference',
   'type': 'VirtualMachine',
   'value': 'vm-96'},
  {'_typeName': 'ManagedObjectReference',
   'type': 'VirtualApp',
   'value': 'resgroup-v44'}
]}
```

The method name itself becomes part of the URL, as described in [Building JSON Request URLs](#).

Note Use the WSDL forms of type names. WSDL names are shown in the *vSphere Web Services API* reference. For example, use the WSDL name `ApplyHostProfileConfigurationSpec`, rather than the package name `vim.profile.host.ProfileManager.ApplyHostConfigSpec`, to build a data object as a parameter to the method `ApplyEntitiesConfig_Task`. Similarly, use the method's WSDL name, `ApplyEntitiesConfig_Task`, rather than the source name, `applyEntitiesConfiguration`.

Retrieving the Service Instance with the JSON Protocol

ServiceInstance is a singleton managed object that acts as a gateway to the resources available on the server. You can locate other important singletons such as SessionManager and SearchIndex from the properties of the ServiceInstance. ServiceInstance is accessible without authentication, so you can retrieve a reference to the SessionManager which you can use to authenticate a session.

To retrieve the properties of the ServiceInstance managed object, do the following.

Prerequisites

You need the IP address or fully qualified domain name of a vCenter Server to which you will send the request.

Procedure

- 1 Synthesize a managed object reference to the ServiceInstance managed object. This managed object reference always has a fixed value of `{type: "ServiceInstance", value: "ServiceInstance"}` because it is a prerequisite for access to other managed object references.

```
motype = 'ServiceInstance'
moid = 'ServiceInstance'
moref = motype + '/' + moid
```

- 2 Specify the property to retrieve from the managed object.

```
property = 'content'
```

- 3 Assemble the URL from the pieces previously defined.

```
scheme = 'https://'
domain = 'vcenter.example.com'
endpoint = '/sdk/vim25'
version = '8.0.1.0'
url = scheme + domain + endpoint + '/' + version + '/' + moref + '/' + property
print(url)
```

```
href="https://vcenter.example.com/sdk/vim25/8.0.1.0/ServiceInstance/ServiceInstance/
content"
```

- 4 Send the URL in an HTTP GET request.

```
from urllib.request import urlopen, Request
import ssl
# Skip certificate verification on test systems only:
unverified_context = ssl._create_unverified_context()
```

```
request = Request( url, headers=headers or {} )
with urlopen( request, context=unverified_context ) as response :
    response_headers = response.headers
    response_body = response.read()
```

5 Display the result of the query.

```
from pprint import pprint
json_body = json.loads(response_body)
pprint(json_body)
```

Results

The program displays a number of managed object references for singletons that you can use for additional requests. The following excerpt includes several important managed objects that you will use for basic operations, including authentication.

Example: Excerpt from Service Content

```
...
'propertyCollector': {'_typeName': 'ManagedObjectReference',
                      'type': 'PropertyCollector',
                      'value': 'propertyCollector'},
'rootFolder': {'_typeName': 'ManagedObjectReference',
               'type': 'Folder',
               'value': 'group-dl'},
'scheduledTaskManager': {'_typeName': 'ManagedObjectReference',
                          'type': 'ScheduledTaskManager',
                          'value': 'ScheduledTaskManager'},
'searchIndex': {'_typeName': 'ManagedObjectReference',
                'type': 'SearchIndex',
                'value': 'SearchIndex'},
'serviceManager': {'_typeName': 'ManagedObjectReference',
                   'type': 'ServiceManager',
                   'value': 'ServiceMgr'},
'sessionManager': {'_typeName': 'ManagedObjectReference',
                   'type': 'SessionManager',
                   'value': 'SessionManager'},
...
```

What to do next

Use the SessionManager URL in the Service Content to open an authenticated session with the vCenter Server instance.

Authenticating a JSON Client with the Session Manager

Most method calls must carry a session ID to authenticate with the server at the time of the call. The session ID is a temporary substitute for username and password, thereby limiting risk to the principal's credentials.

To acquire a session ID, you have these options:

- Invoke the `SessionManager.Login` method with username and password, exchanging the principal's credentials for a session ID token.
- Request a SAML token from the vCenter Single-Sign-On server, and use the token to invoke the `SessionManager.LoginByToken` method, receiving a session ID token in exchange.

After a successful authentication operation with vCenter Server, you extract the session ID token from the HTTP header of the response message. Include the same header in subsequent API request messages.

For example, the response headers might contain lines similar to the following:

```
content-type: application/json
date: Tue, 14 Aug 2007 13:30:00 GMT
vmware-api-session-id: 0123498765fabdec5263748596071829304100ab
x-envoy-upstream-service-time: 45
```

The value `0123498765fabdec5263748596071829304100ab` is the session ID token. Use it in the headers of your API requests, similar to the following:

```
content-type: application/json
vmware-api-session-id: 0123498765fabdec5263748596071829304100ab
```

Basic Authentication for the JSON Protocol

Basic authentication relies on username and password. These credentials are transported over HTTPS with TLS encryption, but for stricter security consider using SAML token authentication instead.

To authenticate with username and password, do the following steps.

Prerequisites

Basic authentication requires that you have a username and corresponding password that have privileges on the vCenter Server instance to which you connect. You also need the IP address or fully qualified domain name of the vCenter Server instance.

Procedure

- 1 Prepare a JSON serialization of your credentials. This will be the body of your request.

```
body_string = '{"userName": "Administrator@vsphere.local", "password": "betyoucantguess"}'
```

- 2 Get the `SessionManager` managed object reference from the `ServiceInstance` content. Format it for the resource portion of the service URL.

```
moref = si_content['sessionManager']
url_moref = '{}/{'}.format(moref['type'], moref['value'])
```

- 3 Assemble the complete URL for a request to invoke the Login method of the SessionManager.

```
scheme = 'https://'
domain = 'vcenter.example.com'
endpoint = '/sdk/vim25'
version = '8.0.1.0'
method = 'Login'
url = scheme + domain + endpoint + '/' + version + '/' + url_moref + '/' + method
print(url)
```

```
https://vcenter.example.com/sdk/vim25/8.0.1.0/SessionManager/SessionManager/Login
```

- 4 Set the request header 'Content-Type' = 'application/json'.

```
request_headers = {}
request_headers['Content-Type'] = 'application/json'
```

- 5 Send the URL and the body in an HTTP POST request.

```
from urllib.request import urlopen, Request
import ssl
# Skip certificate verification on test systems only:
unverified_context = ssl._create_unverified_context()
request = Request(url, headers=request_headers)
with urlopen(request, context=unverified_context) as response :
    response_headers = response.headers
    response_body = response.read()
```

- 6 Extract the session token from the authentication header in the response.

```
token = response_headers['vmware-api-session-id']
```

Results

The token you extracted is your current session ID and is required to access most managed object properties and methods with the API.

Example:

What to do next

Save the token. Add the token header to all subsequent requests before sending.

```
request_headers['vmware-api-session-id'] = token
```

SAML Token Authentication for a JSON Client

To authenticate by SAML token, you must use SOAP message protocol rather than the JSON protocol. After authentication, you can send authenticated requests over the JSON protocol.

SAML token authentication using the SOAP protocol is described in the *vCenter Single Sign-On Programming Guide*. Use the Single-Sign-On server to exchange the principal's credentials for a SAML token, then invoke the `LoginByToken()` method of the `SessionManager` to exchange the SAML token for a session ID token.

The `SessionManager` returns the session ID token in a cookie header. Use the value of the cookie as the value of `vmware-api-session-id` in request message headers, as described in [Authenticating a JSON Client with the Session Manager](#).

Mapping Methods for the JSON Protocol

All methods defined in the WSDL schema for the vSphere Web Services API can be invoked by using a REST-like URL with the POST verb. The method name appends to the URL and the method parameters appear as JSON in the body of the request.

Method names begin with an uppercase letter and match the method name in the vSphere Web Services API Reference. All methods attach to a managed object reference, as in the SOAP API.

For example, the `SessionManager` managed object has a method whose WSDL name is `AcquireLocalTicket`.

The URL format that you use to invoke a method is

```
https://vcenter_fqdn/sdk/vim25/8.0.1.0/mo_ref/method_name
```

For example, you invoke the `AcquireLocalTicket` method with a URL like this:

```
https://vcenter_fqdn/sdk/vim25/8.0.1.0/SessionManager/SessionManager/AcquireLocalTicket
```

When you POST a request to invoke a method on a managed object, the body of the request contains a single anonymous JSON object that specifies the parameter names and values. Unlike the SOAP request protocol, the JSON does not specify a reference to 'this' because the managed object reference is in the URL path.

For example, the method `Folder.CreateFolder()` requires a name for the new folder. The body for that request would look similar to this:

```
{"name": "spare_VM_folder"}
```

Accessing Managed Object Properties with the JSON Protocol

You retrieve a property of a managed object by appending the property name to the URL, after the managed object reference. Property names always begin with a lowercase letter, matching the name in the vSphere Web Services API Reference.

For example, you can retrieve the string property `HostProfile.validationState` with a URL similar to this:

```
https://vcenter_fqdn/sdk/vim25/8.0.1.0/hostprofile_mo_ref/validationState
```

Note You cannot modify properties directly. To modify properties of a managed object, you must invoke a method of the managed object or a related managed object that affects its state on the server.

Retrieving Nested Properties

You can retrieve a top-level property by naming it in the URL, but nested properties are not directly accessible. Many top-level properties are data objects or arrays, containing nested properties of interest. To access nested properties, first retrieve the top-level property, then deserialize the JSON in the response body. After deserialization, you can access the nested properties of the data object or array in the client.

For example, code this request:

```
GET https://vcenter_fqdn/sdk/vim25/8.0.1.0/vm_mo_ref/runtime
```

Then extract and display the nested property like this:

```
data_object = json.loads(response_body)
state = data_object['powerState']
print('VM powerState = ' + state)
```

The nested property displays.

```
VM powerState = poweredOff
```

Retrieving Managed Object Reference Properties

Managed objects often have properties to reference other managed objects. In these cases, you can use a GET request to obtain the managed object reference, but to access the properties or methods of the other managed object you must issue a new GET request, using the managed object reference in the request URL.

If you need to follow a chain of managed object references, you can avoid extra requests by using the `PropertyCollector` managed object. The purpose of the `PropertyCollector` is to delegate the search functions to the server, where they can be done more efficiently.

Using the Property Collector to Retrieve Object Properties

You can use the `PropertyCollector` to retrieve properties from one or more managed objects, whether or not they are related to each other, if you can specify the set of objects in your request. The `PropertyCollector` provides a flexible interface by which you can specify search criteria to execute in the server.

To specify a property retrieval, several approaches are available to you. A few of them are:

- Provide a managed object reference and a set of its properties that you want to retrieve.
- Provide a managed object reference as a starting point for a search that traverses inventory paths to find related managed objects and return their properties.
- Provide a reference to a previous search set as a starting point for a traversal search, often directed to the managed objects contained in the set.

This example shows a hybrid approach where you first locate a `Datacenter` managed object by using the `SearchIndex()` method. Then you use that managed object as a starting point to traverse to `VirtualMachine` objects in the data center's virtual machine folder. If there are subfolders within the virtual machine folder, the search specifications traverse the subfolders recursively.

The goal of this example is to report whether the virtual machines are powered on or powered off. That information is in the `VirtualMachine.runtime.powerState` property.

The `powerState` property is a nested property of the virtual machine, so we need to locate the `VirtualMachine` managed objects and specify a property within the `runtime` data object belonging to each `VirtualMachine`. The JSON protocol has the ability to retrieve the entire runtime data object as a single property, but the `PropertyCollector` has the power to extract the nested property in a single query.

Use the following procedure to build a server-side specification that will retrieve the power states of the virtual machines in the data center's virtual machine folder.

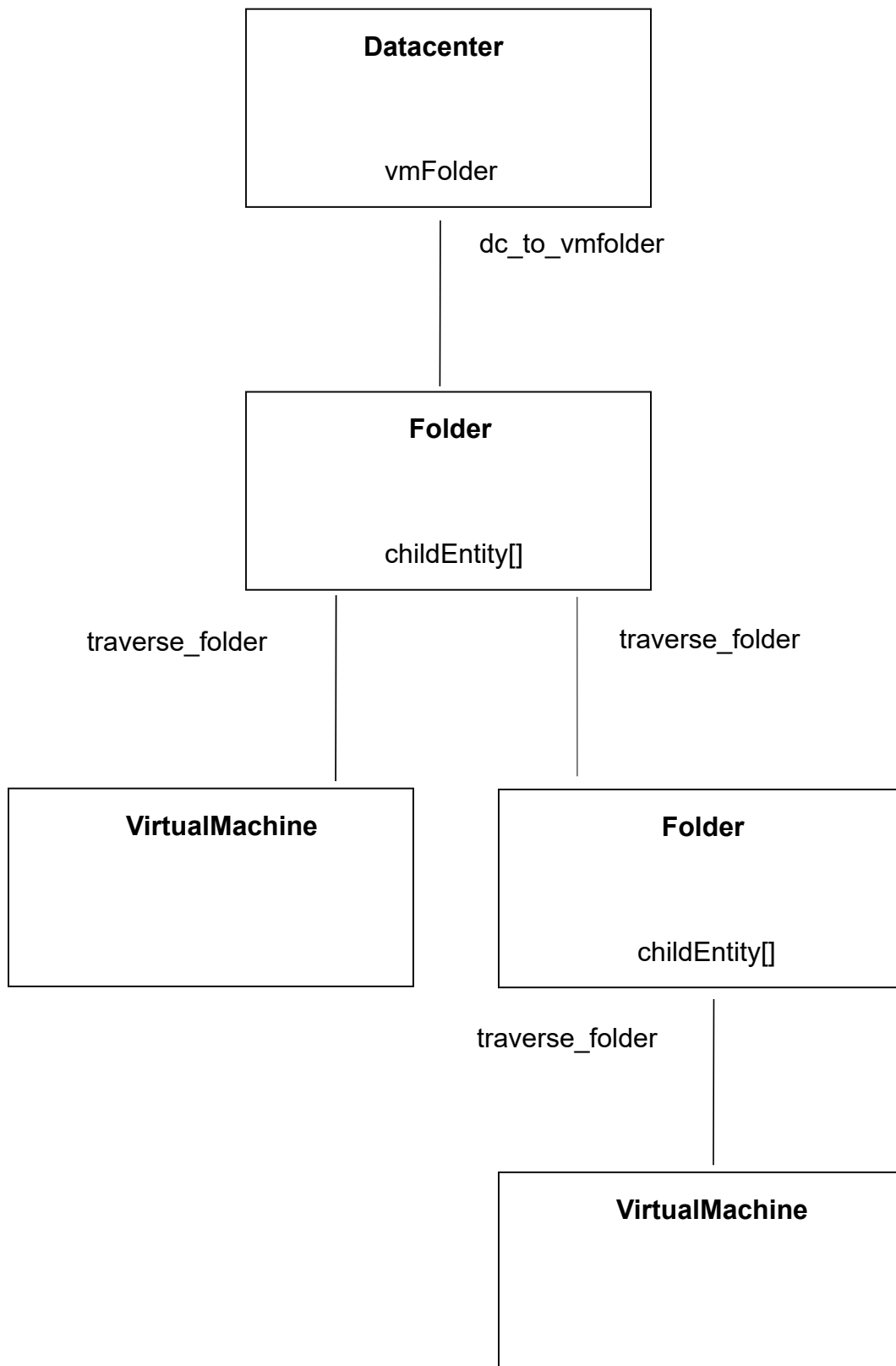
Note For simplicity, this procedure ignores VApps.

- 1 Build a `PropertySpec` data object to identify the managed object type (`VirtualMachine`) and the nested property name (`runtime.powerState`).

```
propSet = [ {'_typeName': 'PropertySpec',
            'type': 'VirtualMachine', 'pathSet': ['runtime.powerState'], 'all': False}]
```

The `all` property, if `True`, overrides the `pathSet` property and collects all properties of the managed object. This behavior is useful if you want to maintain a client-side copy in synch with the server-side managed object.

- 2 (optional) If it helps to clarify the inventory paths to the virtual machines, sketch `VirtualMachine` objects and their parent managed objects, all the way up the chain to the starting `Datacenter` managed object.



For each path between managed objects, note the name of the property in the parent object that links to the child object. You must create a `TraversalSpec` (or a `SelectionSpec` reference to a `TraversalSpec`) to follow each of those links to the next managed object. The `VirtualMachine` managed objects whose properties you want to retrieve do not need `TraversalSpec` objects.

- 3 Working backwards from `VirtualMachine` objects to `Datacenter` object, create a named `TraversalSpec` data object for each path segment.

```
traverse_folder = {'_typeName': 'TraversalSpec'}
traverse_folder['name'] = 'traverse_folder'
traverse_folder['type'] = 'Folder'
traverse_folder['path'] = 'childEntity'
traverse_folder['skip'] = False

dc_to_vmfolder = {'_typeName': 'TraversalSpec'}
dc_to_vmfolder['name'] = 'dc_to_vmfolder'
dc_to_vmfolder['type'] = 'Datacenter'
dc_to_vmfolder['path'] = 'vmFolder'
dc_to_vmfolder['skip'] = True
```

You do not need a `TraversalSpec` for the `VirtualMachine` object, because it is a leaf object in the traversal tree. You do need a `TraversalSpec` for the `Datacenter` object and the `Folder` object, so the `PropertyCollector` will know how to traverse down the chain to the child objects. The `skip` property is an optimization. It tells the `PropertyCollector` that the next managed object after the traversal can be traversed without checking the `PropertySpec`.

Note The `TraversalSpec.selectSet[]` property is not yet set. You will use it to link the specs together.

- 4 If your data center might have nested folders inside its VM folder, create a `SelectionSpec` data object that specifies the name of the `TraversalSpec` for `Folder` objects, so the `PropertyCollector` can follow the `childEntity` path recursively for any nested folders it finds.

```
folder_recursion = {'_typeName': 'SelectionSpec', 'name': 'traverse_folder'}
```

- 5 Link the `TraversalSpec` objects together by using the `selectSet[]` property to build a managed object traversal chain.

```
traverse_folder['selectSet'] = [folder_recursion]
dc_to_vmfolder['selectSet'] = [traverse_folder]
```

- 6 Create an `ObjectSpec` data object to specify starting from your `Datacenter` managed object, and to direct the `PropertyCollector` to the `TraversalSpec` that leads to the top-level VM folder.

```
objectSet = [{'_typeName': 'ObjectSpec',
              'obj': dc_moref, 'skip': False,
              'selectSet': [dc_to_vmfolder] }]
```

The `skip` property in the `ObjectSpec` is different from the `skip` property in a `TraversalSpec`. In the `ObjectSpec`, it tells the `PropertyCollector` that the starting object is not one of the managed objects it is looking for.

- Combine the `PropertySpec` and the `ObjectSpec` into a `PropertyFilterSpec` that will be a parameter to the `PropertyCollector.RetrievePropertiesEx` method.

```
specSet = [{'_typeName': 'PropertyFilterSpec',
            'propSet': propSet, 'objectSet': objectSet}]
```

- Specify the `RetrievePropertiesEx` method parameters in the request body.

```
import json
param_dict = {'options': {'_typeName': 'RetrieveOptions',
                          'maxObjects': 1000}, 'specSet': specSet}
request_body = json.dumps(param_dict).encode('utf-8')
```

- Send the request, using the POST verb.

```
from urllib.request import urlopen, Request
import ssl
moref = si_content['propertyCollector']
method = 'RetrievePropertiesEx'
url = make_url(domain, mo=moref, m_p=method)
request_headers = {}
request_headers['vmware-api-session-id'] = token
request_headers['Content-Type'] = 'application/json'
headers['Content-Type'] = 'application/json'
# Skip certificate verification on test systems only:
unverified_context = ssl._create_unverified_context()
request = Request(url, headers, data=body)
with urlopen(request, context=unverified_context) as response :
    response_headers = response.headers
    response_body = response.read()
    json_body = json.loads(response_body)
    print(json_body)
```

The results are returned in a `RetrieveResult` data object, which encloses an array of `ObjectContent` data objects. Each `ObjectContent` contains a managed object reference and a corresponding array of named properties found by the `PropertyCollector` that match your `PropertySpec`. For this example, the managed object references identify `VirtualMachine` objects, and the property name-value pairs show the `powerState` of each `VirtualMachine`.

Using Polymorphic Types with the JSON Protocol

The JSON protocol deals with managed objects whose properties include WSDL-defined data objects, such as configuration objects. Method parameters can also be data objects. Some of these data objects are polymorphic, in the sense that they can inherit from other data objects and can be referenced as the parent type.

The JSON protocol distinguishes the assigned type by adding a `_typename` property to the JSON serialization. For example, a `TaskInfo` data object contains a `result` property that has the XSD type `xsd:anyType`. The actual type of the assigned value is specified with a `_typename` property in `result`.

For example, the following JSON shows how a result data object is returned in the server response and the assigned type is identified by the `_typeName` property.

```
"result": {
  "_typeName": "ManagedObjectReference",
  "_value": {
    "value": "domain-s16",
    "type": "ComputeResource"
  }
}...
```

When you use a polymorphic type in a request, you must add a `_typename` property to identify the assigned type for the server. For example, if you create a `PropertyFilter` for a request to the `PropertyCollector`, you might include a `TraversalSpec` object, which inherits from the `SelectSet` object. Add the `_typename` property to distinguish a `TraversalSpec` value from a `SelectSet` value:

```
"traverse_vm_folder": {
  "_typeName": "TraversalSpec",
  "type": "Folder",
  "path": "childEntity",
  "skip": false
}
```

Many properties are defined in the WSDL as inheriting from `DynamicData`. For that reason, you will use `_typeName` frequently. A best practice is to add the `_typeName` property to all objects you create for the JSON API.

Python Client Example for the JSON Protocol

This brief example illustrates several basic operations you need to implement a client that uses the JSON protocol.

```
import json
from pprint import pprint

domain = 'vcenter.example.com'
credentials = {'userName': 'Administrator@vsphere.local', 'password': 'betyoucantguess'}
dc_name = 'DC-1'

def make_url(vc, ver='8.0.1.0', mo='ServiceInstance/ServiceInstance', m_p='content'):
    if isinstance(mo, dict):
        mo = '{}/{}'.format(mo['type'], mo['value'])
    url = 'https://{}/sdk/vim25/{}/{}/{}'.format(vc, ver, mo, m_p)
```

```

    return url

def send_request(url, headers=None, body=None):
    from urllib.request import urlopen, Request
    import ssl
    if headers is None: headers = {}
    headers['Content-Type'] = 'application/json'
    # Skip certificate verification on test systems only:
    unverified_context = ssl._create_unverified_context()
    request = Request(url, headers=headers or {}, data=body)
    with urlopen( request, context=unverified_context ) as response :
        response_headers = response.headers
        response_body = response.read()
    return( response_headers, response_body )

def get_service_content(vc):
    url = make_url(vc)
    (response_headers, response_body) = send_request(url)
    json_body = json.loads( response_body )
    return( json_body )

def basic_auth(domain, credentials):
    si_content = get_service_content(domain)
    moref = si_content['sessionManager']
    method = 'Login'
    url = make_url(domain, mo=moref, m_p=method)
    request_headers = {}
    request_headers['Content-Type'] = 'application/json'
    request_body = json.dumps(credentials).encode('utf-8')
    (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
    print('Response:')
    print(response_headers)
    print(json.loads(response_body))
    token = response_headers['vmware-api-session-id']
    return(token)

def logout(domain, session_moref) :
    request_body = {}
    method = 'Logout'
    url = make_url(domain, mo=session_moref, m_p=method)
    request_headers = {}
    request_headers['vmware-api-session-id'] = token
    request_headers['Content-Type'] = 'application/json'
    (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
    return(response_headers)

def get_datacenter_moref(domain, dc_name, token):
    si_content = get_service_content(domain)
    moref = si_content['searchIndex']
    method = 'FindByInventoryPath'
    url = make_url(domain, mo=moref, m_p=method)
    body_dict = {'inventoryPath': dc_name}
    request_body = json.dumps(body_dict).encode('utf-8')

```

```

request_headers = {}
request_headers['vmware-api-session-id'] = token
request_headers['Content-Type'] = 'application/json'
(response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
json_body = json.loads(response_body)
return(json_body)

def get_vm_power_states(domain, dc_moref, token):
    # MO type & properties to retrieve.
    propSet = [ {'_typeName': 'PropertySpec',
                  'type': 'VirtualMachine', 'pathSet': ['runtime.powerState'], 'all': False}]
    # Work backwards to define the steps in the traversal path.
    # traversalspec = type to traverse, path out, skip next?, (selectSet).
    # Traverse any nested folders inside vmFolder:
    traverse_folder = {'_typeName': 'TraversalSpec'}
    traverse_folder['name'] = 'traverse_folder'
    traverse_folder['type'] = 'Folder'
    traverse_folder['path'] = 'childEntity'
    traverse_folder['skip'] = False
    traverse_folder['selectSet'] = [{'_typeName': 'SelectionSpec',
                                     'name': 'traverse_folder'}]
    # Traverse Datacenter to vmFolder:
    dc_to_vmfolder = {'_typeName': 'TraversalSpec'}
    dc_to_vmfolder['name'] = 'dc_to_vmfolder'
    dc_to_vmfolder['type'] = 'Datacenter'
    dc_to_vmfolder['path'] = 'vmFolder'
    dc_to_vmfolder['skip'] = True
    dc_to_vmfolder['selectSet'] = [traverse_folder]
    # objectSet = Starting MO + traversal steps to retrieval objects.
    objectSet = [{'_typeName': 'ObjectSpec',
                  'obj': dc_moref, 'skip': False,
                  'selectSet': [dc_to_vmfolder]}]
    # specSet = total filter specification.
    specSet = [{'_typeName': 'PropertyFilterSpec',
                  'propSet': propSet, 'objectSet': objectSet}]
    # Params to RetrievePropertiesEx():
    param_dict = {'options': {'_typeName': 'RetrieveOptions',
                              'maxObjects': 1000}, 'specSet': specSet}
    # Build request message.
    request_body = json.dumps(param_dict).encode('utf-8')
    moref = si_content['propertyCollector']
    method = 'RetrievePropertiesEx'
    url = make_url(domain, mo=moref, m_p=method)
    request_headers = {}
    request_headers['vmware-api-session-id'] = token
    request_headers['Content-Type'] = 'application/json'
    (response_headers, response_body) = send_request(url, headers=request_headers,
body=request_body)
    json_body = json.loads(response_body)
    return(json_body)

# MAIN

si_content = get_service_content(domain)

```

```
pprint(si_content); print()

token = basic_auth(domain, credentials)
print('Session token: {}'.format(token)); print()

dc_moref = get_datacenter_moref(domain, dc_name, token)
print('{} MOref = {}'.format(dc_name, dc_moref)); print()

response_body = get_vm_power_states(domain, dc_moref, token)
print('Response body:'); pprint(response_body)

moref = si_content['sessionManager']
headers = logout(domain, moref)
print('Logging out:')
print(headers)
```

Datacenter Inventory

5

The vSphere inventory is a representation of the vSphere datacenter and the objects in the datacenter. Knowing how the objects in the datacenter relate to each other helps you traverse the inventory hierarchy and access the objects you want to manipulate.

This chapter includes the following topics:

- [Inventory Overview for the Web Services SDK](#)
- [Inventory Hierarchies and ServiceInstance](#)
- [Accessing Inventory Objects](#)
- [Creating Inventory Objects](#)
- [Privileges Required for Inventory Management](#)
- [Managed and Standalone ESXi Hosts](#)

Inventory Overview for the Web Services SDK

The vSphere inventory contains the following types of objects:

- Systems in the datacenter: `Host`, `VirtualMachine`, and `VirtualApp`.
- Support components: `ComputeResource`, `Datastore`, `Network`, and virtual devices.
- Organizational components: `Folder` and `Datacenter`

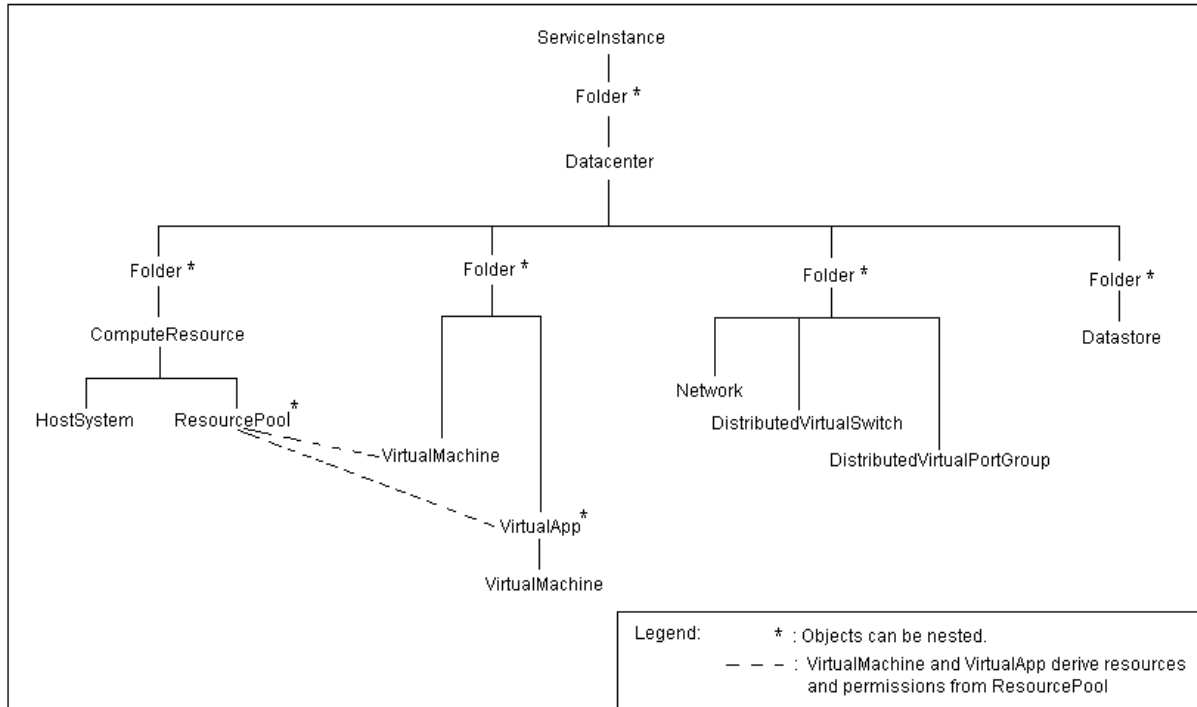
When you manage the virtual infrastructure, you access objects and their properties and methods based on their location in the inventory. Understanding the inventory structure is therefore critical for any programming task. You always start with the `ServiceInstance` associated with a session, which is the root object of the inventory, and traverse the inventory hierarchy from there. See [Chapter 6 Property Collector](#). How you access objects depends on whether your client application is connected to a vCenter Server or an ESXi host.

Inventory Hierarchies and ServiceInstance

When you start a session, vSphere creates a `ServiceInstance` with one root folder, one `Datacenter`, and four folders that hold the different types of inventory objects.

When you access a vCenter Server System, the hierarchy shown in the illustration below allows you to traverse the inventory.

Figure 5-1. vCenter Server Inventory Hierarchy



Caution If your ESXi hosts are managed by vCenter Server, you must always access your hosts through vCenter Server. The vCenter service keeps track of all synchronous and asynchronous operations, and will have the latest status and inventory information about each ESXi host. Therefore, connecting directly to a managed host may give you incorrect or incomplete data.

When you have ESXi hosts that are not managed by vCenter Server, your application can connect to each host directly.

Folders in the Hierarchy

If your installation includes a vCenter Server system, you can create additional datacenters under the root folder. For every **Datacenter** object, the server automatically creates the following **Folder** objects:

- A folder for **VirtualMachine**, template, and **VirtualApp** objects.
- A folder for a **ComputeResource** hierarchy.
- A folder for **Network**, **DistributedVirtualSwitch**, and **DistributedVirtualPortgroup** objects.
- A folder for **Datastore** objects.

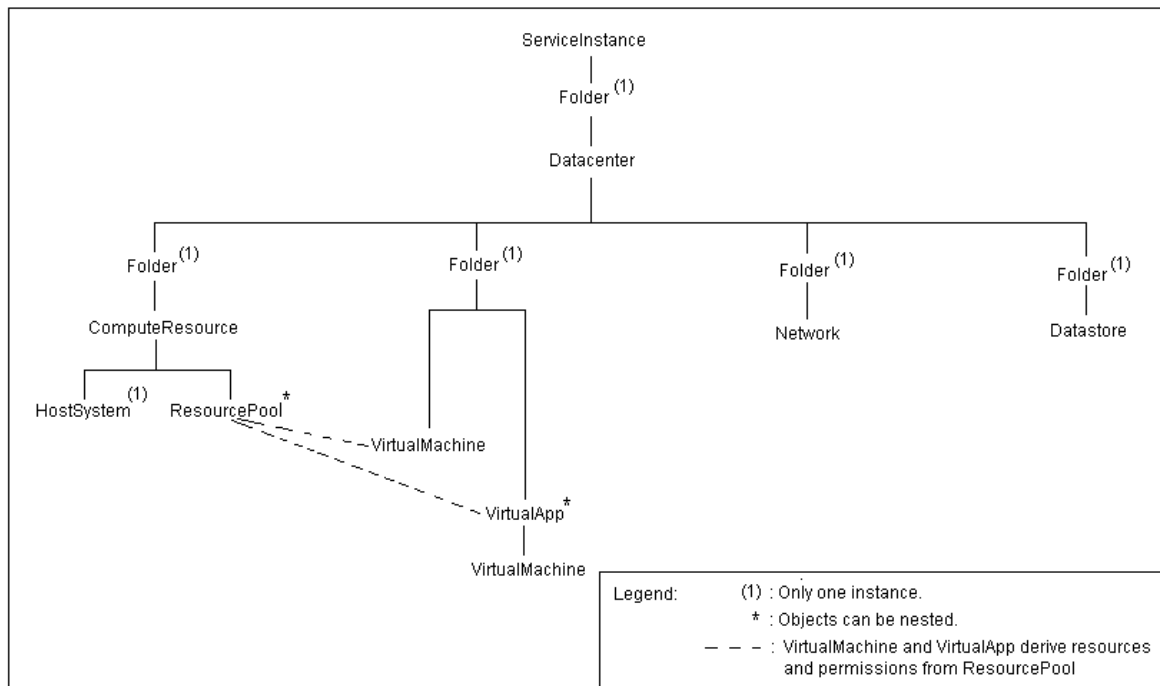
In a large deployment, the nested structure allows you to organize the objects in the datacenter into an easily manageable structure by using multiple folders and datacenters.

For a standalone ESXi system, only a single datacenter is supported, and the `Folder` managed entity does not support creating additional `Folder` objects or `Datacenter` objects.

ESXi Inventory Hierarchy

When you access an ESXi host directly, rather than accessing the host through a vCenter Server system, the hierarchy shown in the illustration below allows you to traverse the inventory.

Figure 5-2. ESXi Inventory Hierarchy



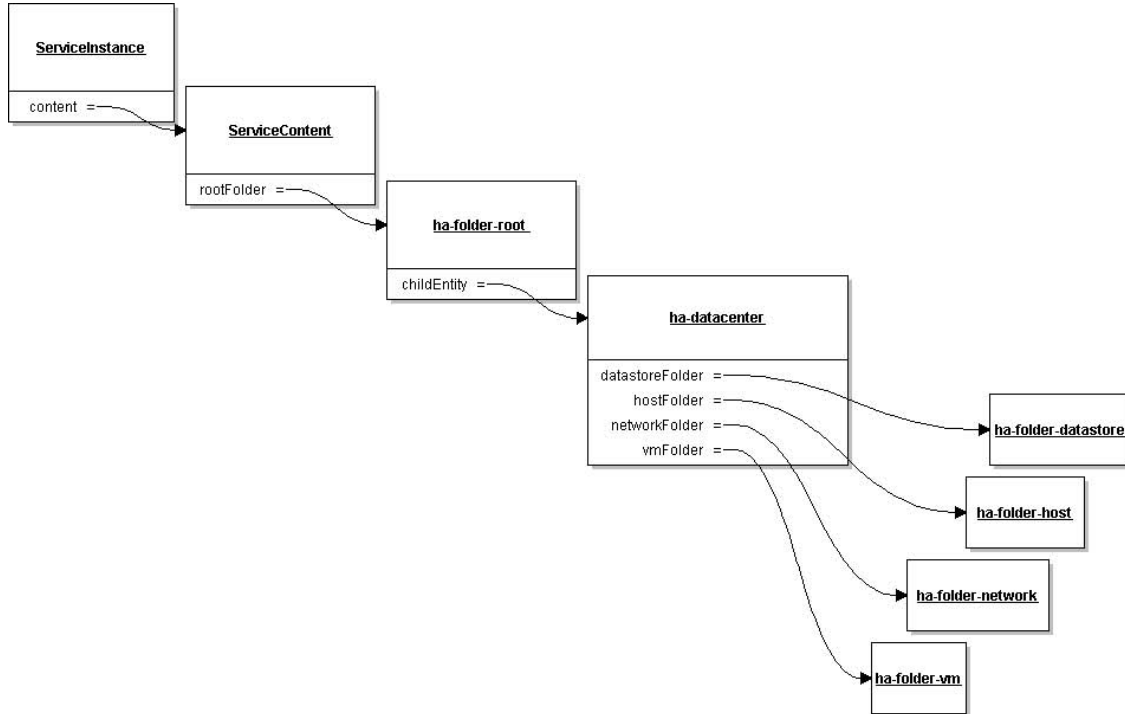
Accessing Inventory Objects

To retrieve information from an inventory object, you start with `ServiceInstance`, the root object of the inventory. You access an object using a `TraversalSpec` in conjunction with a property collector, using the properties that identify an object's position in the hierarchy.

- Every managed entity has a `parent` property that identifies its relative position in the inventory hierarchy.
- The `Folder` managed object has a `childEntity` property that identifies objects in a folder instance.

Figure 5-3. [Instance Diagram of Root Folders in an Inventory](#) shows the `childEntity` and `folder` properties that define the default objects in the inventory of a standalone ESXi system. The inventory begins with the `ServiceContent.rootFolder` property. The `rootFolder` has a `childEntity` that consists of a managed object reference to a `Datacenter` managed object.

Figure 5-3. Instance Diagram of Root Folders in an Inventory



Creating Inventory Objects

The `Folder` managed entity provides methods for creating instances of the following managed entities.

- `Datacenter`
- `DistributedVirtualSwitch`
- `VirtualMachine`
- `Cluster`
- `Folder`

When you create these objects, they appear in the folder you invoked the creation method from.

While some managed entities are created through a method on the `Folder` managed entity, other managed entities are instantiated directly. For example, the `HostDatastoreSystem` has methods for creating datastores such as `CreateNasDatastore` and `CreateVmfsDatastore`.

Important When you create an inventory object, you must stay within the bounds of the host's capabilities, accessible through the `HostSystem.capability` property, which is a `HostCapability` data object. For example, a `HostCapability` object might have the `maxSupportedVMs` property specified.

Privileges Required for Inventory Management

Navigating the inventory requires a user account that can connect to the server and obtain a valid session. The user identity associated with the session is called a principal. When a client application attempts to access an object in the inventory, the server checks the permission object or objects and compares the permissions with the principal's privileges.

For example, creating a virtual machine requires that the principal associated with the session have the following privileges:

- The `VirtualMachine.Inventory.Create` privilege on the folder in which to create the virtual machine.
- The `Resource.AssignVMToPool` privilege on the resource pool from which the virtual machine obtains its allocation of CPU and memory resources.

Reading the `perfCounter` property of the `PerformanceManager` managed object requires the `System.View` privilege on the root folder.

Important Some privileges are specific to objects on vCenter Server or specific to ESXi. For example, the `Alarm.Create` privilege associated with `AlarmManager` is available only through vCenter Server systems.

See [Chapter 7 Authentication and Authorization](#) for more information on authentication, authorization, roles, and user identity.

Privileges

A privilege is a system-defined requirement associated with a VMware vSphere managed object. Privileges are static and do not change for a version of a product. Privileges for vSphere components are defined as follows:

```
<group>[.<group>].privilege
```

For example:

```
Datacenter.Create
Host.Config.Connection
Host.Config.Snmp
```

Permissions

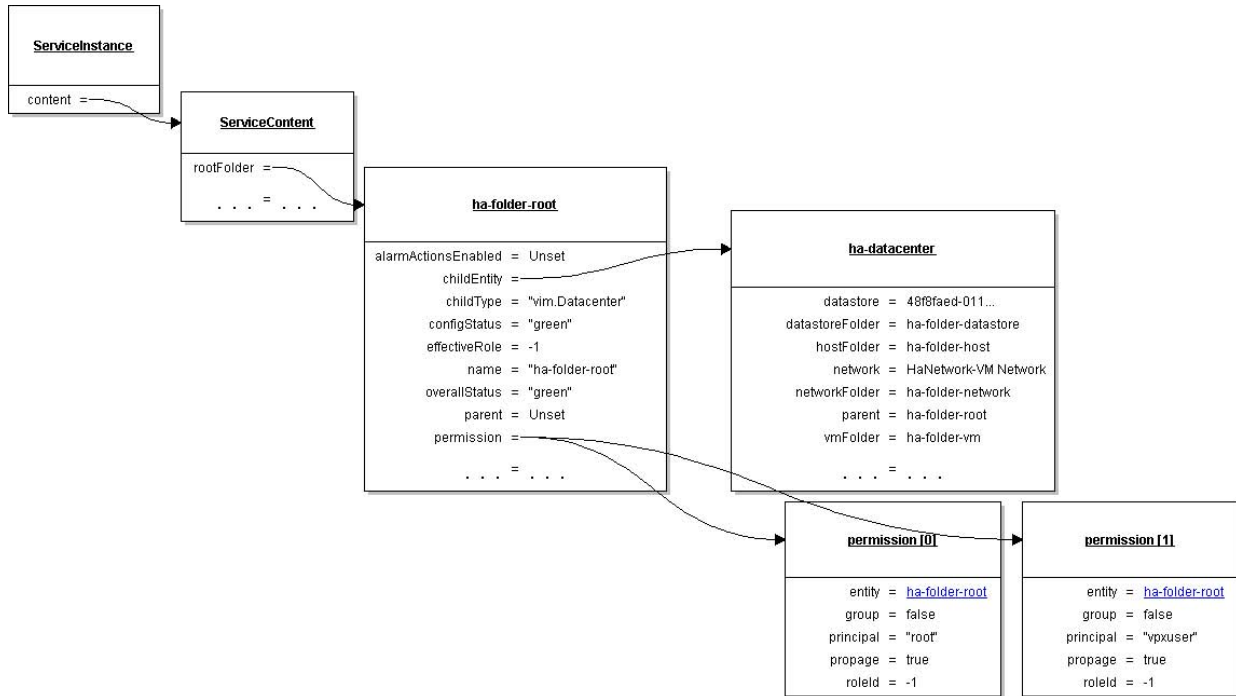
Permissions are the associations of roles with privileges on a specified managed entity. You use permissions to specify which users can access which managed entity.

A child entity inherits the permissions of its parent if the parent's `propagate` property is set to true. A permission that is set directly on a child overrides the permission in the parent. To grant permission to all child entities of a `Datacenter` object, assign permissions to the `Datacenter` object and set the `Permission` object's `propagate` property to true.

Figure 5-4. Inventory and Permissions shows that users `root` and `vpuser` both have permissions on the `rootFolder` of the inventory. The `vpuser` is the account created on a host by the vCenter Server system when that host is added to the vCenter Server system. The vCenter Server system needs access to the inventory objects of the host systems that it manages, so the `vpuser` account is granted privileges to the `rootFolder` of each host.

Important See [Chapter 7 Authentication and Authorization](#) for a detailed discussion of privileges, permissions, and user management.

Figure 5-4. Inventory and Permissions



Managed and Standalone ESXi Hosts

You can run ESXi as a managed or standalone ESXi host.

- Standalone ESXi hosts are standalone hosts with limited capabilities. The inventory of a standalone host can support multiple virtual machines and multiple resource pools, but it contains a single default datacenter and a single root folder. The default datacenter and root folder are not visible in the vSphere Client, but they exist in the inventory of a standalone host and they are visible in the MOB.
- Managed ESXi hosts have been added to the vCenter Server inventory. Available features depend on the licenses available for that host. For example, you can configure two or more hosts for VMware DRS resource management or VMware HA failover protection.

The following table summarizes the differences between the number of objects that an inventory can contain. See also [Figure 5-2. ESXi Inventory Hierarchy](#) and [Figure 5-1. vCenter Server Inventory Hierarchy](#).

ManagedEntity Subtype	ESX/ESXi Inventory	vCenter Server Inventory
ClusterComputeResource	None.	Multiple instances supported.
ComputeResource	Exactly one only.	Multiple instances supported.
Datacenter	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
Datastore	Multiple instances supported.	Multiple instances supported.
DistributedVirtualSwitch	Multiple instances supported.	Multiple instances supported.
Folder	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
HostSystem	Exactly one only.	Multiple instances supported.
Network	Multiple instances supported.	Multiple instances supported.
ResourcePool	Multiple instances supported.	Multiple instances supported.
VirtualApp	None.	Multiple instances supported.
VirtualMachine	Multiple instances supported.	Multiple instances supported.

Property Collector

6

vSphere servers provide the `PropertyCollector` service for accessing data and monitoring changes. Use the `PropertyCollector` to obtain references to managed objects, to obtain values of managed object properties, and to monitor and retrieve modified property values.

This chapter includes the following topics:

- [Introduction to the PropertyCollector](#)
- [vSphere Data Objects for Property Collection](#)
- [vSphere Methods for Property Collection](#)
- [Using the PropertyCollector with RetrievePropertiesEx](#)
- [Retrieve a Property from a Specified Managed Object](#)
- [Retrieve Properties from a Set of Managed Objects with a View](#)
- [Simple Property Collector Program in Java for Web Services SDK](#)
- [Inventory Traversal](#)
- [Client Data Synchronization \(WaitForUpdatesEx\)](#)
- [Server Data Transmission](#)
- [PropertyCollector Performance](#)
- [SearchIndex](#)

Introduction to the PropertyCollector

The `PropertyCollector` service interface provides a way to monitor and retrieve information about managed objects, such as whether a virtual machine is powered on or whether a host in a cluster is offline.

The `PropertyCollector` uses one or more filters to determine the scope of collection and it has methods to retrieve data. A filter uses a set of data objects that specify the following information:

- Starting point for inventory traversal during the collection operation.
- Inventory traversal path.
- Objects and properties from which data will be collected.

A vSphere server creates a default `PropertyCollector` for every session, and allows you to create multiple, additional `PropertyCollector` objects. Create additional `PropertyCollector` objects, using one per thread, to perform concurrent collection operations.

Data Retrieval

There are two ways to retrieve data:

- Property retrieval as a single operation uses the `RetrievePropertiesEx` and `ContinueRetrievePropertiesEx` methods. These methods perform a single collection operation.
- Incremental property retrieval, also referred to as property monitoring, uses the `WaitForUpdatesEx` method. The initial call to this method retrieves a baseline set of managed object property values. Subsequent calls retrieve changes in property values since the last retrieval. Use `WaitForUpdatesEx` to monitor changes to the inventory or any managed object properties.

Note The `PropertyCollector` does not guarantee the order of data that it returns in response to a request for data.

Inventory Traversal and Property Selection

To use the `PropertyCollector`, an important preparatory step is to build a `PropertyFilterSpec` to guide the collector's actions on the server. You pass your `PropertyFilterSpec` either to the `RetrievePropertiesEx` method or to the `CreateFilter` method. The latter establishes a persistent filter on the server side for use with the `WaitForUpdatesEx` method.

A `PropertyFilterSpec` contains:

- The name of at least one managed object class from which you want to collect properties. (`PropertySpec.type`)
- For each managed object class specified, at least one name of a property to return. (`PropertySpec.pathSet`).
- If needed, one or more paths through the inventory hierarchy that the `PropertyCollector` can traverse to reach the managed objects from which you want to collect properties. (`ObjectSpec.selectSet`)
- For each path, a starting managed object reference from which to traverse that path. (`ObjectSpec.obj`)

If you do not have a starting managed object reference from a prior API call, you can start with the root folder at the base of the inventory hierarchy. The `rootFolder` reference is available in the `ServiceContent` data object. For information about getting the `ServiceContent`, see [Overview of a Java Sample Application for the Web Services SDK](#).

Gathering Data with a ViewManager Object

You can use a vSphere view object in a `PropertyCollector` filter to simplify access to a subset of inventory objects. For example, you could use a `ContainerView` to collect information about all `VirtualMachine` objects in a datacenter, or a `ListView` to track information about `Task` objects while an operation is in progress.

The `ViewManager`'s `ListView` method allows you to customize your view with an input object list, the `ContainerView` method lets you view all objects in a folder, datacenter, resource pool, or other container of managed objects, and the `InventoryView` method lets you monitor the entire inventory.

Note A best practice is to use the smallest view you can create. This will be the most efficient way to retrieve data with the `PropertyCollector`.

The `ViewManager` has the following property:

- `viewList` – An array of view references. Each array entry is a managed object reference to a view created by this View Manager.

The view acts as a virtual parent object to the set, which provides the following benefits:

- The view can collect objects from different parts of the inventory and make them available under the same parent.
- All objects in the set are accessible by doing a single traversal step, rather than a chain of several steps starting at the root folder.

A best practice when using views is to call the `DestroyView()` method when a view is no longer needed. This practice frees memory on the server.

See [Retrieve Properties from a Set of Managed Objects with a View](#) for an example that uses the `ContainerView` method to access Inventory data.

For more information about the vSphere inventory, browse the managed object definitions in the *vSphere Web Services API Reference*.

vSphere Data Objects for Property Collection

The following table provides an overview of the `PropertyCollector` data objects. For more detailed descriptions, see the *vSphere API Reference*.

Data Object	Description
<code>PropertyFilterSpec</code>	Provides access to object and property selection data. A <code>PropertyFilterSpec</code> must have at least one <code>ObjectSpec</code> and at least one <code>PropertySpec</code> .
<code>ObjectSpec</code>	Identifies the starting object for property collection, which can be either an inventory object or a view you created. An <code>ObjectSpec</code> also identifies the traversal specs used by the property collector.

Data Object	Description
<code>TraversalSpec</code>	Identifies the objects and paths that can be traversed to find the objects and properties identified by the <code>PropertySpec</code> .
<code>SelectionSpec</code>	Lists the <code>TraversalSpec.objects</code> available at each step of an inventory traversal. Supports recursive traversal by name.
<code>PropertySpec</code>	Identifies object types and their properties for collection.
View objects	Use a <code>view</code> to form a set of starting objects for traversal and collection.

vSphere Methods for Property Collection

The `PropertyCollector` supports the following approaches to obtaining objects and properties from the server:

- If your client application does not keep a synchronized representation of server state, use the `RetrievePropertiesEx` method. `RetrievePropertiesEx` instantiates a filter, collects the specified objects and properties, and returns the data to your client application as an `ObjectContent` data object. The server does not add the filter to the `PropertyCollector.filter` array. The server destroys the filter after returning the results to your client.
- If your application maintains a synchronized representation of server state, use the `CreateFilter` and `WaitForUpdatesEx` methods. `WaitForUpdatesEx` returns descriptions of property changes, organized by the filter that identified the properties.

In either case, you create a `PropertyFilterSpec` data object to specify the objects and properties you want to retrieve from the server.

The following table shows the `PropertyCollector` methods organized by the context in which you use them. For more information about these methods, see the *vSphere API Reference*.

Method Context	Method	Description
Monitor properties using different filters	<code>CreatePropertyCollector</code>	Creates a new <code>PropertyCollector</code> object to monitor properties using different filters. The vSphere server handles requests for a <code>PropertyCollector</code> instance independently of any other instances of the <code>PropertyCollector</code> on the server.
	<code>DestroyPropertyCollector</code>	Destroys an instance of a <code>PropertyCollector</code> that was created by a call to <code>CreatePropertyCollector</code> from your client application.
Single collection operation	<code>RetrievePropertiesEx</code>	Retrieves property data for the managed objects specified by a single-use <code>PropertyFilterSpec</code> .
	<code>ContinueRetrievePropertiesEx</code>	Retrieves additional property data for an operation started by <code>RetrievePropertiesEx</code> .

Method Context	Method	Description
	<code>CancelRetrievePropertiesEx</code>	Cancels a <code>RetrievePropertiesEx</code> or <code>ContinueRetrievePropertiesEx</code> operation.
Incremental collection or monitoring operation	<code>WaitForUpdatesEx</code>	Retrieves changes to property data since the last <code>WaitForUpdatesEx</code> cycle. <code>WaitForUpdatesEx</code> blocks until one or more changes are detected or until the request times out.
	<code>CancelWaitForUpdatesEx</code>	Cancels a <code>WaitForUpdatesEx</code> operation.
General	<code>CreateFilter</code>	Creates a new instance of a <code>PropertyFilter</code> managed object.

Using the PropertyCollector with RetrievePropertiesEx

The following procedures show how to retrieve properties of selected managed objects by using the `RetrievePropertiesEx` method. Two use cases are shown.

A simple use case is to retrieve properties from a managed object for which a reference has been supplied. You need only create a `PropertyFilterSpec` that contains a `PropertySpec` to identify the property to retrieve and an `ObjectSpec` to identify the managed object. Pass this `PropertyFilterSpec` to the `RetrievePropertiesEx` method and get the property value from the structured result.

A more complicated use case retrieves properties from a set of managed objects found in the inventory hierarchy. For this use case, you create a `View` managed object that provides a path to all the target managed objects within one traversal step. You need to create a `PropertyFilterSpec` that contains a `PropertySpec` to identify the property to retrieve and an `ObjectSpec` that identifies the `View` object and also contains a `TraversalSpec` to describe the path from the `View` to its children. Pass this `PropertyFilterSpec` to the `RetrievePropertiesEx` method and get the property value from the structured result.

Retrieve a Property from a Specified Managed Object

This procedure shows how to use the `PropertyCollector` to retrieve a property from a `VirtualMachine` object with the `RetrievePropertiesEx` method. The example retrieves a managed object reference to the parent object of a `VirtualMachine` object in the inventory hierarchy.

To retrieve a reference to the parent object of a specific virtual machine, you must:

- Prepare a `PropertySpec` to specify the `parent` property and the `parentVApp` property of the `VirtualMachine` class. A virtual machine can have only one parent, but the parent can be either a `Folder` object or a `VirtualApp` object. This example collects both properties, one of which will be null.

- Prepare an `ObjectSpec` to identify the starting object, which is the specified `VirtualMachine`. There is no need for a `TraversalSpec` because the property belongs to the starting object itself. However, the `ObjectSpec.skip` property is set to `false`; a value of `true` would cause the `PropertyCollector` not to collect properties from the starting object.
- Assemble a `PropertyFilterSpec` from the prepared data.
- Invoke the `RetrievePropertiesEx` method.

Prerequisites

For this task you need:

- A virtual machine managed object reference in a variable named `vmRef`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A `VimPort` binding provider referenced by the variable `methods`, which is attached to the vSphere server connection context.
- A `PropertyCollector` instance referenced by the variable `pCollector`.

Note This procedure shows only how to use the `PropertyCollector`. For a description of server connection and getting a reference to the `PropertyCollector`, see [Build a Simple vSphere Client Application for the Web Services SDK](#).

Procedure

- 1 Declare a function that accepts a virtual machine MOref and returns a MOref to its parent object in a key-value pair.

```
private static DynamicProperty getVMParentProperty(ManagedObjectReference vmRef)
throws Exception {
```

- 2 Specify the properties for retrieval (`VirtualMachine.parent` and `VirtualMachine.parentVApp`).

Because the `VirtualMachine` has two mutually exclusive parent properties, depending on the parent type, this code collects both properties.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("parent");
pSpec.getPathSet().add("parentVApp");
```

3 Create an `ObjectSpec` to define the property collection.

Use the `setObj` method to specify that the `vmRef` is the starting object for this property collection. Set the `skip` property to `false` to indicate that you want to collect properties from the starting object. Omit the `selectSet` property because the property collection does not need to traverse away from the starting object.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(false);
```

4 Create a `PropertyFilterSpec` and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

5 Create a list for the filters and add the property filter spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

6 Use the filter spec to retrieve the property collection from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector, fSpecList, ro);
```

7 Unwrap the parent property.

```
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                if (dp.getName().equals("parent") || dp.getName().equals("parentVApp")) {
                    return dp;
                }
            }
        }
    }
}
System.out.println("Parent not found.");
throw new Exception();
}
```

Results

The method returns a key-value pair, `name` and `val`, indicating the name of the parent property and a managed object reference to the parent object.

What to do next

```
DynamicProperty parent = getVMParentProperty( vmRef );
if (parent.getName().equals("parentVApp")) {
    System.out.format("VApp MObjref: %s", parent.getVal());
} else {
    System.out.format("Folder MObjref: %s", parent.getVal());
}
```

Retrieve Properties from a Set of Managed Objects with a View

This procedure shows how to use the `PropertyCollector` with a `ContainerView` to retrieve properties from `VirtualMachine` objects with the `RetrievePropertiesEx` method.

To collect the names of all virtual machines in the inventory, you must:

- Prepare a `ContainerView` that contains only the virtual machines.
- Prepare a `PropertySpec` to specify the name property of the `VirtualMachine` class.
- Prepare a `TraversalSpec` to specify the path from the view to its contents.
- Assemble a `PropertyFilterSpec` from the prepared data.
- Invoke the `RetrievePropertiesEx` method.

Note This procedure shows only how to use the `PropertyCollector`. For a description of server connection, see [Build a Simple vSphere Client Application for the Web Services SDK](#).

To collect the names of all virtual machines in the inventory, use the following steps.

Procedure

- 1 Get references to the `ViewManager` and the `PropertyCollector`.

In the example, `sContent` is the variable for the `ServiceContent` data object. `sContent` provides the methods to retrieve the managed object references to the vSphere services.

```
ManagedObjectReference viewMgrRef = sContent.getViewManager();
ManagedObjectReference propColl = sContent.getPropertyCollector();
```

- 2 Create a container view for virtual machines.

`methods` is the variable for the `VimPortType` object. `VimPortType` defines the Java methods that correspond to the vSphere API methods. The `createContainerView` parameters are as follows:

- `container`: Selection starts within the inventory root folder.
- `type`: Only `VirtualMachine` type managed objects are selected for the `ContainerView`, starting at the root folder.

- `recursive`: The value `true` for the last parameter extends the selection beyond the children of the root folder so that the `ViewManager` will follow child folder paths to add virtual machines to the view.

This container view provides references to all virtual machines in the inventory.

```
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");

ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
    sContent.getRootFolder(),
    vmList,
    true );
```

3 Create an object specification to define the starting point for inventory navigation.

The `ObjectSpec.obj` property identifies the starting object (the container view). This example collects only virtual machine data, so the `skip` property is set to `true` to ignore the container view itself during collection.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);
```

4 Create a traversal specification to identify the path for collection.

The `TraversalSpec` properties `type` and `path` determine path traversal. `TraversalSpec.type` identifies the type of managed object that this spec can traverse. `TraversalSpec.path` identifies a property that links to other managed objects. The `PropertyCollector` uses the `path` object to select additional objects.

This example uses a single `TraversalSpec` to walk the list of virtual machines that are available through the container view. The following code fragment specifies the `ContainerView` object for the `TraversalSpec.type` property and the `view` property in the `ContainerView` for the `TraversalSpec.path` property. The `skip` property is set to `false`, so the `PropertyCollector` will collect data from the `path` objects (the virtual machines in the container view).

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setPath("view");
tSpec.setSkip(false);
tSpec.setType("ContainerView");
```

5 Add the `TraversalSpec` to the `ObjectSpec.selectSet` array. The `TraversalSpec` tells the `PropertyCollector` how to traverse from the starting object to other managed objects.

```
oSpec.getSelectSet().add(tSpec);
```

6 Identify a managed object type and the properties to be retrieved from it.

The example program creates a `PropertySpec` data object to specify the properties to be collected. The `type` property is set to a managed object type present in the container view. The `pathSet` property identifies one or more properties in the `type` object.

This example specifies the `VirtualMachine.name` property.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");
```

7 Add the object and property specifications to the property filter specification.

A `PropertyFilterSpec` must have at least one `ObjectSpec` and one `PropertySpec`. The `PropertyFilterSpec` specifies what properties to collect, while the `ObjectSpec` tells the `PropertyCollector` how to find them.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

8 Create a list for the filters and add the `PropertyFilterSpec` to it.

This example needs only one filter to collect the names of virtual machines.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

9 Retrieve the data.

To invoke a single property collection operation, call the `RetrievePropertiesEx` method. The example application passes the populated `PropertyFilterSpec` and an empty `options` structure to the method. The default for the `RetrieveOptions.maxObjects` specifies no maximum for the number of objects that can be returned.

Note The `PropertyCollector` can impose a maximum in some circumstances. If the number of collected objects is greater than the maximum, the `PropertyCollector` returns a token value in the `RetrieveResult` data object and this token is used to retrieve the remaining properties using the `ContinueRetrievePropertiesEx` API method. For more information, see [Server Data Transmission](#).

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl, fSpecList, ro);
```

10 Print the virtual machine names.

The following code fragment walks the list of `ObjectContent` objects returned in the `RetrieveResult` object. For each object (`ObjectContent`), the inner loop prints the name-value pairs.

```
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        String vmName = null;
        String path = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                vmName = (String) dp.getVal();
                path = dp.getName();
                System.out.println(path + " = " + vmName);
            }
        }
    }
}
```

Results

You can also see these code fragments in the context of an end-to-end example that includes connection and authentication logic, in [Build a Simple vSphere Client Application for the Web Services SDK](#).

Simple Property Collector Program in Java for Web Services SDK

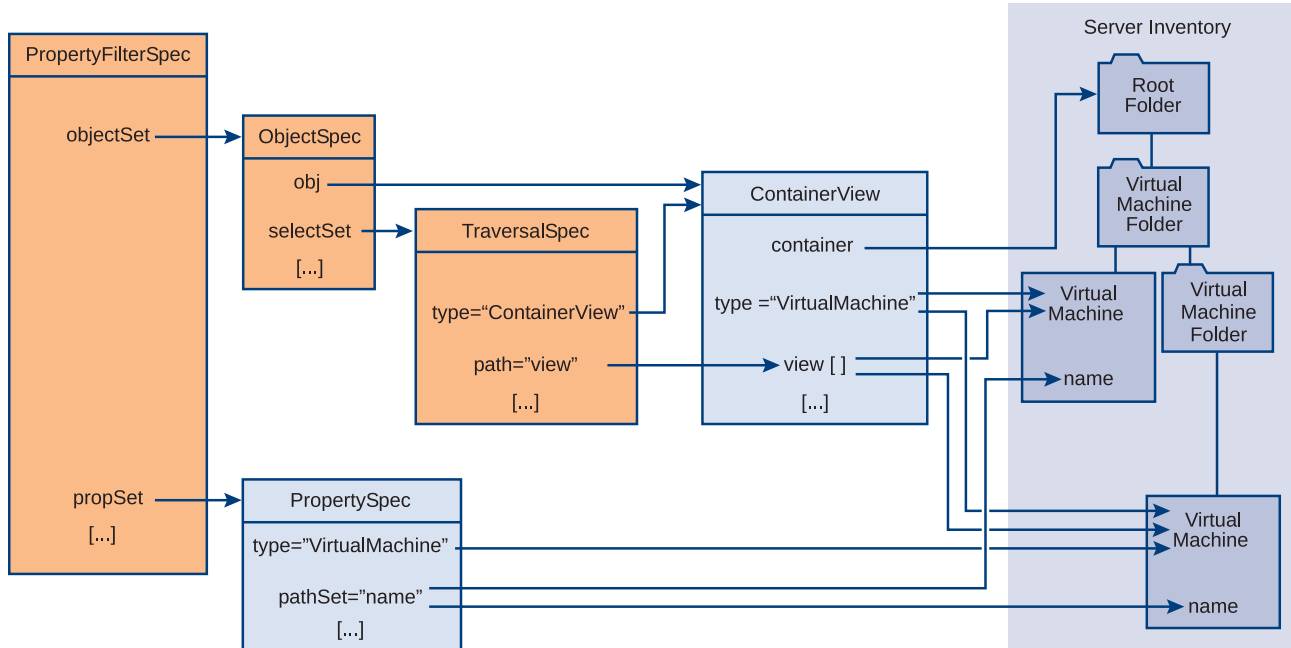
The Property Collector provides a powerful, flexible, and efficient way to collect data from vCenter Server or ESXi.

[Simple PropertyCollector Example \(Java\)](#) is a simple `PropertyCollector` example written in Java. The example uses a `ContainerView` for efficient access to the inventory and a `PropertyFilterSpec` that contains one `ObjectSpec`, one `TraversalSpec`, and one `PropertySpec`. The program performs the following tasks:

- 1 Accepts command line arguments for the vSphere server name (DNS name or IP address), user name, and password.
- 2 Connects to a vSphere server.
- 3 Uses a `ContainerView` to create a subset of the inventory; the subset contains only virtual machines.
- 4 Uses the `RetrievePropertiesEx` method for a single retrieval operation.
- 5 Collects the names of all of the virtual machines in the inventory and prints the names using the standard output stream.
- 6 Closes the connection to the server.

Figure 6-1. Property Filter Specification shows the objects used in Simple PropertyCollector Example (Java). The figure represents properties that identify inventory elements directly or indirectly. It does not show all the properties for the different objects.

Figure 6-1. Property Filter Specification



Example: Simple PropertyCollector Example (Java)

```
import com.vmware.vim25.*;

import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

// PropertyCollector example
// command line input: server name, user name, password

public class PCollector {

    private static void collectProperties(VimPortType methods,
                                         ServiceContent sContent) throws Exception {
        // Get references to the ViewManager and PropertyCollector
        ManagedObjectReference viewMgrRef = sContent.getViewManager();
        ManagedObjectReference propColl = sContent.getPropertyCollector();

        // use a container view for virtual machines to define the traversal
        // - invoke the VimPortType method createContainerView (corresponds
        // to the ViewManager method) - pass the ViewManager MOR and
        // the other parameters required for the method invocation
    }
}
```

```

// - createContainerView takes a string[] for the type parameter;
//   declare an arraylist and add the type string to it
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");

ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
                                                             sContent.getRootFolder(),
                                                             vmList,
                                                             true);

// create an object spec to define the beginning of the traversal;
// container view is the root object for this traversal
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);

// create a traversal spec to select all objects in the view
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setPath("view");
tSpec.setSkip(false);
tSpec.setType("ContainerView");

// add the traversal spec to the object spec;
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the list
oSpec.getSelectSet().add(tSpec);

// specify the property for retrieval (virtual machine name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter method to reference
// the mapped XML representation of the lists and add the specs
// directly to the list
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);

// Create a list for the filters and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl, fSpecList, ro);

// go through the returned list and print out the data
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        String vmName = null;
        String path = null;
    }
}

```



```
List<DynamicProperty> dps = oc.getPropSet();
if (dps != null) {
    for (DynamicProperty dp : dps) {
        vmName = (String) dp.getVal();
        path = dp.getName();
        System.out.println(path + " = " + vmName);
    }
}
}
}
}
}

//end collectProperties()

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
//
// For the purposes of this example, this TrustManager implementation
// will accept all certificates. This is only appropriate for
// a development environment. Production code should implement certificate support.

private static class TrustAllTrustManager
    implements javax.net.ssl.TrustManager,
               javax.net.ssl.X509TrustManager {
    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                                   String authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                                    String authType)
        throws java.security.cert.CertificateException {
        return;
    }
}

public static void main(String [] args) throws Exception {

    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String url = "https://" + serverName + "/sdk/vimService";
```

```

// Variables of the following types for access to the API methods
// and to the vSphere inventory.
// -- ManagedObjectReference for the ServiceInstance on the Server
// -- VimService for access to the vSphere Web service
// -- VimPortType for access to methods
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;

// Declare a host name verifier that will automatically enable
// the connection. The host name verifier is invoked during
// the SSL handshake.
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};

// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Set the default host name verifier to enable the connection.
HttpURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server.
// The client API will include the Server's HTTP cookie in its

```

```

// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
              userName,
              password,
              null);

// retrieve data
collectProperties(vimPort, serviceContent);

// close the connection
vimPort.logout(serviceContent.getSessionManager());

}
}

```

Inventory Traversal

The Property Collector provides a facility to traverse the inventory on the server in arbitrary ways, enabling you to follow links between related objects.

[Simple PropertyCollector Example \(Java\)](#) uses a `ContainerView` to specify the objects that start the collection process. This is the simplest way to set up a filter, using a single reference to a view to provide the `PropertyCollector` with access to a set of objects. To locate objects in the inventory, a filter includes `TraversalSpec` and possibly `SelectionSpec` objects. Use these objects to direct the `PropertyCollector` to the managed objects and their properties named in the `PropertySpec` data objects of the `PropertyFilterSpec`.

TraversalSpec Traversal

Use a `TraversalSpec` object to identify a managed object type and a traversal property in that type. `TraversalSpec` contains the following properties:

- `type` – identifies an inventory object type.
- `path` – specifies a managed object reference property in the `type` object. This property provides the traversal path extending from this object.
- `selectSet` – specifies an optional list of selection objects for additional object traversal paths. The `PropertyCollector` tries to apply the `TraversalSpec` objects in the `selectSet` array to the results of the traversal (the targets of `TraversalSpec.path`). If the type of a referenced managed object matches one of the specs in the `SelectSet` array, the `PropertyCollector` follows that `TraversalSpec` next.

The `selectSet` array can also contain `SelectionSpec` objects; a `SelectionSpec` is a reference to a named `TraversalSpec`. See [SelectionSpec Traversal](#).

- `skip` – indicates whether to collect properties for managed objects referenced by the `path` property.

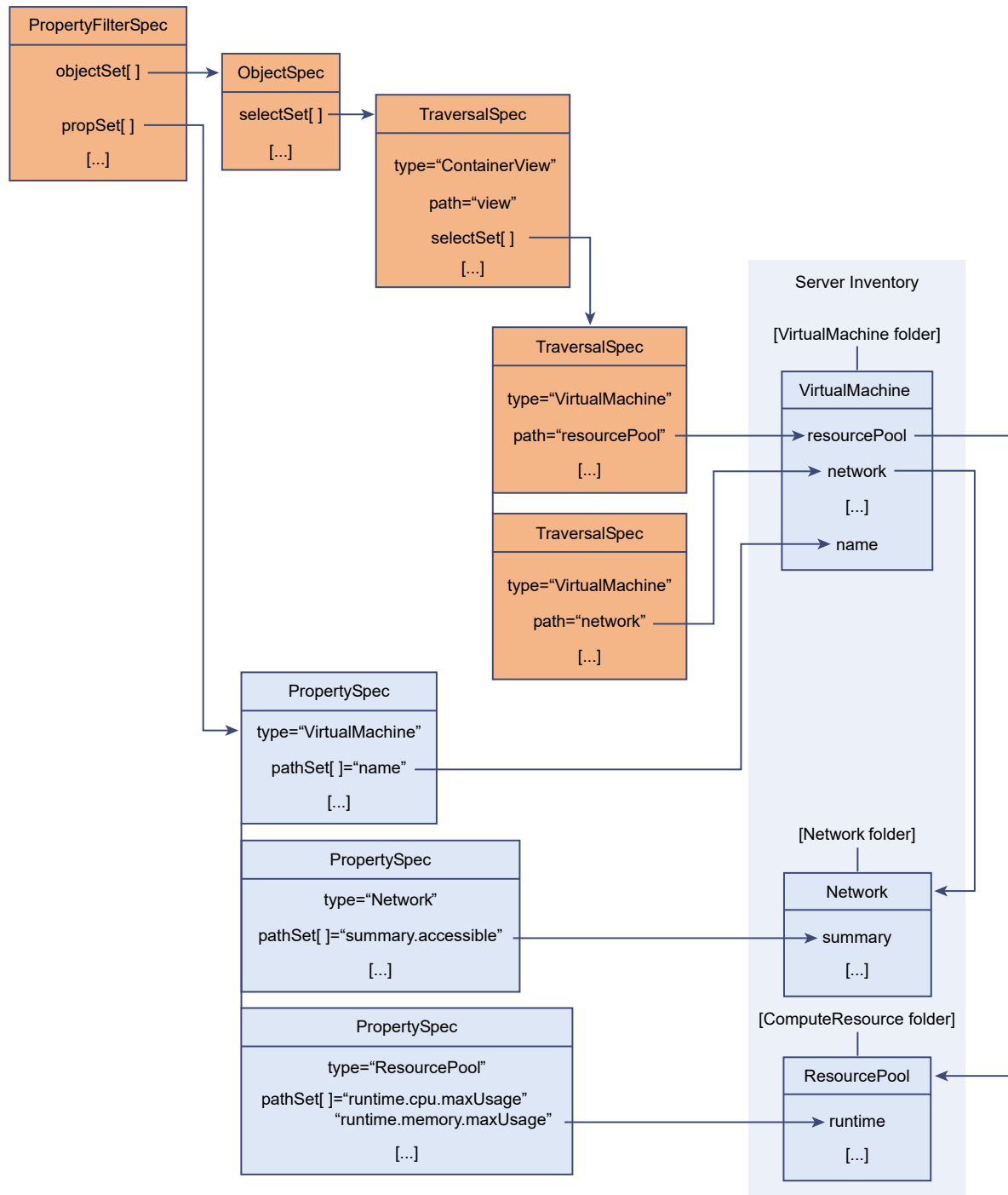
During inventory traversal, the `PropertyCollector` visits managed objects referenced by the `TraversalSpec.path` property values. When the `PropertyCollector` visits an object, it collects properties if the object's type matches a `PropertySpec.type` value.

Traversal starts with the managed object referenced by `ObjectSpec.obj`, and the `PropertyCollector` also collects its properties if it matches a `PropertySpec` and if `ObjectSpec.skip` is `false`. Then the `PropertyCollector` follows the `path` property of each applicable `TraversalSpec` in the `ObjectSpec.selectSet` array. If the `TraversalSpec.skip` property is `false`, the `PropertyCollector` can also collect properties from the objects referenced by the `pathSet` property array.

Finally, the `PropertyCollector` checks the `TraversalSpec.selectSet` array for any traversal specs that apply to the referenced objects, and the traversal continues until no more traversal specs apply.

[Figure 6-2. Inventory Navigation](#) is a representation of a `PropertyFilterSpec` that defines traversal of `VirtualMachine` objects to access related `Network` objects. The filter uses a `ContainerView` as a starting point. The `TraversalSpec` for the `ContainerView` specifies the `view` property for access to the view's virtual machines. The figure shows `TraversalSpec` objects that extend navigation from a `VirtualMachine` object to the associated `Network` and `ResourcePool` objects. The `PropertyCollector` applies these `TraversalSpec` objects to each of the `VirtualMachine` objects in the view list. The figure also shows the `PropertySpec` objects for collecting data from `VirtualMachine`, `Network`, and `ResourcePool` objects.

Figure 6-2. Inventory Navigation



Traverse the Inventory By Using the Property Collector

To traverse inventory objects, you define one or more `TraversalSpec` objects. In each `TraversalSpec` object, you specify a property of one managed object that holds a reference to another managed object. The `TraversalSpec` defines a path that the `PropertyCollector` can follow between objects.

To define inventory traversal, use the following steps.

Procedure

- 1 Create a `ContainerView` for virtual machines.
- 2 Create an `ObjectSpec` that uses the container view as the collection starting point.
- 3 Create a `TraversalSpec` to be applied to the `ContainerView` to select `VirtualMachine` objects.
- 4 Create additional `TraversalSpec` objects to select additional objects.

The `SelectSet` list for the container view `TraversalSpec` has two `TraversalSpec` objects. Both specify a `VirtualMachine` object context. One object uses the `network` property to extend traversal to the `Network` managed object. The other uses the `resourcePool` property to extend traversal to the `ResourcePool` managed object.

- 5 Create `PropertySpec` objects to retrieve `VirtualMachine`, `Network`, and `ResourcePool` properties.

To retrieve properties that are embedded in data objects, the `PropertySpec.PathSet` property uses dot notation to specify the property paths.

Inventory Traversal Example in Java

This example shows a complete Java program that collects and prints selected properties from the inventory, using traversal specs with a property collector. The example lists networks, resource pool status, and virtual machines.

Example: Inventory Traversal in Java

```
import com.vmware.vim25.*;

import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

// PropertyCollector example
// command line input: server name, user name, password

public class PCollector_traversal {

    private static void collectProperties(VimPortType methods,
                                         ServiceContent sContent) throws Exception {

        // Get references to the ViewManager and PropertyCollector
        ManagedObjectReference viewMgrRef = sContent.getViewManager();
        ManagedObjectReference propColl = sContent.getPropertyCollector();
```

```

// use a container view for virtual machines to define the traversal
// - invoke the VimPortType method createContainerView (corresponds
//   to the ViewManager method) - pass the ViewManager MOR and
//   the other parameters required for the method invocation
//   (use a List<String> for the type parameter's string[])
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");

ManagedObjectReference cViewRef =
    methods.createContainerView(viewMgrRef,
                                sContent.getRootFolder(),
                                vmList,
                                true);

// create an object spec to define the beginning of the traversal;
// container view is the root object for this traversal
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);

// create a traversal spec to select all objects in the view
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setPath("view");
tSpec.setSkip(false);
tSpec.setType("ContainerView");

// add the traversal spec to the object spec;
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the selectSet list
oSpec.getSelectSet().add(tSpec);

// extend from virtual machine to network
TraversalSpec tSpecVmN = new TraversalSpec();
tSpecVmN.setType("VirtualMachine");
tSpecVmN.setPath("network");
tSpecVmN.setSkip(false);

// extend from virtual machine to resourcepool
TraversalSpec tSpecVmRp = new TraversalSpec();
tSpecVmRp.setType("VirtualMachine");
tSpecVmRp.setPath("resourcePool");
tSpecVmRp.setSkip(false);

// add the network and resource pool traversal specs
// to the virtual machine traversal;
// the accessor method (getSelectSet) returns a reference
// to the mapped XML representation of the list; using this
// reference to add the spec will update the selectSet list
tSpec.getSelectSet().add(tSpecVmN);
tSpec.getSelectSet().add(tSpecVmRp);

// specify the properties for retrieval

```

```

// (virtual machine name, network summary accessible, rp runtime props);
// the accessor method (getPathSet) returns a reference to the mapped
// XML representation of the list; using this reference to add the
// property names will update the pathSet list
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");

PropertySpec pSpecNs = new PropertySpec();
pSpecNs.setType("Network");
pSpecNs.getPathSet().add("summary.accessible");

PropertySpec pSpecRPr = new PropertySpec();
pSpecRPr.setType("ResourcePool");
pSpecRPr.getPathSet().add("runtime.cpu.maxUsage");
pSpecRPr.getPathSet().add("runtime.memory.maxUsage");
pSpecRPr.getPathSet().add("runtime.overallStatus");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter methods to reference
// the mapped XML representation of the lists and add the specs
// directly to the objectSet and propSet lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
fSpec.getPropSet().add(pSpecNs);
fSpec.getPropSet().add(pSpecRPr);

// Create a list for the filters and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl, fSpecList, ro);

// go through the returned list and print out the data
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        String value = null;
        String path = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                path = dp.getName();
                if (path.equals("name")) {
                    value = (String) dp.getVal();
                }
                else if (path.equals("summary.accessible")) {
                    // summary.accessible is a boolean
                    value = String.valueOf( dp.getVal() );
                }
                else if (path.equals("runtime.cpu.maxUsage")) {
                    // runtime.cpu.maxUsage is an xsd:long
                    value = String.valueOf( dp.getVal() );
                }
            }
        }
    }
}

```



```

    }
    else if (path.equals("runtime.memory.maxUsage")) {
        // runtime.memory.maxUsage is an xsd:long
        value = String.valueOf( dp.getVal() );
    }
    else if (path.equals("runtime.overallStatus")) {
        // runtime.overallStatus is a ManagedEntityStatus enum
        value = String.valueOf( dp.getVal() );
    }

    System.out.println(path + " = " + value);
}
}
}
}
}

} //end collectProperties()

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
//
// For the purposes of this example, this TrustManager implementation
// will accept all certificates. This is only appropriate for
// a development environment. Production code should implement certificate support.
private static class TrustAllTrustManager
    implements javax.net.ssl.TrustManager,
               javax.net.ssl.X509TrustManager {

    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
        return;
    }

}

public static void main(String [] args) throws Exception {

```

```

// arglist variables
String serverName = args[0];
String userName = args[1];
String password = args[2];
String url = "https://" + serverName + "/sdk/vimService";

// Variables of the following types for access to the API methods
// and to the vSphere inventory.
// -- ManagedObjectReference for the ServiceInstance on the Server
// -- VimService for access to the vSphere Web service
// -- VimPortType for access to methods
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;

// Declare a host name verifier that will automatically enable
// the connection. The host name verifier is invoked during
// the SSL handshake.
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        return true;
    }
};

// Create the trust manager.
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;

// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Set the default host name verifier to enable the connection.
HttpURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.

```

```

vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server.
// The client API will include the Server's HTTP cookie in its
// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
              userName,
              password,
              null);

// retrieve data
collectProperties( vimPort, serviceContent );

// close the connection
vimPort.logout(serviceContent.getSessionManager());

}
}

```

SelectionSpec Traversal

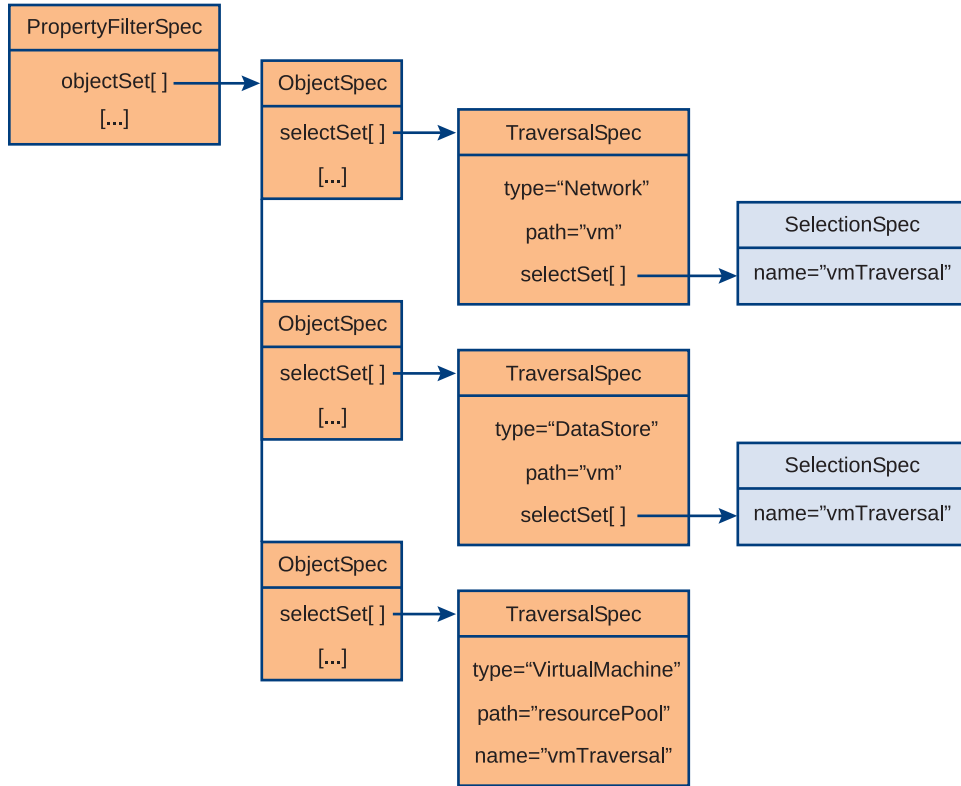
The `selectSet` array in `ObjectSpec` and `TraversalSpec` objects can include `TraversalSpec` objects and `SelectionSpec` objects. `SelectionSpec` is the base class for `TraversalSpec` objects. `SelectionSpec` defines the `name` property. You can use a `SelectionSpec` object in a `selectSet` array as a reference to a named `TraversalSpec` object. By using `SelectionSpec` references, you can reuse a `TraversalSpec` and you can define recursive traversal.

Simple Reference SelectionSpec

Use `SelectionSpec` references to avoid writing duplicate `TraversalSpec` declarations.

The `TraversalSpec` identified in a `SelectionSpec` reference must be within the same `PropertyFilterSpec`. [Figure 6-3. SelectionSpec Reference](#) shows the use of `SelectionSpec` references to a virtual machine `TraversalSpec`. The `SelectionSpec` references are associated with `Network` and `Datastore` traversals.

Figure 6-3. SelectionSpec Reference

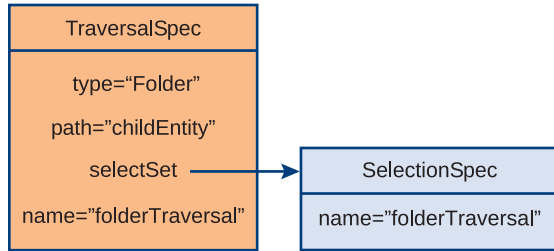


If the `ObjectSpec.selectSet` array contains a `SelectionSpec`, the referenced `TraversalSpec` must identify the same object type. `TraversalSpec.type` must match the type of the object specified in `ObjectSpec.obj`. The `PropertyCollector` applies the `TraversalSpec` to the object and uses the `TraversalSpec.path` property to extend its traversal.

Recursive Traversal

Use a `SelectionSpec` to apply a `TraversalSpec` to the results of its own traversal. To use a recursive filter construction, create a `SelectionSpec` that specifies the name of a `TraversalSpec` and add it to the named `TraversalSpec` selection set. The recursive construction extends inventory traversal beyond the paths directly represented by `TraversalSpec` objects.

You can use recursive traversal on any inventory objects that can be nested. See [Inventory Hierarchies and ServiceInstance](#) for a general representation of the structure of an inventory. For example, on a vCenter Server, folders can nest to arbitrary depths. To describe a traversal path through a succession of folders, you can add a `SelectionSpec` to the `Folder TraversalSpec`. The `SelectionSpec` must reference the `TraversalSpec`. [Figure 6-4. Recursive TraversalSpec and SelectionSpec](#) shows a representation of a `TraversalSpec` and its associated `SelectionSpec` for nested folder traversal.

Figure 6-4. Recursive TraversalSpec and SelectionSpec

Use Recursive TraversalSpec to Traverse Nested Inventory Folders

To traverse inventory objects that are nested to a variable depth, define one or more recursive `TraversalSpec` objects. In each `TraversalSpec` object, you specify a property of one managed object that holds a reference to another managed object. The `TraversalSpec` defines a path that the property collector can follow between objects.

To define recursive inventory traversal, use the following steps.

Procedure

- 1 Use the `SearchIndex` managed object to retrieve the managed object reference for the top-level virtual machine folder.

This folder is used as the beginning of the inventory traversal. For more information see [SearchIndex](#).

- 2 Create an `ObjectSpec` object that references the top-level virtual machine folder.
- 3 Create a `SelectionSpec` object that references the `Folder` `TraversalSpec` by name.
- 4 Create a named `TraversalSpec` for `Folder` objects.

The `TraversalSpec.path` property identifies the `Folder.childEntity` property for traversal to any child objects.

- 5 Add the `SelectionSpec` to the `TraversalSpec` to create the recursive filter.
- 6 Add the `TraversalSpec` to the `ObjectSpec`.
- 7 Create a `PropertySpec` for the `Folder` name.
- 8 Add the object and property specifications to the `PropertyFilterSpec`.
- 9 Call the `RetrievePropertiesEx` method.

Nested Folder Traversal in Java

This example shows the use of a recursive `TraversalSpec` with the property collector. The recursive traversal spec enables the property collector to navigate a chain of nested objects of the same type.

Example: Nested Folder Traversal

```

import com.vmware.vim25.*;

import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

// PropertyCollector example
// command line input: server name, user name, password

public class nestedTraversal {

    private static void collectProperties(VimPortType methods,
                                         ServiceContent sContent)
        throws Exception {

        // Get reference to the PropertyCollector
        ManagedObjectReference propColl = sContent.getPropertyCollector();

        // get the top-level vm folder mor
        ManagedObjectReference sIndex = sContent.getSearchIndex();
        ManagedObjectReference rootVmFolder =
            methods.findByInventoryPath(sIndex, "datacenter1/vm");
        // create an object spec to define the beginning of the traversal;
        // root vm folder is the root object for this traversal
        ObjectSpec oSpec = new ObjectSpec();
        oSpec.setObj(rootVmFolder);
        oSpec.setSkip(true);

        // folder traversal reference
        SelectionSpec sSpecF = new SelectionSpec();
        sSpecF.setName("traverseFolder");

        // create a folder traversal spec to select childEntity
        TraversalSpec tSpecF = new TraversalSpec();
        tSpecF.setType("Folder");
        tSpecF.setPath("childEntity");
        tSpecF.setSkip(false);
        tSpecF.setName("traverseFolder");

        // use the SelectionSpec as a reflexive spec for the folder traversal;
        // the accessor method (getSelectSet) returns a reference to the
        // mapped XML representation of the list; using this reference
        // to add the spec will update the list
        tSpecF.getSelectSet().add(sSpecF);

        // add folder traversal to object spec
        oSpec.getSelectSet().add(tSpecF);

        // specify the property for retrieval (folder name)
        PropertySpec pSpec = new PropertySpec();

```

```

pSpec.setType("Folder");
pSpec.getPathSet().add("name");

// create a PropertyFilterSpec and add the object and
// property specs to it; use the getter method to reference
// the mapped XML representation of the lists and add the specs
// directly to the lists
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);

// Create a list for the filter and add the spec to it
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

// get the data from the server
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl, fSpecList, ro);

// go through the returned list and print out the data
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        String folderName = null;
        String path = null;
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                folderName = (String) dp.getVal();
                path = dp.getName();
                System.out.println(path + " = " + folderName);
            }
        }
    }
}
} //end collectProperties()

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
//
// For the purposes of this example, this TrustManager implementation
// will accept all certificates. This is only appropriate for
// a development environment. Production code should implement certificate support.
private static class TrustAllTrustManager
    implements javax.net.ssl.TrustManager,
        javax.net.ssl.X509TrustManager {

    public java.security.cert.X509Certificate[] getAcceptedIssuers() {
        return null;
    }

    public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }
}

```

```

    public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
        return true;
    }

    public void checkServerTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
        return;
    }

    public void checkClientTrusted(java.security.cert.X509Certificate[] certs, String
authType)
        throws java.security.cert.CertificateException {
        return;
    }
}

public static void main(String [] args) throws Exception {

    // arglist variables
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String url = "https://" + serverName + "/sdk/vimService";
    // Variables of the following types for access to the API methods
    // and to the vSphere inventory.
    // -- ManagedObjectReference for the ServiceInstance on the Server
    // -- VimService for access to the vSphere Web service
    // -- VimPortType for access to methods
    // -- ServiceContent for access to managed object services
    ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
    VimService vimService;
    VimPortType vimPort;
    ServiceContent serviceContent;

    // Declare a host name verifier that will automatically enable
    // the connection. The host name verifier is invoked during
    // the SSL handshake.
    HostnameVerifier hv = new HostnameVerifier() {
        public boolean verify(String urlHostName, SSLSession session) {
            return true;
        }
    };

    // Create the trust manager.
    javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
    javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
    trustAllCerts[0] = tm;

    // Create the SSL context
    javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");

    // Create the session context
    javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();

```



```

// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);

// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

// Set the default host name verifier to enable the connection.
HttpURLConnection.setDefaultHostnameVerifier(hv);

// Set up the manufactured managed object reference for the ServiceInstance
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

// Create a VimService object to obtain a VimPort binding provider.
// The BindingProvider provides access to the protocol fields
// in request/response messages. Retrieve the request context
// which will be used for processing message requests.
vimService = new VimService();
vimPort = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();

// Store the Server URL in the request context and specify true
// to maintain the connection between the client and server.
// The client API will include the Server's HTTP cookie in its
// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
              userName,
              password,
              null);

// retrieve data
collectProperties( vimPort, serviceContent );

// close the connection
vimPort.logout(serviceContent.getSessionManager());

}
}

```

Client Data Synchronization (WaitForUpdatesEx)

To maintain a client-side representation of server object state (by monitoring the properties for the inventory), use the `CreateFilter` and `WaitForUpdatesEx` methods. The `WaitForUpdatesEx` method supports an incremental retrieval model. When the

`WaitForUpdatesEx` call returns an `UpdateSet`, apply the updates as patches to the properties of managed objects in your client-side copy.

In some circumstances, the server can return redundant updates. If the client response includes side effects in addition to updating client data, the additional actions should be idempotent.

The `WaitForUpdatesEx` call cannot guarantee synchronization between client and server in the presence of rapid server data changes, but when the server state is static calls to `WaitForUpdatesEx` result in eventual synchronization.

Note If you create additional instances of `PropertyCollector`, each instance uses independent version IDs for data synchronization. You should maintain a separate client copy of server data for each `PropertyCollector` instance when you use `WaitForUpdatesEx`.

Important The filters you use for incremental retrieval persist for the duration of the session or until you destroy them.

Property Filters

A `PropertyCollector` can have one or more associated `PropertyFilter` objects.

A `PropertyFilter` has one or more associated `PropertyFilterSpec` objects. A `PropertyFilterSpec` that is used with the `RetrievePropertiesEx` method has a limited lifespan; the server destroys the filter after returning results to your client. For a sequence of incremental property collection operations, the `WaitForUpdatesEx` method relies on `PropertyFilterSpec` objects that are available for multiple calls to the method.

To create persistent property filter specifications, use the `CreateFilter` method. When you call `CreateFilter`, you pass a `PropertyFilterSpec` object to the method. The method adds the new filter to the `PropertyCollector` associated with the method invocation and returns a reference to the new filter. After you have created the filter, you can add additional `PropertyFilterSpec` objects. You cannot share a filter with a `PropertyCollector` in another session.

WaitForUpdatesEx

The `WaitForUpdatesEx` method supports a polling mechanism for property collection that is based on a specified wait time.

Specify the following parameters when you call `WaitForUpdatesEx`:

- Managed object reference to a `PropertyCollector` instance.
- `version` value that identifies a sequence value. The first time you call `WaitForUpdatesEx`, specify an empty string ("") to retrieve a complete set of results for the specified properties. Your subsequent calls should use the version value returned in the previous call. If you don't include the version value, the server returns everything. For more information about data versions, see [Server Data Transmission](#).

- `options` specifying the amount of data to transmit in a single response (the `WaitOptions.maxObjectUpdates` property) and the number of seconds the `PropertyCollector` should wait for updates (the `WaitOptions.maxWaitSeconds` property).

The value of the `WaitOptions.maxWaitSeconds` property determines whether the `PropertyCollector` uses an instant retrieval or a polling model. When you call `WaitForUpdatesEx` with a wait time of 0, it checks for updates and returns immediately. When you call `WaitForUpdatesEx` with a wait time greater than 0, the method waits until the specified time or until a change. `WaitForUpdatesEx` blocks your process until updates occur or until it times out. The time-out is affected by the `maxWaitSeconds` value, the amount of time it takes to collect updated property values, and `PropertyCollector` policy.

If the property collection operation times out, and there are no updates to the requested properties, the `PropertyCollector` returns null for the `WaitForUpdatesEx` response.

- `maxWaitSeconds` is an optional property. If you do not specify a value, the `PropertyCollector` waits as long as possible for updates. Therefore, if `maxWaitSeconds` is unset, the `waitForUpdatesEx` method will block the thread after all of the data has been retrieved, waiting for the TCP connection with the vSphere server to timeout. Your code can handle this in one of the following ways: call `waitForUpdatesEx` from a separate thread; look for specific updates and then stop calling the method; or change the TCP connection timeout, `BindingProviderProperties.CONNECT_TIMEOUT`.
- `maxWaitSeconds` set to zero specifies an immediate call and response. The `PropertyCollector` checks for updates for all properties specified by the union of all filters associated with that instance of the `PropertyCollector`. The `PropertyCollector` returns any results, or null if there have been no updates.
- `maxWaitSeconds` greater than zero specifies a wait followed by polling. The `PropertyCollector` returns null if no updates are available within `maxWaitSeconds`.

Comparing Usage of `MaxWaitSeconds`

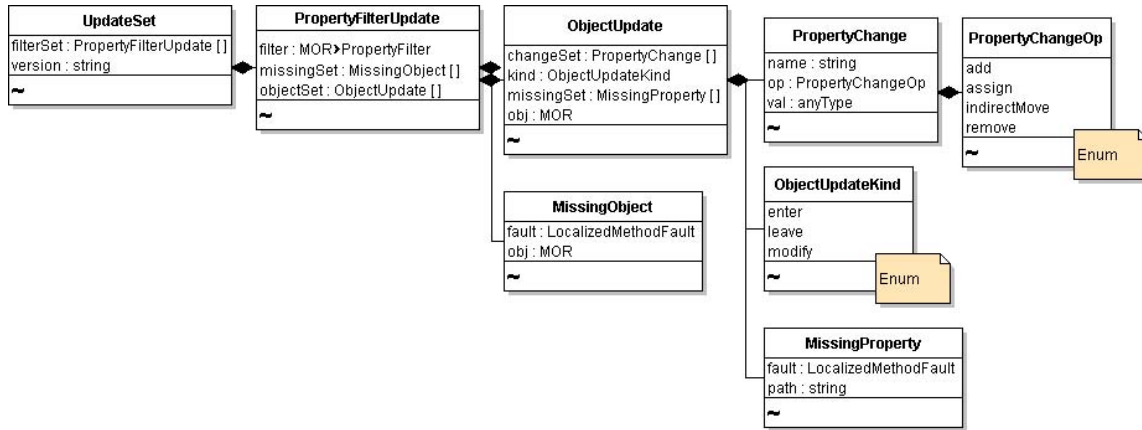
The following table lists some of the advantages and disadvantages of these two operations.

Operation	Advantages	Disadvantages
<code>MaxWaitSeconds = 0</code>	Returns only properties that have changed since the version specified. Returns changed data only, providing better network utilization than <code>RetrieveProperties</code> .	Returns an empty set even when nothing has changed on the server. Depending on your client application, this might be inefficient.
<code>MaxWaitSeconds > 0</code>	Blocks thread until an update occurs. Efficient use of network resources. The only operation that you can cancel.	Blocks processing thread until updates occur. However, this call can be cancelled so you can monitor the time the operation is taking and cancel if necessary.

UpdateSet Data Object from the Property Collector

The `WaitForUpdatesEx` method returns an `UpdateSet` data object, the composite data structure shown in the following figure.

Figure 6-5. UpdateSet Data Object Returned by WaitForUpdates Operations



Server Data Transmission

Property collection can involve the retrieval of large amounts of data, depending on the number of properties implied in the collection request. The vSphere server supports segmented data transmission, or chunking, when it sends collected data to a client. If the amount of collected data exceeds the chunk size, the server returns a chunk of data in a single response, and indicates additional data can be retrieved. For information about chunk size, see the description of the `RetrieveOptions.maxObjects` and `WaitOptions.maxObjectUpdates` properties in the *vSphere API Reference*.

- The `WaitForUpdatesEx` method returns an `UpdateSet` data object. The `UpdateSet.truncated` property indicates whether you must call `WaitForUpdatesEx` again to retrieve additional data. If `truncated` is `true`, the `WaitForUpdatesEx` method returns a version string to identify chunked data. When your client application receives an indication that additional data are available, it must send the returned `UpdateSet.version` string in the subsequent call to `WaitForUpdatesEx` to retrieve the next chunk of data.
- The `RetrievePropertiesEx` method returns a `RetrieveResult` data object. The `RetrieveResult.token` property indicates whether you must call the `ContinueRetrievePropertiesEx` method to retrieve additional data. If the `token` property has a value, it identifies chunked data. When your client application receives an indication that additional data are available, it must send the returned token in the subsequent call to `ContinueRetrievePropertiesEx` to retrieve the next chunk of data.

Version strings and tokens are sequenced. Your client application must keep track of the sequence of values. If an error interrupts the collection operation, resume the operation by using the version string or token that was submitted before the interruption.

PropertyCollector Performance

These factors can affect the performance of a `PropertyCollector` for any given session:

- Number of objects
- Number of properties
- Density of property data (composite, nested data objects)
- Frequency of changes to the objects and properties on the server
- Depth of traversal (number of properties traversed)

In addition, a vSphere server is affected by the number of `PropertyCollector` instances and the number of filters each instance is supporting across all sessions on the server.

To minimize `PropertyCollector` overhead and the amount of network traffic for your client application, use `View` objects with the `PropertyCollector`. [Simple PropertyCollector Example \(Java\)](#) illustrates using views with the `PropertyCollector`.

SearchIndex

The `SearchIndex` managed object provides a set of methods to retrieve references to managed objects in the vSphere inventory. You can search by managed objects inventory path, IP address, datastore path, DNS name, and various other identifying attributes.

For example, if you know the IP address of a virtual machine, you can obtain its managed object reference by using the `SearchIndex.FindByIp` method. You can use `SearchIndex` to obtain the reference to a server object, and then use that reference as the starting object for property collection. See the sample applications `SearchIndex.java` and `SearchIndex.cs` for more information about using `SearchIndex`. See the *vSphere API Reference* for more information about `SearchIndex` methods.

Authentication and Authorization

7

VMware vSphere implements mechanisms to ensure that only valid users can access virtual infrastructure components. Each property and method in the API has an associated privilege requirement, and only users with corresponding privileges can access the entities. This chapter discusses approaches to securing the system and the related service interfaces. The chapter also discusses the user model, which is different in ESXi systems and vCenter Server systems.

See the *vSphere Datacenter Administration Guide* for a list of required privileges for common tasks and best practices for roles and permissions.

This chapter includes the following topics:

- [Objects for Authentication and Authorization Management](#)
- [Authentication and Authorization for ESXi and vCenter Server](#)
- [Obtaining User and Group Information from UserDirectory](#)
- [Managing ESXi Users with HostLocalAccountManager](#)
- [Managing Roles and Permissions with AuthorizationManager](#)
- [Authenticating Users Through SessionManager](#)
- [Using the Credential Store for Automated Login](#)
- [Managing Licenses with LicenseManager](#)

Objects for Authentication and Authorization Management

VMware vSphere includes the following interfaces for authenticating users and protecting virtual infrastructure components from unauthorized access:

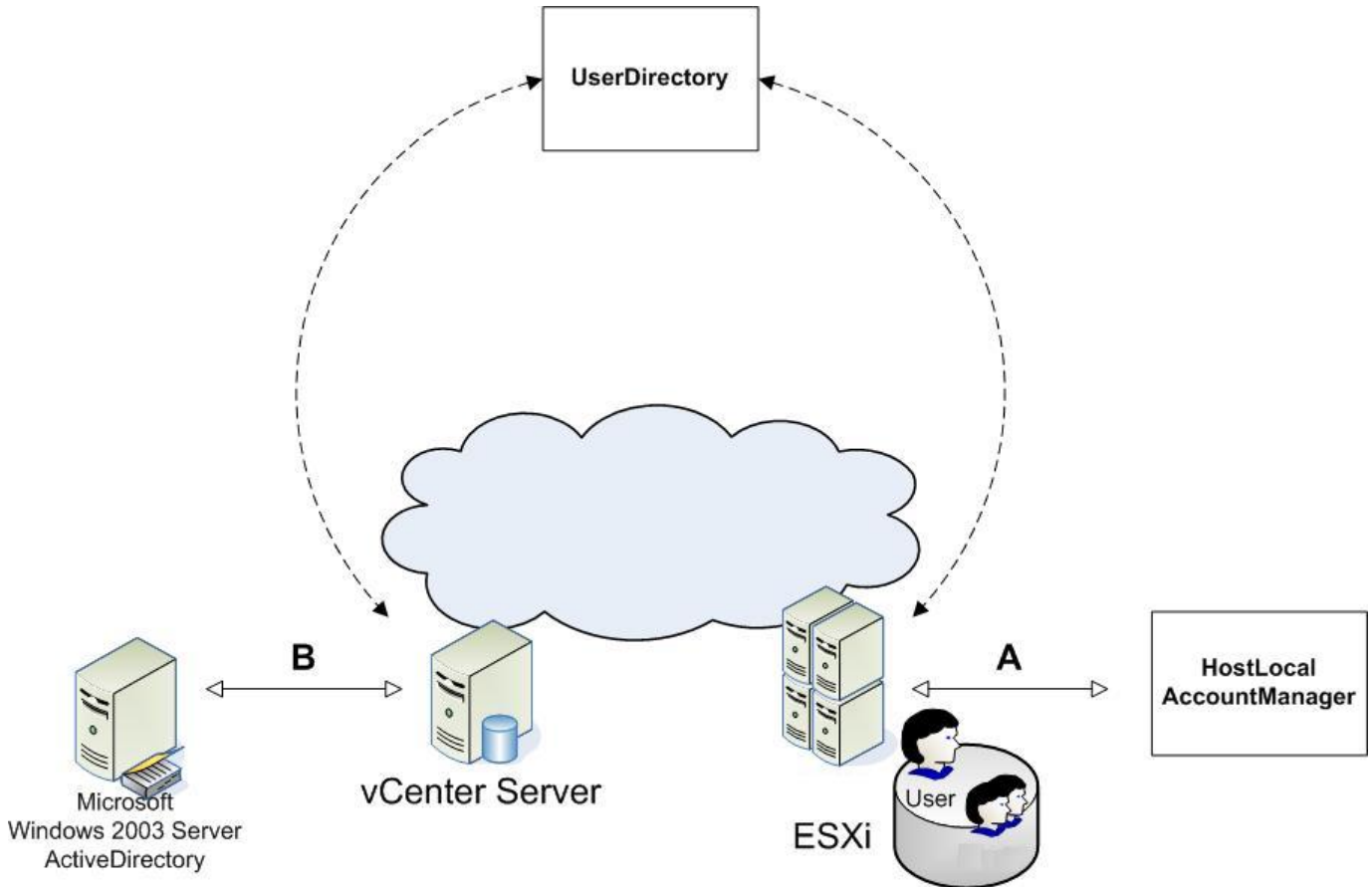
- `HostLocalAccountManager` is used to create and manage user accounts on ESXi systems. Authenticated users can view objects or invoke operations on the server depending on the permissions associated with their account. See [Managing ESXi Users with HostLocalAccountManager](#).

- `AuthorizationManager` protects vSphere components from unauthorized access. Access to components is role-based: Users are assigned roles that encompass the privileges needed to view and perform operations on vSphere objects. `AuthorizationManager` has operations for creating new roles, modifying roles, setting permissions on entities, and handling the relationship between managed objects and permissions.
- `UserDirectory` provides a look-up mechanism that returns user-account information to `AuthorizationManager` or to another requestor, such as a client application. See [Obtaining User and Group Information from UserDirectory](#).
- `SessionManager` provides an interface to the authentication infrastructure on the target server system (see [Authenticating Users Through SessionManager](#)).
 - For vCenter Server systems, `SessionManager` supports single sign-on based on SSO tokens obtained from a VMware SSO Server. See [Establishing a Single Sign-On Session with a vCenter Server](#).
 - For ESXi systems, `SessionManager` supports authenticating user accounts as defined on the host system, such as accounts created using vSphere Client or accounts created programmatically through the `HostLocalAccountManager` API.
- Even if a user is authorized to perform operations on a vSphere object, the operation fails if the licenses for the host or the feature have not been assigned. You use `LicenseManager` and `LicenseAssignmentManager` to manage the licenses. See [Managing Licenses with LicenseManager](#).

Authentication and Authorization for ESXi and vCenter Server

Several server-side mechanisms authenticate a human user when a client application, such as the vSphere Client or a vSphere Web Services SDK application, connects to the server. Because ESXi uses Linux-based authentication, and vCenter Server ran on Windows, the two systems can use different approaches for handling user accounts. The following illustration shows two different user management mechanisms associated with the VMware vSphere server.

Figure 7-1. Managed Objects for Handling User Accounts



These services work together to ensure that only authenticated users can connect to ESXi or vCenter Server systems, and that they can access only those objects—folders, virtual machines, datacenters, virtual services, and so on—for which they have the required privileges and which they are authorized to use or to view.

In addition, the vSphere Web Services SDK supports automated login through a credential store. See [Using the Credential Store for Automated Login](#).

ESXi User Model

When users enter their user account and credential from a client application, the server consults the appropriate user account store and validates the authenticity of the user account and the associated credential. Currently, the credential consists of a password, but vSphere also supports certificates, such as X.509 certificates. Authenticated users can then access objects they are authorized to use. Authentication succeeds if a user identity exists as a user account on the target system or in a supported directory service.

ESXi leverages standard Linux infrastructure, including the Linux pluggable authentication module (PAM) mechanism for user account creation and management. The VMware authentication daemon (`vmware-authd`) is implemented as a PAM module. You can create and manage user accounts on an ESXi system by using `HostLocalAccountManager`.

vCenter Server User Model

The pre-appliance vCenter Server was a Windows-based service using native Windows facilities and the Windows user model for identification and authentication. The vCenter Server Web service is associated with the Windows user account that was logged in to the machine for vCenter Server installation. This vCenter Server administrator account was a member of the local Windows Administrator group on the machine.

VMware recommended creating a dedicated Windows user account for installing and managing the vCenter Server system. Other vCenter Server users who connect to the Web service also needed a Windows account on the local Administrator group.

More recently the vCenter Server Appliance uses the Platform Services Controller for authentication.

Important Even if a user with the same name exists on an ESXi host and a vCenter Server system, the two users have different accounts.

For details, see the *Datacenter Administration Guide* in the VMware vSphere documentation set.

Organizations that are using Microsoft Active Directory can use the user identities contained in a Windows Server domain controller or Active Directory service across their virtual infrastructure. Microsoft Active Directory identities are supported for all clients that run vSphere Web Services SDK applications from Windows-based systems.

A vCenter Server client uses a SAML token to establish a single sign on session with the Server. See [Establishing a Single Sign-On Session with a vCenter Server](#).

vSphere Security Model

Although the details of authentication and authorization differ between ESXi and vCenter Server, the model itself is the same for both systems. It relies on privileges, roles, and permissions.

Privileges

A privilege is a system-defined requirement associated with a VMware vSphere object. Privileges are defined by VMware. Privileges are static, and do not change for a single version of a product. Each managed object has one or more privileges that a principal (user, group member) must have to invoke an operation or to view a property. For example, managed entities such as `Folder` and `VirtualMachine` require the principal to have the `System.Read` privilege on the entity to view the values of its properties.

The *vSphere API Reference* includes information about privileges required to invoke operations and to view properties on the **Required Privileges** labels on the documentation page for each managed object. Privileges for vSphere components are defined as follows:

```
<group>[.<group>].privilege
```

For example:

```
Datacenter.Create
Host.Config.Connection
Host.Config.Snmp
```

A privilege might be specific to vCenter Server or to ESXi systems. For example, the `Alarm.Create` privilege is defined on vCenter Server. Setting alarms is done through the `AlarmManager` service interface, which requires a running vCenter Server system.

Privilege requirements apply to system objects regardless of how a given client application attempts to access server content (vSphere Client, CLI, or SDK). For example, you can use the following URL to access virtual machine datastore files:

```
https://<hostname>/folder[/<path>]?dcPath=<datacenter_path>[&dsName=<datastore_name>]
```

The URL accesses a `Datastore` object in the inventory. You must have privileges to access each object in the hierarchy, corresponding to the elements of the URL.

Privileges for Datastore Objects in the Web Services API

The following table shows the privileges needed by methods that access datastore objects..

Object Associated with File	URL Element	Required Privileges
Root folder	/folder	System.View
Datacenter	?dcPath	Datastore.Browse
		Datastore.FileManagement
Datastore	&dsName	Datastore.Browse
		Datastore.FileManagement
Host	/host	Host.Config.AdvancedConfig
	/tmp/	Host.Config.SystemManagement

Roles

A role is a predefined set of privileges. Users are granted privileges to objects through roles.

When you assign user or group permissions, you pair the user or group with a role and associate that pairing with an inventory object. A single user might have different roles for different objects in the inventory.

For example, if you have two resource pools in your inventory, Pool A and Pool B, you might assign a particular user the role `Virtual Machine User` on Pool A and the role `ReadOnly` on Pool B. These assignments allow that user to turn on virtual machines in Pool A. In Pool B, the user can view the status of virtual machines, but cannot turn on virtual machines.

Permissions

In vSphere, a permission consists of a user or group and an assigned role for an inventory object, such as a virtual machine or ESXi host. Permissions grant users the right to perform the activities specified by the role on the object to which the role is assigned.

For example, to configure memory for an ESXi host, a user must be granted a role that includes the `Host.Configuration.Memory` privilege. By assigning different roles to users or groups for different objects, you can control the tasks that users can perform in your vSphere environment.

Many tasks require permissions on more than one object.

Setting Up Users, Groups, and Permissions

Setting up users, groups, and permissions consists of these tasks:

- 1 Get information about privilege requirements and privileges associated with system and sample roles.
 - Find out which operations on vSphere objects require which privileges. See the *API Reference*.
 - Find out which operations the system roles and sample roles can perform. See [Description of Roles on vSphere Servers](#).
- 2 If necessary, create additional roles (sets of privileges). See [Modifying Sample Roles to Create New Roles](#).
- 3 Retrieve information about existing users and groups (see [Obtaining User and Group Information from UserDirectory](#)) and create additional groups if needed.
- 4 Associate users or groups with roles using permissions. See [Managing Roles and Permissions with AuthorizationManager](#).

At runtime, use `SessionManager` to log in to the server. vCenter Servers support single sign-on sessions. To establish a single sign-on session, use the `SessionManager.LoginByToken` method. To establish a session with a standalone ESXi host, use the `SessionManager.Login` method.

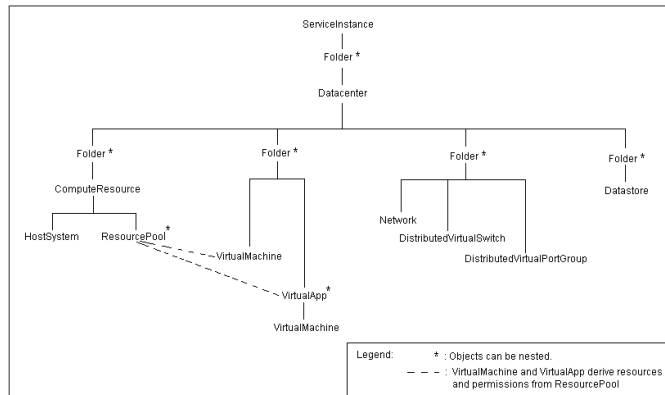
Obtaining User and Group Information from UserDirectory

The `UserDirectory` managed object allows a client application to obtain information about users and groups on a VMware vSphere server. Properties and results vary, depending on whether the server is a vCenter Server or an ESXi system.

- vCenter Server system. Domain controller, Active Directory, or local Windows account repository.
- ESXi host. Linux password file in `/etc/passwd` on the host.

For example, vCenter Server user accounts can be managed in a Windows Active Directory server or domain controller from which the `domainList` property of `UserDirectory` is derived. For ESXi systems, the `domainList` property is empty.

Figure 7-2. UserDirectory Managed Object



RetrieveUserGroups Method

`UserDirectory` allows you to obtain information about users and groups using the `RetrieveUserGroups` method. The method can obtain a list of all user accounts from the host, and can search for specific users or groups based on specific criteria to filter the results. You can search by user name, by group name, for an exact match, or for a partial string (substring).

- ESXi does not support local user groups, so this method will not return group information for a host. This method will return information about Active Directory groups.
- For ESXi systems, search returns all users from the `passwd` file. If this file contains Network Information System (NIS) or NIS+ users, `RetrieveUserGroups` returns these accounts as well.
- For vCenter Server, search is limited to the specified Windows domain. If the domain is omitted, the search is performed on local users and groups.

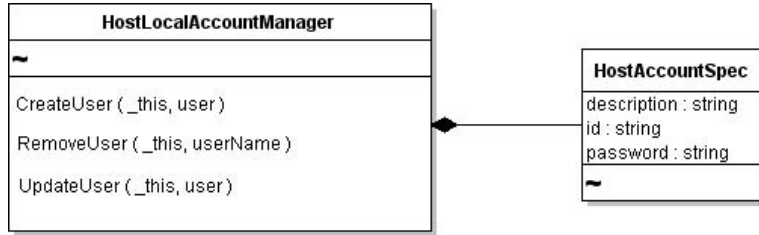
Important Do not configure an ESXi system to use NIS or NIS+, unless it is acceptable to have NIS (or NIS+) user information available through the `UserDirectory.RetrieveUserGroups` API.

Managing ESXi Users with `HostLocalAccountManager`

The `HostLocalAccountManager` managed object supports user administration tasks. `HostLocalAccountManager` is available only on ESXi system.

Important vCenter Server systems use different methods. See [vCenter Server User Model](#).

Figure 7-3. HostLocalAccountManager Managed Object



Methods Available for Local Account Management

`HostLocalAccountManager` provides the following methods for local user account management:

- `CreateUser`
- `RemoveUser`
- `UpdateUser`

These methods accept a `HostAccountSpec` data object. Specify the object properties according to the requirements on the target system. Examples of user account requirements are password length requirements and restricted use of dictionary words.

Create a Local User Account on an ESXi System

You can use the API to create a user account on an ESXi system.

Procedure

- 1 Obtain a managed object reference to the `HostLocalAccountManager` of the target system.
- 2 Create a `HostAccountSpec` data object that defines the properties of the user account, including description and password.

Define account names and passwords according to the configuration required by your ESXi system for user account naming conventions and password requirements, such as minimum length, character set, and other requirements.

- 3 Call the `HostLocalAccountManager.CreateUserAccount` method, passing in the managed object reference (from step 1) and the `HostAccountSpec` data object (step 2).

After creating user accounts on the ESXi system, you can grant these users access to virtual components by using `AuthorizationManager` methods. See [Managing Roles and Permissions with AuthorizationManager](#).

Managing Roles and Permissions with AuthorizationManager

`AuthorizationManager` is the service interface for handling permissions and roles assigned to the users and groups you define with `HostLocalAccountManager`. `AuthorizationManager` methods allow you to create, modify, and manage roles and permissions, and to obtain

information about the roles and permissions defined in the system. If a predefined role does not meet your needs, define a new one that contains only the minimum set of required privileges.

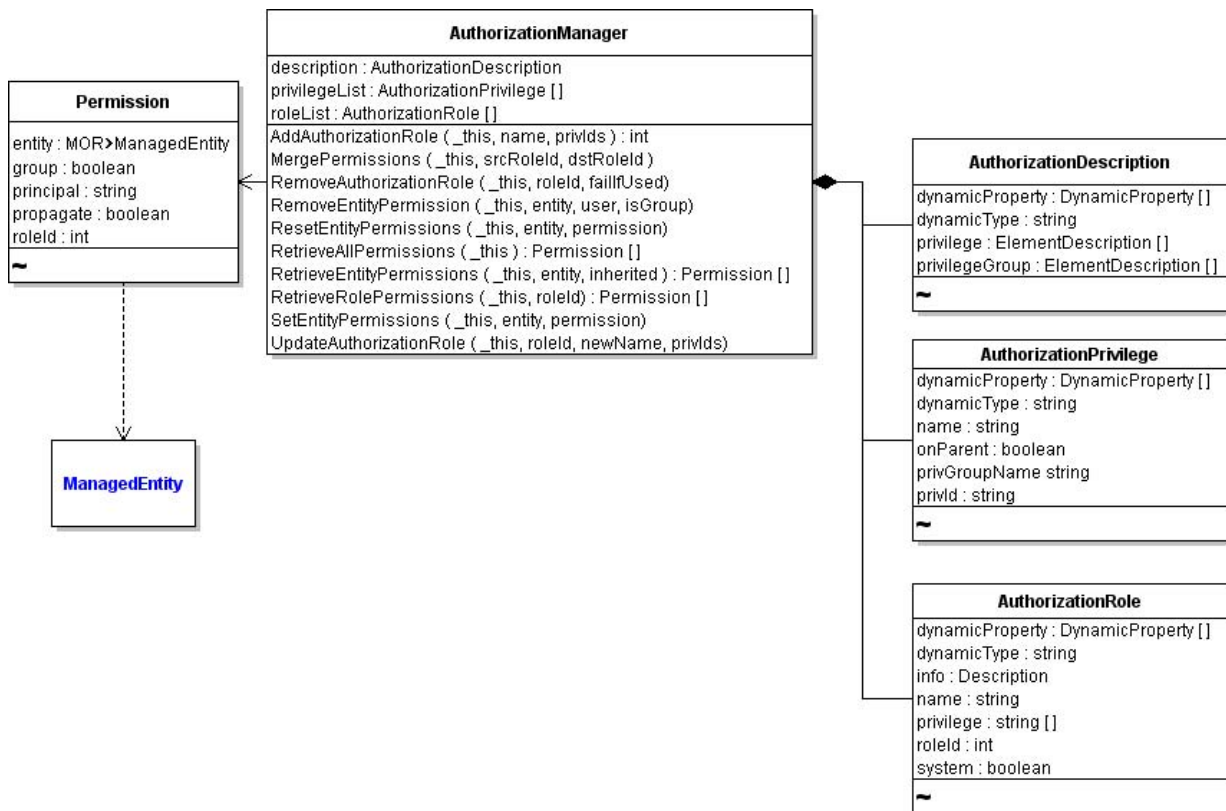
The `AuthorizationManager` also allows access and prevents access to specific server objects based on the permissions associated with the object.

`AuthorizationManager` includes methods for managing roles and for managing permissions:

- **Roles Management.** `AddAuthorizationRole`, `RemoveAuthorizationRole`, and `UpdateAuthorizationRole`. See [Using Roles to Consolidate Sets of Privileges](#) and [Modifying Sample Roles to Create New Roles](#).
- **Permissions Management.** `MergePermissions`, `RemoveEntityPermission`, `ResetEntityPermissions`, `RetrieveAllPermissions`, `RetrieveEntityPermissions`, `RetrieveRolePermissions`, and `SetEntityPermissions`. See [Granting Privileges Through Permissions](#).

The following diagram shows these methods in a UML diagram for `AuthorizationManager` and some of its associated data objects.

Figure 7-4. `AuthorizationManager` Managed Object



`AuthorizationManager` properties allow access to information. For example:

- The `privilegeList` property returns a list of all privileges defined on the system, as an array of `AuthorizationPrivilege` data objects. Privileges are defined by VMware, on the objects and properties contained in the system. These privileges are fixed and cannot be changed by client applications.
- The `roleList` property returns a list of all currently defined roles, including the system-defined roles, as an array of `AuthorizationRole` data objects.

Using Roles to Consolidate Sets of Privileges

A role is a named set of one or more privileges. A role is normally defined for a group of people who have common responsibilities in the system, for example, administrators. Each role can have zero to multiple privileges. ESXi defines system roles and user roles.

- System roles. Cannot be modified or deleted.
- User roles. Apply to different user communities or restrict access for add-on tools. Several predefined user roles are included with vCenter Server and with ESXi systems. You can create new roles using these predefined user roles as a starting point.

Description of Roles on vSphere Servers

The following table describes system roles and user roles in more detail and lists currently available roles as examples.

Type	Role name	Role ID	Description
System Roles	Administrator	-1	Superuser access. Encompasses the set of all defined privileges. This role cannot be deleted. By default, the Administrator role is granted to the user or group that owns the root node.
	Anonymous	-4	Cannot be granted. Default access role associated with any user account that has logged in.
	No Access	-5	No access. Explicitly denies access to the user or group with this role. Assigning this role to a user account prevents the user from seeing any objects. Use the <code>No Access</code> role to mask subobjects under a higher-level object that has propagated permissions defined.
	Read-Only	-2	Read-only access. Encompasses the set of all nonmutable privileges. (<code>System.Anonymous</code> , <code>System.Read</code> , and <code>System.View</code>). Equivalent to a user role with no permissions. Users with this role can read data or properties and call query methods, but cannot make changes to the system.
	View	-3	Visibility access consisting of <code>System.Anonymous</code> and <code>System.View</code> privileges. Cannot be granted.
Sample Roles	Virtual Machine Administrator	1	Set of privileges necessary to manage virtual machines and hosts within the system.

Type	Role name	Role ID	Description
	Datacenter Administrator	2	Set of privileges necessary to manage resources, but not interact with virtual machines.
	Virtual Machine Provider	3	Set of privileges necessary to provision resources.
	Virtual Machine Power User	4	Set of privileges for a virtual machine user that can also make configuration changes and create new virtual machines.
	Virtual Machine User	5	Set of privileges necessary to use virtual machines only. Cannot reconfigure virtual machines.
	ResourcePool Administrator	6	Available on vCenter Server systems only.
	VMware Consolidated Backup Utility	7	Available on vCenter Server systems only. Set of privileges necessary to run the Consolidated Backup Utility.

Modifying Sample Roles to Create New Roles

The system roles cannot be modified or deleted. However, you can create new roles, or modify the sample roles.

To create new roles using the API, use the following steps.

Procedure

- 1 Starting with the `ServiceContent` object in `ServiceInstance.content`, obtain a managed object reference to the `AuthorizationManager` for the server.
- 2 Invoke the `AddAuthorizationRole` method. Parameters are a reference to `AuthorizationManager`, a name for the role (as a string), and an array of privileges (array of strings) that should be assigned to the role.

`AddAuthorizationRole` returns an integer (`xsd:int`) value for the `roleId` that the system assigns to the newly defined role.

- 3 In subsequent code, use the `roleId` to assign the role to specific users or groups.

Granting Privileges Through Permissions

When you use one of the `AuthorizationManager` objects to assign or modify permissions, you use a `Permission` data object. `Permission` associates a principal with a set of privileges. A permission identifies:

- The user or group (`principal`) to which the permission applies.
- The role containing the privileges that should be granted to the user or group.
- The managed object reference to the entity to which the permission applies.

Every managed entity has at least one `Permission` object associated with it. A managed entity can have more than one `Permission` assigned to it, effectively granting different privileges to different users or groups. Permissions are defined for managed entities either explicitly or through inheritance.

Obtaining Information About Permissions

Users with the Administrator role can obtain information about `Permission` objects at different levels of detail.

- For an array of `Permission` objects, call the `AuthorizationManager.RetrieveAllPermissions` method.
- For specific inventory objects, such as managed entities, folders, datacenters, or virtual services, call the `AuthorizationManager.RetrieveEntityPermissions` method.
- For a role defined in the system, call the `AuthorizationManager.RetrieveRolePermissions` method.

See the *vSphere API Reference*.

Setting, Changing, or Deleting Permissions

The `Permission` data object associates the privileges required to perform an action on an object with the principals (user, group). Principals have privileges through their role. To set or update permissions on an object, use the `AuthorizationManager.SetEntityPermissions` method.

To set permissions on an entity, use the following steps.

Procedure

- 1 Obtain a reference to the `AuthorizationManager` for the server from the `ServiceContent` object associated with the `ServiceInstance`. For example:

```
ManagedObjectReference hostAuthorizationManager = service.getAuthorizationManager();
```

- 2 Create a `Permission` data object that identifies the user (or group) name, the role, the entity to which the permission should apply, and whether the permission should be applied to the entity's children.

For example, the following code fragment creates a permission on the root folder of the inventory granting a user Administrator role to the root folder and all its children.

```
Permission per = new Permission();
per.setGroup(false);
per.setPrincipal("new_user_name");
per.setRoleId(-1);
per.setPropagate(true);
per.setEntity(rootFolder);
```

Permissions cannot be set directly on children in a complex entity. For complex entities, set permissions on the parent entity and set the `propagate` flag to true to apply permissions to the child entities.

To replace existing permissions with a new set of permissions, use the `AuthorizationManager.ResetEntityPermissions` method.

Impact of Group Membership on Permissions

Users can be members of multiple groups. The system handles multigroup membership as follows:

- Permissions are applied to inventory objects from the containing object to each of its child entities.
- If a user has no explicit user-level permissions, group-level permissions apply as if granted to the user directly.
- Membership in multiple groups with permissions on the same object results in a union of permissions.
- User-level permissions always take precedence over group-level permissions.

Applying Permission to a Managed Entity

The following example shows some of the code required to create a user account and apply a permission to an entity that grants access to the user account based on a role.

The role with role ID 4, assigned in this example, is defined as a “Virtual Machine Power User.” The sample uses `AuthorizationManager` to grant permissions to the user and to associate the permission with the managed entity in the inventory—in this example, the `rootFolder`. The example uses the `apputil` helper classes to access the objects.

Example: Creating a User Account

```
...
ManagedObjectReference _authManRef = _sic.getAuthorizationManager();
public class CreateUser {
    private static AppUtil appUtil= null;
    private void createUser() throws Exception {
        ManagedObjectReference hostLocalAccountManager =
            appUtil.getConnection().getServiceContent().getAccountManager();
        ManagedObjectReference hostAuthorizationManager =
            appUtil.getConnection().getServiceContent().getAuthorizationManager();

        // Create a user
        HostAccountSpec hostAccountSpec = new HostAccountSpec();
        hostAccountSpec.setId(userName);
        hostAccountSpec.setPassword(password);
        hostAccountSpec.setDescription("my delegated admin auto-agent software");
        appUtil.getConnection().getService().createUser(hostLocalAccountManager, hostAccountSpec);
        ManagedObjectReference rootFolder =
            appUtil.getConnection().getServiceContent().getRootFolder();
```

```

Permission permission = new Permission();
permission.setGroup(false);
permission.setPrincipal(userName);

// Assign the Virtual Machine Power User role
permission.setRoleId(4);
permission.setPropagate(true);
permission.setEntity(rootFolder);
appUtil.getConnection().getService().setEntityPermissions(hostAuthorizationManager,
rootFolder,
    new Permission [] {permission});
...

```

Authenticating Users Through SessionManager

The `SessionManager` managed object controls user access to the server. `SessionManager` includes methods for logging in to the server, obtaining a session, and logging out. The `SessionManager` defines the lifetime and visibility of many objects. Session-specific objects are not visible outside the session in which they are created.

Important Each user session uses system resources and creates locks on the server side. Too many concurrent sessions can slow down the server. By default, vCenter Server terminates a session after 30 minutes.

Upon successful authentication of a user account, `SessionManager` returns a `UserSession` data object to the client application. The session is associated with that user account for the duration of the session. The client application can save the session locally, to a secure file, and reuse the session later to reconnect to the server. You can also configure an ESXi or vCenter Server system to support local sessions, which enable users with credentials on the host to log in based on those privileges.

The `SessionManager` provides these capabilities:

- Log in and log out. Basic operations to log in to ESXi or vCenter Server system, obtain a session, and log out. When a session terminates, all session-specific objects are destroyed.
- Impersonation. One user session adopts the authorization level of another user session. Impersonation is common in Web based scenarios in which a middle-tier application functions as a central account that interacts with other back-end servers or processes. Windows services impersonate a client when accessing resources on behalf of the client. `SessionManager` supports impersonation through its `ImpersonateUser` method.
- Delegation. A client application that is running on behalf of a local user can call the `SessionManager.AcquireLocalTicket` method to obtain a one-time user name and password for logging in. Delegation is useful for host-based utilities that run in the local console.

If the user account associated with the session does not have the permissions required to perform an action, the `AuthorizationManager` returns a `NoPermission` fault to the client application.

Using VMware Single Sign On for vCenter Server Sessions

vSphere supports single sign on for a single point of authentication for vCenter Server clients. To use VMware Single Sign On, your vSphere Web Services SDK client connects to the VMware SSO Server to obtain an SSO token. Your client includes the token in the `SessionManager.LoginByToken` request to start a vSphere session.

For more information about creating a session, see [Establishing a Single Sign-On Session with a vCenter Server](#).

Using the Credential Store for Automated Login

To facilitate automated login for unattended applications, the vSphere Web Services SDK includes client-side credential store libraries and tools for automating the login process in a more secure manner. The libraries eliminate the need for system administrators to keep passwords in local scripts.

Important These libraries are built on top of the vSphere Web Services SDK.

The credential store has the following components:

- A persistence file (credential store backing file) that stores authentication credentials. Currently, only passwords are supported. The persistence file maps a remote user account from an ESXi host to the password for that user on the server.
- Java and Perl libraries for managing the credential store programmatically. See [Credential Store Methods](#) for available methods.
- Java and Microsoft PowerShell-based command-line utilities for managing the credential store.

In addition to the libraries listed in [Credential Store Libraries](#), the vSphere Web Services SDK includes the `CredentialStoreAdmin` tool for creating, examining, and managing the credential store. You can use the tool to examine the contents of the credential store, for example, the generated user accounts and passwords.

If you use these credential store client libraries in an application, you must set up the credential store on all client machines that run your application.

Credential Store Libraries

The following table shows the credential management libraries for Java.

Package com.vmware.security.credstore (Java)

CredentialStore.java

CredentialStoreFactory.java

Several of the helper classes provided with the sample applications use the credential store mechanism.

Credential Store Methods

The following table shows the methods used in Java clients for managing credentials.

Java	Description
<code>addPassword(hostname, username, password)</code>	Stores the password for the specified host and user. Overwrites any existing password for that user in the credential store. Creates the default credential store backing file in the default location (if it does not exist).
<code>removePassword(hostname, username)</code>	Deletes the password for the specified user from the credential store.
<code>clearPasswords()</code>	Deletes all passwords from the credential store.
<code>getPassword(hostname, username)</code>	Returns the password for the specified host and user from the credential store.
<code>getHosts()</code>	Returns the set of hosts contained in the credential store.
<code>getUsernames(hostname)</code>	Returns the collection of all user names that have passwords stored for the specified <code>hostname</code> .
<code>close()</code>	Closes the credential store, preventing further method invocations. Releases associated resources.

Credential Store Backing File

The credential store backing file is an XML file that is saved locally on the client machine for access at runtime. Unless otherwise specified, the backing file is located in the following location:

- Linux. `$HOME/.vmware/credstore/vicredentials.xml`
- Windows Vista and later. `C:\Users\[user_name]\AppData\Roaming\VMware\credstore\vicredentials.xml`

- Windows XP and Windows 2000.

```
C:\Documents and Settings\[user_name]\Application
Data\VMware\credstore\vicredentials.xml
```

The credential store persists locally on a per-user basis—each user has his or her own credential store backing file.

Caution The credential store backing files use filesystem-level permissions to ensure that passwords remain confidential. Protect the credential store backing file with appropriate file permissions.

[Credential Store File Format](#) shows the XML elements that are read and written to the file.

Example: Credential Store File Format

```
<?xml version="1.0" encoding="UTF-8"?>
  <viCredentials>
    <version>1.0</version>
    <passwordEntry>
      <host>mi6.vmware.com</host>
      <username>agent007</username>
      <password>IhWS1saIhtsw2FbIh0w2F2...</password>
    </passwordEntry>
    <passwordEntry>
      ...
    </passwordEntry>
    ...
  </viCredentials>
```

Credential Store Samples

The `CreateUser` and `SimpleAgent` sample applications demonstrate how to use the credential store client libraries.

- The `CreateUser` sample creates a user account and password for the server based on random-number-generation scheme. The sample populates the local credential store backing file with this information. If the backing file does not exist, it is created in the default location.

When you run `CreateUser`, specify the name of an ESXi system, and an administrator user name and password. A user account name and password are created on the server. Specify `--ignorecert` unless your system has a secure connection to the target. Do not use `--ignorecert` in a production environment.

```
java com.vmware.samples.simpleagent.CreateUser --server <servername> --url
https://<servername>/sdk --username <adminuser> --password <pwd> --ignorecert ignorecert
```

Caution The `CreateUser` sample application is for demonstration purposes only and should not be used as a model for production code. The sample breaks the principle of least privilege by granting the user account the Administrator role (-1). Never do this in a production environment.

- The `SimpleAgent` sample application demonstrates how to use credential store libraries to extract the user account and password at runtime to authenticate a user noninteractively.

```
java com.vmware.samples.simpleagent.SimpleAgent <servername>
```

Specifying Roles and Users with the Credential Store

VMware recommends that you apply the principle of least privilege to any agent-like software or automated application that uses the credential store in a production environment. Give user accounts the minimal number of privileges on the system that they require to do their jobs.

Specify roles and users as follows:

Procedure

- 1 For each SDK-based application, use one specific role, newly created or predefined, that has appropriate privileges.

For example, if you are developing an agent-like application to automatically start the VMware Consolidated Backup utility, you might use the “VMware Consolidated Backup Utility” role (roleID 7).

If no predefined user role that meets the needs of your application exists, create a role with only those privileges needed for the application. See [Using Roles to Consolidate Sets of Privileges](#) for more information about roles.

- 2 Create a user account for use with the agent or application.
- 3 Apply the role created in [Step 1](#) to the user account created in [Step 2](#).
- 4 Store the user account and password in the credential store, using the `CredentialStoreAdministration` tool.

Never grant administrator privileges to a user account associated with an automated script or software agent, especially one that uses the credential store.

Managing Licenses with LicenseManager

When you want to perform tasks in the vSphere environment, you must have licenses to do so. Licensing applies to ESXi hosts, vCenter Server, and special features such as VMware HA or VMware vMotion.

The *vSphere Datacenter Administration Guide* explains how to manage ESXi and vCenter Server licenses using the vSphere Client, and gives background information about license keys, license inventory, and related topics.

You can also manage licenses using the `LicenseManager` and `LicenseAssignmentManager` managed objects. You use `LicenseManager` to explicitly manage the pool of available licenses on ESXi systems released before vSphere 4.0. You use `LicenseAssignmentManager`, available through the `LicenseManager.licenseAssignmentManager` property, to manage assignment of licenses to entities in the vCenter Server inventory. You can retrieve information, add licenses, and remove licenses.

Retrieve Information

- Retrieve the `LicenseManager.evaluation` and `LicenseManager.licenses` properties to obtain information on evaluation licenses and full licenses.
- Call `LicenseManager.DecodeLicense` to decode license information. The call returns a `LicenseManagerLicenseInfo` data object, which encapsulates information about the license.
- Call `LicenseAssignmentManager.QueryAssignedLicenses` for information about assigned licenses.

Add Licenses

- Call `LicenseManager.AddLicense`, passing in a license key, to add a license to the inventory of available licenses.
- Call `LicenseAssignmentManager.UpdateAssignedLicense`, passing in a license key, to update the licenses for an entity, for example, a host system.

Remove Licenses

- Call `LicenseAssignmentManager.RemoveAssignedLicense` to remove all licenses from an entity, passing in an entity to remove licenses from. You can then assign those licenses to other entities.
- Call `LicenseManager.RemoveLicense`, passing in a license key, to remove a license from the inventory of available licenses.

Many of the operations in your vSphere environment involve setting up the ESXi hosts on which the virtualization layer runs. You can set up storage and networking, and those settings directly affect the virtual machine. You must also manage other aspects of the host, as discussed in this chapter.

Important See the *ESX Configuration Guide* and the *ESXi Configuration Guide* for important information on security considerations, not included here.

This chapter includes the following topics:

- [Host Management Objects](#)
- [Retrieving Host Information](#)
- [Configuring and Reconfiguring Hosts](#)
- [Managing the Host Lifecycle](#)
- [Querying and Changing the Host Time](#)
- [Querying Virtual Machine Memory Overhead](#)

Host Management Objects

The vSphere Web Services SDK includes several objects for host management.

The central object is `HostSystem`. Each property of `HostSystem` is a data object that encapsulates some information about the host. For example, the `capability` property is a `HostCapability` object, the `runtime` property is a `HostRuntimeInfo` object. See the *API Reference* for a list of the properties and the corresponding data objects.

`HostSystem` methods allow you to perform certain tasks on ESX/ESXi hosts. However, many tasks are not performed through `HostSystem` methods, but through methods in managed objects related to `HostSystem`. For example, you manage the host time using the `HostDateTimeSystem` and you manage kernel modules using `HostKernelModuleSystem`.

Retrieving Host Information

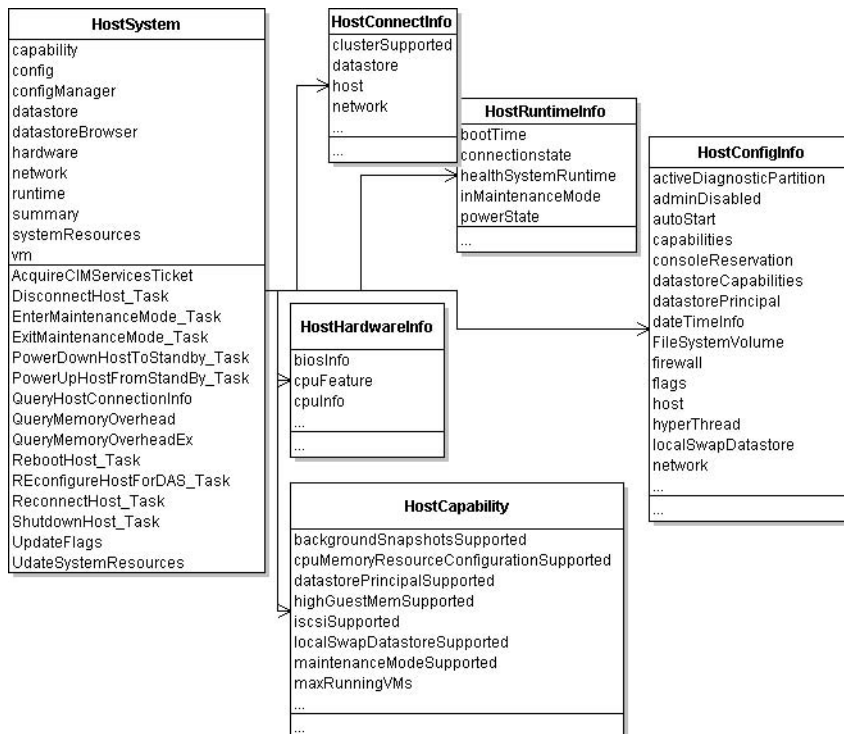
You retrieve information about the host by accessing data objects defined for the `HostSystem`.

- `HostSystem.capability` is a `HostCapability` object. The `HostCapability` properties indicate the features that are supported by the host, for example, `maintenanceModeSupported` or `recursiveResourcePoolsSupported`.
- `HostSystem.runtimeInfo` is a `HostRuntimeInfo` object that contains several data objects with detailed information about the current state of the host. You can, for example, extract the health status as a `HealthSystemRuntime` object or the power state as a `HostPowerState` object.
- `HostSystem.hardware` is a `HostHardwareInfo` object that allows you to retrieve the host's hardware configuration including CPU and NUMA information and memory size.
- `HostSystem.config` is a `HostConfigInfo` object. This data object type encapsulates a typical set of host configuration information that is useful for displaying and configuring a host. You can access the `HostConfigInfo` object only on managed hosts, and only if the host is connected.

`HostSystem` has several additional properties that allow you to directly access the virtual machines, datastores, and networks associated with that system.

The `QueryHostConnectionInfo`, `QueryMemoryOverhead`, and `QueryMemoryOverheadEx` methods are available for information retrieval.

Figure 8-1. `HostSystem` and Information Properties



Configuring and Reconfiguring Hosts

When you configure or reconfigure an ESX/ESXi host, you usually do not use the methods in `HostSystem` directly, but work with managed objects available for configuration of that part of the system. For example, `HostNetworkSystem` allows you to configure the network, and `HostAuthorizationManager` is for managing users, groups, and permissions on a host. The objects and related methods are discussed in the corresponding chapters of this guide.

Some methods are defined locally in `HostSystem`. See the *vSphere API Reference* for details on each method.

- **CIM Management** – `AcquireCimServicesTicket`. For additional information on using vSphere with CIM, see the VMware CIM APIs documentation.
- **Host Lifecycle** – `RebootHost_Task`, `ShutdownHost_Task`, `PowerDownHostToStandBy_Task`, `PowerUpHostFromStandBy_Task`, `DisconnectHost_Task`, `ReconnectHost_Task`. See [Managing the Host Lifecycle](#).
- **Maintenance Mode** – `EnterMaintenanceMode_Task`, `ExitMaintenanceMode_Task`.
- **Updates** – `UpdateFlags`, `UpdateIpmi`, `UpdateSystemResources`.

Managing the Host Lifecycle

A host's lifecycle depends in part on whether the host is a standalone host or managed by a vCenter Server system.

Reboot and Shutdown

You can reboot and shut down managed and standalone hosts. The `ShutdownHost_Task` method is not supported on all hosts. Check the host capability `shutdownSupported`.

You can call both methods with a `force` parameter, which specifies whether to reboot hosts even when virtual machines are running or other operations are in progress on the host. If you set the parameter to `false`, hosts are rebooted only when they are in maintenance mode.

- `ShutdownHost_Task` – Shuts down a host. If connected directly to the host, the client never receives an indicator of success in the returned task, but temporarily loses connection to the host. If the method does not succeed, an error is returned.
- `RebootHost_Task` – Reboots a host. If the command is successful, then the host has been rebooted. Clients connected directly to the host do not receive an indication of success in the returned task, but temporarily lose connection to the host. If the method does not succeed, an error is returned.

Using Standby Mode

Standby is a power state in which the host does not support provisioning or power on of virtual machines. VMware power management module might evacuate and put a

host in standby mode to save power. The host can be powered up remotely by using `PowerUpHostFromStandBy_Task`.

The following methods support standby mode. Both methods are cancelable.

- `PowerDownHostToStandBy_Task` – Puts the host in standby mode, a mode in which the host is in a standby state from which it can be powered up remotely. The command is only supported on hosts on which the host capability `standbySupported` is `true`.

While this task is running, no virtual machines can be powered on and no provisioning operations can be performed on the host.

Calling this method does not directly initiate any operations to evacuate or power down powered-on virtual machines. However, if VMware DRS is enabled, the vCenter Server migrates powered-off virtual machines or recommends migration to a different host, depending on the automation level. If the host is part of a cluster and the task is issued with a vCenter Server target with the method's `evacuatePoweredOffVms` parameter set to `true`, the task does not succeed unless all the powered-off virtual machines are reregistered to other hosts.

- `PowerUpHostFromStandBy_Task` – Takes the host out of standby mode. If the command is successful, the host wakes up and starts sending heartbeats. This method might be called automatically by VMware DRS to add capacity to a cluster, if the host is not in maintenance mode.

Disconnecting and Reconnecting Hosts

You can make a host a managed host by adding it to the vCenter Server system. You can later disconnect and reconnect the host, for example, to refresh the agents.

You can use the following methods, which are only supported if you access the host through a vCenter Server system.

- `QueryHostConnectionInfo` – Returns a `HostConnectInfo` object, which is the same object that the `Datacenter.QueryConnectionInfo` returns. The information in this object can be used by a connection wizard, like the wizard used in the vSphere Client.
- `DisconnectHost_Task` – Disconnects from a host and instructs the vCenter Server system to stop sending heartbeats to the host.
- `ReconnectHost_Task` – Reconnects a host to the vCenter Server system. This process reinstalls agents and reconfigures the host, if it has gotten out of sync with the server. The reconnection process checks for the correct set of licenses and for the number of CPUs on the host, ensures the correct set of agents is installed, and ensures that networks and datastores are discovered and registered with the vCenter Server system.

Client applications can change the IP address and port of the host when doing a reconnect operation. This can be useful if the client wants to preserve existing metadata, such as statistics, alarms, and privileges, even though the host is changing its IP address.

Querying and Changing the Host Time

The `HostDateTimeSystem` supports date and time related configuration on a host and supports both Network Time Protocol (NTP) and Precision Time Protocol (PTP) configuration.

The `HostDateTimeSystem.dateTimeInfo` property allows you to retrieve and set date and time information. The `HostDateTimeInfo` data object's properties contain these data objects for date and time management:

- `HostNTPConfig` contains a list of NTP servers for use by the host.
- `HostPTPConfig` contains device and networking information.
- `HostDateTimeSystemTimeZone` specifies the time zone including the GMT offset, identifier for the time zone, and name.

You can also query the host's time information by calling one of the `HostDateTimeSystem` methods.

- `QueryAvailableTimeZones` – Retrieves the list of available timezones on the host. The method uses the public domain `tz` timezone database. The method returns an array of `HostDateTimeSystemTimeZone` objects.
- `QueryDateTime` – Returns the current date and time on the host.

You can modify the host's date and time information by calling one of the following `HostDateTimeSystem` methods:

- `RefreshDateTimeSystem` – Refreshes the date and time related settings to pick up any changes that might have occurred.
- `UpdateDateTime` – Updates the date and time on the host using the date and time passed into the method. Use with caution. Network delays or execution delays can result in time skews.
- `UpdateDateTimeConfig` – Updates the date and time configuration of the host. You call this method with a `HostDateTimeConfig` parameter, which allows you to specify NTP configuration, PTP configuration, and the time zone.

Querying Virtual Machine Memory Overhead

Each virtual machine you power on requires a certain amount of memory for its use. In addition, the host must have some memory overhead available for each virtual machine. To find out about memory overhead, call the `HostSystem.QueryMemoryOverheadEx` method. The method takes a `VirtualMachineConfigInfo` data object as an argument, and determines the amount of overhead necessary to power on a virtual machine with those characteristics.

The method returns the amount of memory required, in bytes.

A virtual machine uses a virtual disk to store its operating system, program files, and other data. A virtual disk is a large physical file, or a set of files, that can be copied, moved, archived, and backed up like other files. To store and manipulate virtual disk files, a host requires dedicated storage space. ESX/ESXi supports storage in multiple ways. Hosts that are managed by a vCenter Server system can share storage.

Any type of network-attached storage requires complete configuration of networking in the VMkernel to support network-based access to the storage media. The VMkernel requires its own IP address. See [Chapter 10 vSphere Networks](#).

This chapter includes the following topics:

- [Storage Management Objects](#)
- [Introduction to Storage](#)
- [Choosing the Storage API to Use](#)
- [Configuring Disk Partitions](#)
- [Multipath Management](#)
- [Configuring iSCSI Storage](#)
- [Creating and Managing Datastores](#)
- [Managing VMFS Volume Copies \(Resignaturing\)](#)
- [Managing Diagnostic Partitions](#)
- [Sample Code Reference](#)

Storage Management Objects

You can access the objects that support storage management through the `HostSystem` managed object.

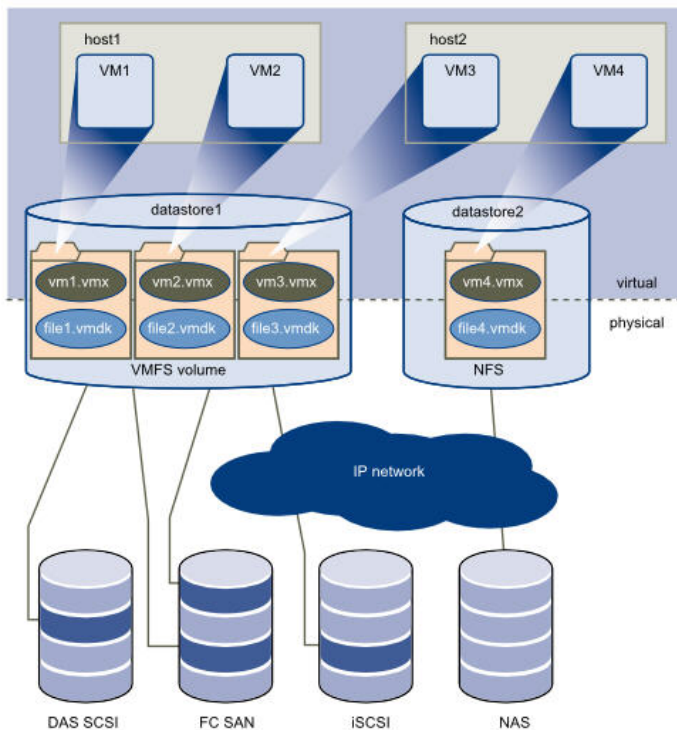
- `HostStorageSystem` – The `HostSystem.storageSystem` property is a managed object reference to the `HostStorageSystem` of the ESX/ESXi system. `HostStorageSystem` is a low-level interface that is used mainly for configuring the physical storage. See [Configuring Disk Partitions](#).

- `HostDatastoreSystem` – The `HostSystem.datastoreSystem` property is a managed object reference to a `HostDatastoreSystem` managed object. `HostDatastoreSystem` methods allow you to create, configure, extend, and remove datastores. While `HostStorageSystem` supports access and configuration of physical storage, `HostDatastoreSystem` supports access and configuration of logical storage through the volumes (`Datastore` managed objects) the host can use for virtual machines. See [Creating and Managing Datastores](#).
- `HostDatastoreBrowser` – Provides access to the contents of one or more datastores. The items in a datastore are files that contain configuration, virtual disk, and other data associated with a virtual machine.
- `Datastore` – The `Datastore` managed entity provides methods for mounting datastores, browsing datastores, and obtaining information about the datastores associated with a virtual machine. See [Creating and Managing Datastores](#).
- `HostDiagnosticPartition` – Supports creating and querying diagnostic partitions for your ESX/ESXi host. See [Managing Diagnostic Partitions](#).

Introduction to Storage

The VMware vSphere storage architecture consists of layers of abstraction that hide and manage the complexity and differences of physical storage subsystems, shown in the following illustration.

Figure 9-1. Storage Architecture



How Virtual Machines Access Storage

Virtual machines use virtual disks for their operating system, application software, and other data files. A virtual disk is stored as a VMDK file on a datastore. The virtual disk hides the physical storage layer from the virtual machine's operating system. Regardless of the type of storage device that your host uses, the virtual disk always appears to the virtual machine as a local SCSI device. As a result, you can run operating systems that are not certified for specific storage equipment, such as SAN, in the virtual machine.

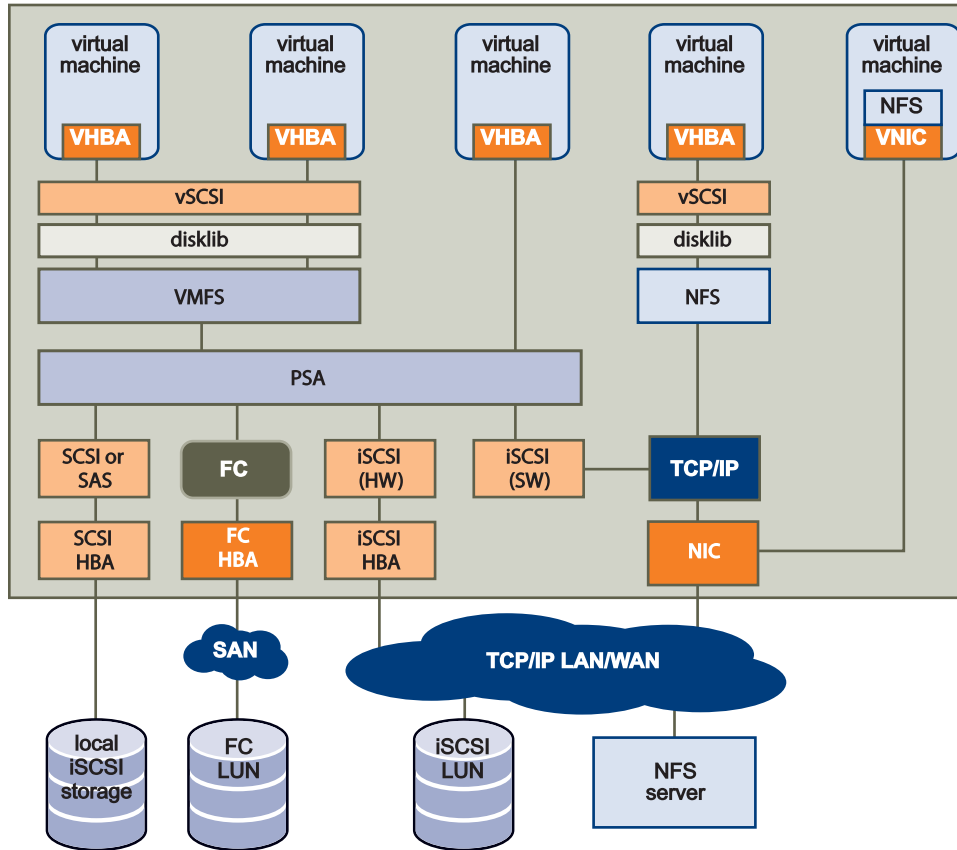
When a virtual machine communicates with its virtual disk stored on a datastore, it issues SCSI commands. Because datastores can exist on different types of physical storage, these commands are encapsulated into other forms, depending on the protocol that the ESX/ESXi host uses to connect to the physical storage device.

To the applications and guest operating systems running on each virtual machine, the storage subsystem appears as a virtual SCSI controller connected to one or more virtual SCSI disks as shown in the top half of [Figure 9-1. Storage Architecture](#). These controllers are the only types of SCSI controllers that a virtual machine can see and access, and include the objects that extend `VirtualSCSIController`:

- `ParaVirtualSCSIController`
- `VirtualBusLogicController`
- `VirtualLsiLogicController`
- `VirtualLsiLogicSASController`

How precisely a virtual machine accesses storage depends on the setup of the host. [Figure 9-2. Storage API Architecture](#) gives an overview of the different possibilities.

Figure 9-2. Storage API Architecture



Datstores

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on. vSphere supports two types of datastores, VMFS and NAS.

- If you want to use a NAS volume, mount it using `CreateNasDatastore` and unmount it using `RemoveDatastore`. The two commands are host specific, you must invoke the create and remove methods on each host on which you want to mount or unmount the datastore.
- To create a VMFS datastore, call `CreateVmfsDatastore`, passing in any existing disk. As a result of the call, the disk is formatted with VMFS and the datastore is automounted on all ESX/ESXi hosts on which the disk is visible the next time you perform a rescan. When you call `RemoveDatastore` on a VMFS datastore, the datastore is destroyed. After a rescan, the datastore is no longer available to any ESX/ESXi systems. In contrast to NAS datastores, you do not have to invoke methods for creation and removal of the datastore on each host.

An ESX/ESXi host automatically discovers the VMFS volume on attached Logical Unit Numbers (LUNs) on startup and after re-scanning the host bus adapter. When you create a VMFS datastore, the datastore label is based on the VMFS volume label. If there is a conflict with an existing datastore, the label is made unique by appending a suffix. The VMFS volume label remains unchanged.

Destroying a VMFS datastore removes the partitions that compose the VMFS volume.

Datastores can span multiple physical storage devices. A single VMFS volume can contain one or more LUNs from a local SCSI disk array on a physical host, a Fibre Channel SAN disk farm, or iSCSI SAN disk farm. The ESX/ESXi system detects new LUNs that are added to any of the physical storage subsystems. When the user queries for a list of available devices, the newly discovered devices are included. You can extend storage capacity on an existing VMFS volume without powering down physical hosts or storage subsystems.

If any of the LUNs within a VMFS volume fails or becomes unavailable, only virtual machines with data on that LUN are affected. An exception is the LUN that has the first extent of the spanned volume (multi-extent volume). All other virtual machines with virtual disks residing on other LUNs continue to function normally.

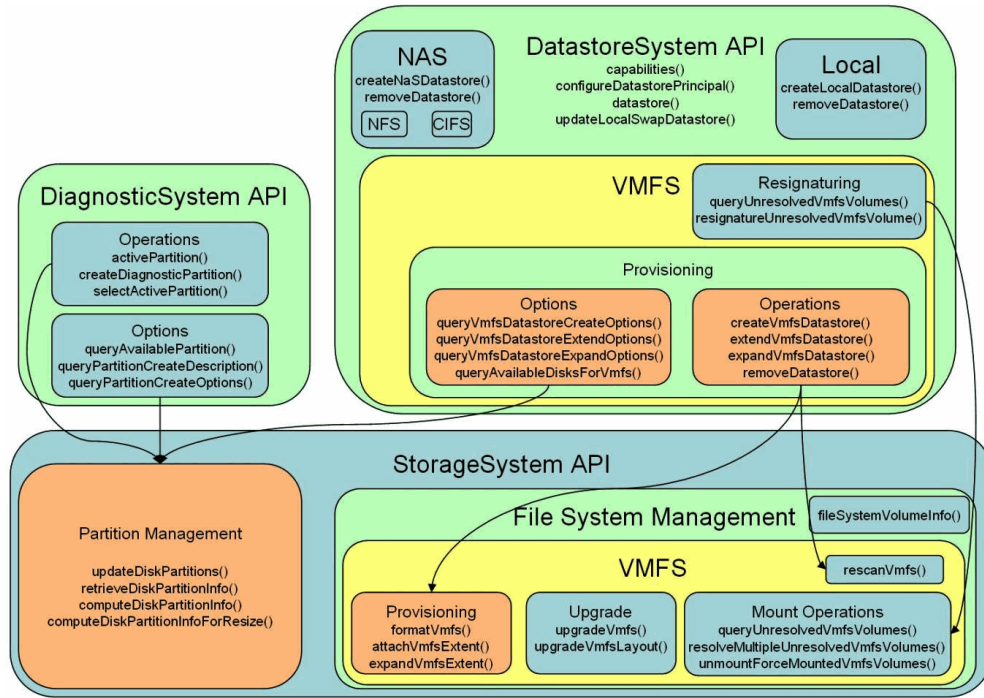
Choosing the Storage API to Use

The `HostStorageSystem` APIs are low-level enough for performing VMFS provisioning operations. They require a knowledge of partitioning details and VMFS extent composition. They do not enforce VMFS best practices like partition alignment and optimum VMFS block sizes, and they allow you to mix extents from different datastores on the same LUN and to add extents even though expansion is preferable in most cases.

The `HostDatastoreSystem` APIs are primarily used for managing VMFS volumes. They don't require an in-depth knowledge of storage systems, and do enforce best practices.

[Figure 9-3. Storage APIs](#) gives an overview of the different APIs. See [Managed Objects for Working with Storage](#) to determine which tasks are commonly performed with which API.

Figure 9-3. Storage APIs



Managed Objects for Working with Storage

The Web Services API provides several managed objects that are designed to work with ESXi storage devices.

Managed Object	Task	See
<code>HostStorageSystem</code>	Low-level operations associated with individual hosts, such as resizing or updating disk partitions.	Configuring Disk Partitions
<code>HostStorageSystem</code>	Multipath management.	Multipath Management
<code>HostStorageSystem</code>	iSCSI Storage setup and configuration.	Configuring iSCSI Storage
<code>HostDatastoreSystem</code>	Creating and managing VMFS datastores and remote datastores.	Creating and Managing Datastores
<code>HostDatastoreSystem</code> <code>HostStorageSystem</code>	Managing VMFS volume copies (resignature or force mount).	Managing VMFS Volume Copies (Resignaturing)
<code>HostDiagnosticSystem</code>	Creating and managing diagnostic partitions.	Managing Diagnostic Partitions

Configuring Disk Partitions

`HostStorageSystem` manages low-level storage components including HBAs, SCSI LUNs, file system volumes, and so on. You can use this API to set up the partitions before creating, extending, or expanding a VMFS file system.

- `ComputeDiskPartitionInfo` – Computes the disk partition information based on the specified disk layout. The server computes a new `HostDiskPartitionInfo` object for a specific disk using the layout that is specified by the `HostDiskPartitionLayout` object. Inside the `HostDiskPartitionLayout` object, you specify the list of block ranges for that partition, and optionally the total number and size of the blocks. You can then use that information inside the `HostDiskPartitionSpec` when updating a disk partition.
- `ComputeDiskPartitionInfoForResize` – Computes the disk partition information to support resizing a given partition. Returns the resized disk partition information as a `HostDiskPartitionInfo` object. You can then use that information inside the `HostDiskPartitionSpec` when resizing the disk partition.
- `RetrieveDiskPartitionInfo` – Allows you to specify an array of device path names that identify disks and returns an array of `HostPartitionInfo` objects for each of those disks.
- `UpdateDiskPartitions` – Changes the partitions on a disk by supplying a partition specification (`HostDiskPartitionSpec`) and device name.

After you have updated the disk partitions for the host, you must perform a rescan by using one of the following methods. Complete rescans might take a long time.

- `RefreshStorageSystem` – Refreshes the storage information and settings to pick up changes, but does not explicitly issue commands to discover new devices.
- `RescanAllHba` – Rescans all host bus adapters for new storage devices. This method might take a long time.
- `RescanHba` – Rescans a specific host bus adapter for new devices.

`HostStorageSystem` methods are also used for setting up iSCSI storage. See [Configuring iSCSI Storage](#).

Multipath Management

The *vSphere Storage* documentation includes information about using multipathing for failover and load balancing. You can manage multipathing using the vSphere Client, the `esxcli` command, or using the following commands. Use the `HostStorageSystem.multipathStateInfo` property to access the `HostMultipathStateInfo` data object that describes runtime information about the state of multipathing on a given host.

- `EnableMultipathPath` – Enables a disabled path for a device. Use the pathname from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.

- `QueryPathSelectionPolicyOptions` – Obtains the set of path-selection-policy options. These options determine the path that can be used by a device that is managed by native multipathing. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
- `QueryStorageArrayTypePolicyOptions` – Obtains the set of storage-array-type policy options. These options determine the storage-array-type policies that a device that is managed by native multipathing might use. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
- `SetMultipathLunPolicy` – Updates the path selection policy for a LUN. Specify the LUN using the LUN UUID from the `HostMultipathInfoLogicalUnit` object.
- `DisableMultipathPath` – Disables an enabled path for a device. Use the pathname from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.

Configuring iSCSI Storage

vSphere supports software iSCSI, dependent hardware iSCSI, and independent hardware iSCSI. See *Configuring iSCSI Adapters and Storage* in the *vSphere Storage* documentation for a detailed discussion.

The following `HostStorageSystem` methods are available for iSCSI storage management.

- Add a dynamic or static target.
 - `AddInternetScsiSendTarget` – Adds send target entries to the host bus adapter discovery list if the `DiscoveryProperties.sendTargetsDiscoveryEnabled` flag is set to true.
 - `AddInternetScsiStaticTargets` – Adds static target entries to the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` flag must be set to true.
- Configure targets.
 - `UpdateInternetScsiAdvancedOptions` – Updates the advanced options that the iSCSI host bus adapter or the discovery addresses and targets associated with it.
 - `UpdateInternetScsiAlias` – Updates the alias of an iSCSI host bus adapter.
 - `UpdateInternetScsiAuthenticationProperties` – Updates the authentication properties for one or more targets or discovery addresses associated with an iSCSI host bus adapter.
 - `UpdateInternetScsiDigestProperties` – Updates the digest properties for the iSCSI host bus adapter or the discovery addresses and targets associated with it.
 - `UpdateInternetScsiDiscoveryProperties` – Updates the discovery properties for an iSCSI host bus adapter.

- `UpdateInternetScsiIPProperties` – Updates the IP properties for an iSCSI host bus adapter.
- `UpdateInternetScsiName` – Updates the name of an iSCSI host bus adapter.
- `UpdateSoftwareInternetScsiEnabled` – Enables and disables software iSCSI in the VMkernel.
- Remove a dynamic or static target.
 - `RemoveInternetScsiSendTargets` – Removes send target entries from the host bus adapter discovery list. The `DiscoveryProperty.sendTargetsDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.
 - `RemoveInternetScsiStaticTargets` – Remove static target entries from the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.

iSCSI initiators and targets have unique, permanent iSCSI names and addresses. An iSCSI name correctly identifies a specific iSCSI initiator or target, regardless of physical location. Names must be in EUI or IQN format, as specified by the storage vendor's hardware.

Configure the VMkernel To Support Software iSCSI

Before you can set up iSCSI on a system, you must create a dedicated VMkernel network interface. See [Network Configuration](#). You can then enable the VMkernel to support iSCSI and configure the initiator.

To enable the VMkernel to support software iSCSI, use the following steps.

Procedure

- 1 Obtain a managed object reference to the host system's `HostStorageSystem`.
- 2 Invoke the `UpdateSoftwareInternetScsiEnabled` method, passing the reference to the `HostStorageSystem` and the value `true`.

Configure iSCSI Initiators

After you have configured the VMkernel to support software iSCSI, you can configure the storage initiators for iSCSI.

To configure iSCSI initiators, use the following steps.

Procedure

- 1 Access the list of available HBAs on the host system.

You can do this by creating a property collector with `HostSystem` as the starting point. See [Chapter 6 Property Collector](#). From the `HostSystem.config` property, you can obtain the list (array) of host bus adapters by specifying this property path:

```
config.storageDevice.hostBusAdapter
```

The property path returns an array of host bus adapters. For example:

```
hostBusAdapter["key-vim.host.BlockHba-vmhba32"]
hostBusAdapter["key-vim.host.BlockHba-vmhba33"]
hostBusAdapter["key-vim.host.BlockHba-vmhba34"]
hostBusAdapter["key-vim.host.BlockHba-vmhba35"]
hostBusAdapter["key-vim.host.BlockHba-vmhba1"]
...
```

- 2 From the array, select the host bus adapter (instance of `HostHostBusAdapter`) that you want to configure and obtain its `key` property, which is the device name of the host bus adapter as a string.
- 3 Determine the capabilities of the adapter by retrieving the properties of the `HostHostBusAdapter` object.
- 4 Configure the initiator.
 - For an independent hardware initiator, configure the IP address.
 - For a software initiator, enable the software initiator in the VMkernel.
- 5 Configure the iSCSI name by calling `HostStorageSystem.UpdateInternetScisiName` and the alias by running `HostStorageSystem.UpdateInternetScisiAlias`.
- 6 Configure target discovery by calling `HostStorageSystem.UpdateInternetScisiHbaDiscoveryProperties`.

The method takes a `HostInternetScisiHbaDiscoveryProperties` data object that you can configure.

- 7 (Optional) Set the authentication information by calling `HostStorageSystem.UpdateInternetScisiAuthenticationProperties`.

The `HostInternetScisiHbaAuthenticationProperties` object you pass into that method includes properties for configuring CHAP and Mutual CHAP. See the *vSphere Storage* documentation for information about securing your iSCSI storage array.

- 8 Configure access to the targets.

9 Rescan the HBAs.

Rescan enables the HBAs to discover the new storage devices. You can either rescan a single HBA with `HostStorageSystem.RescanHba`, specifying the HBA ID as a parameter, or rescan all HBAs using `HostStorageSystem.RescanAllHba`.

Creating and Managing Datastores

Each datastore is a logical container, analogous to a file system on a logical volume, where the host places virtual disk files and other virtual machine files. Datastores hide specifics of the physical storage device and provide a uniform model for storing virtual machine files.

The `HostDatastoreSystem` managed object provides methods for creating and managing datastores. All `HostDatastoreSystem` methods require a managed object reference to `HostDatastoreSystem`, and return a reference to the `Datastore` object after it is created.

`HostDatastoreSystem` allows you to create and expand, query, and remove or update datastores. `HostDatastoreSystem` also allows you to configure a datastore principal for a host by calling `ConfigureDatastorePrincipal`. All virtual machine-related file I/O is performed under this user.

VMFS provisioning tasks are often performed as follows:

- 1 Call `QueryAvailableDisksForVmfs` to get the subset of disks that are well suited for holding VMFS datastores.

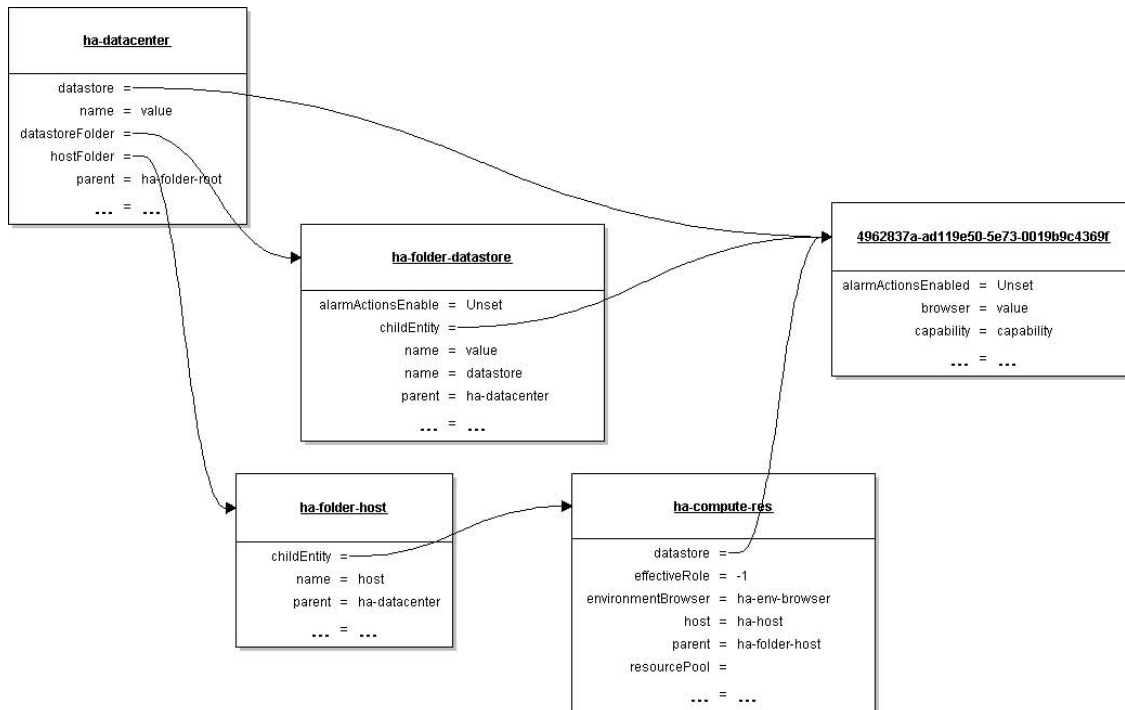
`QueryAvailableDisksForVmfs` obtains a list of disks that can be used to contain VMFS datastore extents. You can provide a datastore name to obtain the list of disks that can contain extents for the specified VMFS datastore. The operation does not return disks currently used by the VMFS datastore, nor does it return management LUNs and disks that are referenced by RDMS. RDM disks are not usable for VMFS datastores.
- 2 Get information about provisioning options by calling one of the following methods, passing in the selected disk:
 - `QueryVmfsDatastoreCreateOptions` – Obtains information about options for creating a new VMFS datastore on a disk. The method returns an array of `VmfsDatastoreOption` data objects.
 - `QueryVmfsDatastoreExpandOptions` – Obtains information about options for expanding the extents of an existing VMFS datastore.
 - `QueryVmfsDatastoreExtendOptions` – Obtains information about options for extending an existing VMFS datastore for a disk.
- 3 If required, change the layout by calling `HostStorageSystem.ComputeDiskPartitionInfo` and then `HostStorageSystem.UpdateDiskPartition` to resize the partition.
- 4 Call `CreateVmfsDatastore`, `ExtendVmfsDatastore`, or `ExpandVmfsDatastore` to complete the VMFS provisioning operation.

Accessing Datastores

The following diagram illustrates how you can access or specify datastores.

- Each `Datacenter` managed object has a `datastore` property that contains an array of datastores.
- Each `Datacenter` managed object has a `datastoreFolder` property that is a reference to the folder (or folder hierarchy) that contains the datastores for this datacenter.
- Each `Datacenter` managed object has a `hostFolder` property that is a reference to the folder (or folder hierarchy) that contains the compute resources, including hosts and clusters, for this datacenter. Each `HostSystem` or `ComputeResource` has a `datastore` property that is an array of `Datastore` managed objects.

Figure 9-4. Datastore Managed Object



For more information about the hierarchy of managed objects, see [Chapter 5 Datacenter Inventory](#).

Creating and Modifying a VMFS Datastore

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on.

VMFS is a proprietary file system VMware designed for virtual machines. VMFS is well suited for storing a small number of large data files like virtual disks. These files are mostly used by a single host. VMFS differs from other filesystem formats like FAT16/FAT32 and so on in that it can be accessed by multiple hosts connected to the same SAN LUN.

You can set up a VMFS datastore on any SCSI-based storage device that the host can access. VMFS volume creation, extension, and expansion requires first partitioning operations and the VMFS volume operations.

Set Up Disk Partitions

The Web Services API enables you to manage disk partitions on ESXi host storage devices.

To set up the disk partitions, use the following steps.

Procedure

- 1 Call `HostStorageSystem.RetrieveDiskPartitionInfo` to retrieve information about existing partitions.
- 2 Call `HostStorageSystem.ComputeDiskPartition`, passing in the desired disk layout. The server computes a new partition information object for a specific disk representing the desired layout and returns a `HostDiskPartitionInfo` object that you can use in the `HostDiskPartitionSpec` you pass into `UpdateDiskPartitions`.
- 3 Call `HostStorageSystem.UpdateDiskPartitions` to update partitions by passing in a `HostDiskPartitionSpec`.

Create the VMFS Datastore

After you create disk partitions on host storage, you can configure VMFS datastores.

To create a VMFS datastore, use the following steps.

Procedure

- 1 Configure and install any third-party adapter your storage requires and rescan the adapters by calling `HostStorageSystem.RescanAllHba`.
- 2 Call `HostDatastoreSystem.QueryAvailableDisksForVmfs` for information about disks that can be used to contain VMFS datastore.

This method filters out disks that are currently in use by an existing VMFS unless the VMFS using the disk is one being extended. It will also filter out management LUNs and disks that are referenced by RDMS. These disk LUNs are also unsuited for use by a VMFS. The method returns an array of `HostScisiDisk` objects.

- 3 Call `HostDatastoreSystem.QueryVmfsDatastoreCreateOptions` for information about options for creating a new VMFS datastore. The call returns an array of `VmfsDatastoreCreateOption` data objects that allow you to access the UUIDs of suitable data stores.
- 4 (Optional) If no suitable partitions for your VMFS volume exist, you might have to create them. Use the `ComputeDiskPartitionInfo` and `UpdateDiskPartitions` methods in `HostStorageSystem`.

5 Create the datastore.

- Call `HostDatastoreSystem.CreateVmfsDatastore` to create a VMFS datastore. The method takes a `VmfsDatastoreCreateSpec` data object that consists of a partition, a `HostVmfsSpec`, and an optional extent. The `HostVmfsSpec` allows you to specify the block size, extent, major version, and volume name for the VMFS.
- Call `HostDatastoreSystem.CreateNasDatastore` to create a network-attached storage based datastore.

Results

You can later expand and extend the VMFS datastore by calling one of the following methods.

- Call first `QueryVmfsDatastoreExpandOptions` and then `ExpandVmfsDatastore` to expand an existing VMFS datastore using the specification provided in the `VmfsDatastoreExpandSpec` data object (which contains the name of the extent and partition information). `ExpandVmfsDatastore` increases the size of the datastore up to the full size provisioned for the datastore, if necessary.
- Call first `QueryVmfsDatastoreExtendOptions` and then `ExtendVmfsDatastore` to extend an existing VMFS datastore using the specification provided in the `VmfsDatastoreExtendSpec` data object.

Removing and Updating Datastores

The Web Services API has methods to update and remove datastores.

- `RemoveDatastore` – Removes a datastore from a host.
- `UpdateLocalSwapDatastore` – Choose the `localSwapDatastore` for this host. Any change to this setting affects virtual machines that subsequently power on or resume from a suspended state at this host, or that migrate to this host while powered on. Virtual machines that are currently powered on at this host are not affected.

See the *vSphere API Reference* for more information about the `HostDatastoreSystem` operations, including constraints and limitations.

Managing VMFS Datastores with HostStorageSystem

In most cases, the `Datastore` methods are appropriate for creating and managing VMFS datastores. However, in some cases the following `HostStorageSystem` commands are used instead:

- `AttachVmfsExtent` – Extends a VMFS by attaching a disk partition as an extent.
- `ExpandVmfsExtent` – Expands a VMFS extent as specified by the disk partition specification.

- `FormatVmfs` – Formats a new VMFS on a disk partition based on the `HostVmfsSpec` that you pass in. Returns a `HostVmfsVolume` that represents the new VMFS file system. The `HostVmfsVolume` includes the block size, list of partition names of the disk's VMFS extents, and other information including the UUID.

This command is a low-level API you can use to partition disks explicitly. In most cases, the Datastore VMFS commands are more suitable.

- `RescanVmfs` – Rescans for new VMFS instances.
- `UpgradeVmfs` – Upgrades the VMFS to the current VMFS version.

Update and Upgrade with HostStorageSystem

You can use the `HostStorageSystem` managed object to alter characteristics of storage units.

- `HostStorageSystem.UpdateScsiLunDisplayName` – Update the mutable display name associated with a SCSI LUN. The SCSI LUN to be updated is identified using the LUN UUID.
- `HostStorageSystem.UpgradeVmLayout` – Iterates over all registered virtual machines. For each virtual machine, upgrades the layout and logs an event. After the method has been called, the information in the `VirtualMachineFileLayout` data object is correct.

Managing VMFS Volume Copies (Resignaturing)

By default, ESX/ESXi hosts mount all VMFS datastores. Each VMFS datastore that is created in a partition on a LUN has a unique UUID that is stored in the file system superblock. In addition, the LUN ID of the source LUN is unique and is stored in the VMFS metadata.

When a LUN is replicated or a copy is made, the resulting LUN copy is identical, byte-for-byte, with the original LUN. As a result, if the original LUN contains a VMFS datastore with UUID X, the LUN copy appears to contain an identical VMFS datastore, or a VMFS datastore copy, with exactly the same UUID X. ESX/ESXi can determine whether a LUN contains the VMFS datastore copy, and considers the copy unresolved and does not mount it automatically.

To make the data on the LUN copy available, you can either force mount the copy if you are sure the original is not in use, or you can resignature the copy. When you perform datastore resignaturing, consider the following points:

- Datastore resignaturing is irreversible because it overwrites the original VMFS UUID.
- The LUN copy that contains the VMFS datastore that you resignature is no longer treated as a LUN copy, but instead appears as an independent datastore with no relation to the source of the copy.
- A spanned datastore can be resignatured only if all its extents are online.
- The resignaturing process is crash and fault tolerant. If the process is interrupted, you can resume it later.

- You can mount the new VMFS datastore without a risk of its UUID colliding with UUIDs of any other datastore, such as an ancestor or child in a hierarchy of LUN snapshots.

See the *vSphere Storage* documentation for additional information.

Resignaturing Volumes with `ResignatureUnresolvedVmfsVolume_Task`

The easiest way to resignature unresolved volumes is by using the `HostDatastoreSystem.ResignatureUnresolvedVmfsVolume_Task` method. The method assigns a new `DiskUuid` to a VMFS volume, but keep its contents intact. The method supports safe volume sharing across hosts and is appropriate in most cases.

You can instead use the low-level `HostStorageSystem` methods to find, force mount, or unmount unresolved volumes:

- `HostStorageSystem.QueryUnresolvedVmfsVolume` – Obtains the list of unbound VMFS volumes. For sharing a volume across hosts, a VMFS volume is bound to its underlying block device storage. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unbound.
- `HostStorageSystem.ResolveMultipleUnresolvedVmfsVolumes` – Resignatures or force mounts unbound VMFS volumes. This method takes a `HostUnresolvedVmfsResolutionSpec` data object as input. The `HostUnresolvedVmfsResolutionSpec.resolutionSpec` property is an array of `HostUnresolvedVmfsResolutionSpec` data objects that contain a `HostUnresolvedVmfsResolutionSpecVmfsUuidResolution` enumeration. The enumeration is either `forceMount` or `resignature`.
- `UnmountForceMountedVmfsVolume` – Unmounts a force mounted VMFS volume. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unresolved. For the VMFS volume to be usable, a resolution operation is applied. As part of resolution operation, you might decide to keep the original VMFS UUID. Once the resolution is applied, the VMFS volume is mounted on the host for its use. This method allows you to unmount the VMFS volume if it is not used by any registered virtual machines.

Managing Diagnostic Partitions

Your host must have a diagnostic partition (dump partition) to store core dumps for debugging and for use by VMware technical support.

For more information about diagnostic bundles, see [Generating Diagnostic Bundles](#). For information about how to collect diagnostic partitions for a purple screen fault in ESXi, see the VMware knowledge base article at <http://kb.vmware.com/kb/1004128>.

A 100MB diagnostic partition for each host is recommended. If more than one ESX/ESXi host uses the same LUN as the diagnostic partition, that LUN must be zoned so that all the ESX/ESXi host can access it. Each host needs 100MB of space, so the size of the LUN determines how many servers can share it. Each ESX/ESXi host is mapped to a diagnostic slot. VMware recommends at least 16 slots (1600MB) of disk space if servers share a diagnostic partition. You can set up a SAN LUN with FibreChannel or hardware iSCSI. SAN LUNs accessed through a software iSCSI initiator are not supported.

Caution If two hosts that share a diagnostic partition fail and save core dumps to the same slot, the core dumps might be lost. To collect core dump data, reboot a host and extract log files immediately after the host fails. If another host fails before you collect the diagnostic data of the first host, the second host does not save the core dump.

Retrieving Diagnostic Partition Information

The `HostDiagnosticSystem` managed object allows you to retrieve information in several ways.

- Retrieve the `HostDiagnosticPartition` object from the `HostDiagnosticSystem.activePartition` property to examine the properties of the active partition.
- Call the `HostDiagnosticPartition.QueryAvailablePartition` method to retrieve a list of available diagnostic partitions, in order of suitability.
- Call the `HostDiagnosticPartition.QueryPartitionCreateOptions` method to retrieve a list of disks with sufficient space to contain a diagnostic partition of the specified type. The choices are returned in order of suitability.

Create a Diagnostic Partition

Creating a diagnostic partition requires that you find a suitable partition using one of the query methods. You can then retrieve a creation specification, and perform the actual creation.

To create a diagnostic partition, use the following steps.

Procedure

- 1 Find a suitable partition by calling `HostDiagnosticPartition.QueryAvailablePartition` or `HostDiagnosticPartition.QueryPartitionCreateOptions`.
- 2 Call `HostDiagnosticPartition.CreateDiagnosticPartition`, passing in a `HostDiagnosticPartitionCreateSpec`, which includes information about the diagnostic type, id, storage type, and so on.

Results

On success, this method creates the partition and makes the partition the active partition if specified in the `active` parameter. On failure, the diagnostic partition might exist, but will not be active even if the partition was supposed to be made active.

Sample Code Reference

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java (SDK\vsphere- ws\java\JAXWS\samples\com\vmware\)
--

scsilun\SCSILunName.java

httpfileaccess\GetVMFiles.java

Before you add storage and virtual machines to an ESXi system, you should have completed networking setup. This chapter describes how to set up virtual switches in the vSphere environment.

This chapter includes the following topics:

- [Virtual Switches](#)
- [Using a Distributed Virtual Switch](#)
- [VMware Standard Virtual Switch](#)
- [Sample Code Reference](#)

Virtual Switches

vSphere supports the use of virtual switches to manage network traffic to and from virtual machines.

- vCenter Server supports a distributed network model in which a distributed virtual switch manages ESXi host proxy switch configuration. In the distributed network model, a host proxy switch reflects the distributed virtual switch port settings, describes how physical network adapters are bridged to the switch, and performs network I/O.
- On a standalone ESXi host, you can use a VMware standard virtual switch to support network traffic to and from virtual machines on the host.

To configure a vSphere network you perform the following operations:

- Set up virtual switches
- Define portgroups
- Configure physical network adapters

Port Groups

Port groups aggregate multiple ports under a common configuration. Each port can connect to a network adapter of a virtual machine, or an uplink adapter on the physical machine.

Each port group is identified by a network label, which is unique to the current host. Network labels make virtual machine configuration portable across hosts. All port groups in a datacenter that are physically connected to the same network (in the sense that each can receive broadcasts from the others) are given the same label. Conversely, if two port groups cannot receive broadcasts from each other, they have distinct labels.

You can use a VLAN ID to restrict port group traffic to a logical Ethernet segment within the physical network. For a port group to reach port groups located on other VLANs, the VLAN ID must be set to 4095. If you use VLAN IDs, you must change the port group labels and VLAN IDs together so that the labels properly represent connectivity.

Virtual Machine Network Interfaces

When you create a virtual machine, you include a `VirtualMachineConfigSpec`, which, in turn, includes a `VirtualDeviceConfigSpec`. The `device` property of `VirtualDeviceConfigSpec` is a `VirtualDevice` data object.

One of the available virtual devices is `VirtualEthernetCard`. You can use one of the subtypes of `VirtualEthernetCard` to specify the virtual card to use and to specify the MAC address and whether wake-on-LAN is enabled for this virtual card. See [Adding Devices to Virtual Machines](#). A limited number of adapters is supported. KB article 1001805 (<http://kb.vmware.com/kb/1001805>) discusses available network adapters and which adapter is appropriate in which situation.

VMkernel Network Interfaces

The network services that the VMkernel provides (iSCSI, NFS, and VMotion) use a TCP/IP stack in the VMkernel. This stack accesses various networks by attaching to one or more port groups on one or more virtual switches.

The VMware VMkernel TCP/IP networking stack handles iSCSI, NFS, and VMotion in the following ways.

- iSCSI as a virtual machine datastore
- iSCSI for the direct mounting of ISO files, which are presented as CD-ROMs to virtual machines
- NFS as a virtual machine datastore
- NFS for the direct mounting of ISO files, which are presented as CD-ROMs to virtual machines
- Migration with VMotion

If you have two or more physical NICs for iSCSI, you can create multiple paths for the software iSCSI by using port binding. For more information on port binding, see the *iSCSI SAN Configuration Guide*.

A freshly installed ESX/ESXi system does not include VMkernel network interfaces. When you wish to migrate a virtual machine with VMotion, your VMkernel networking stack must be set up properly. When you want to use storage types that use TCP/IP network communications, such as iSCSI, you must provide a separate VMkernel network interface for that storage device. You must create any VMkernel ports you might need (see [Adding a VMkernel Network Interface](#)).

Physical Network Adapter (pnict)

The term pnict refers to the physical network adapters as seen by the primary operating system. When using the vSphere Web Services SDK, you can manipulate the adapter directly. When using the vSphere Client GUI, you manipulate instead the uplink adapter. On an ESXi host, each pnict has one associated uplink adapter.

In a vDS environment, you use a DVS uplink instead of an uplink adapter.

Using a Distributed Virtual Switch

A `DistributedVirtualSwitch` managed object is a virtual network switch that is located on a vCenter Server. A distributed virtual switch manages configuration for proxy switches (`HostProxySwitch`). A proxy switch is located on an ESXi host that is managed by the vCenter Server and is a member of the switch. A distributed switch also provides virtual port state management so that port state is maintained when vCenter Server operations move a virtual machine from one host to another.

A proxy switch performs network I/O to support the following network traffic and operations:

- Network traffic between virtual machines on any hosts that are members of the distributed virtual switch.
- Network traffic between a virtual machine that uses a distributed virtual switch and a virtual machine that uses a VMware standard virtual switch.
- Network traffic between a virtual machine and a remote system on a physical network connected to the ESXi host.
- vSphere system operations to support capabilities such as VMotion or High Availability.

A `DistributedVirtualSwitch` is the base distributed switch implementation. It supports a VMware distributed virtual switch implementation and it supports third party distributed switch implementations. The base implementation provides the following capabilities (defined in the `DVSFeatureCapability` object):

- NIC teaming
- Network I/O control
- Network resource allocation
- Quality of service tag support
- User-defined resource pools

- I/O passthrough (VMDirectPath Gen2)

A `VmwareDistributedVirtualSwitch` supports the following additional capabilities (defined in the `DVSFeatureCapability` and `VMwareDVSFeatureCapability` objects):

- Backup, restore, and rollback for a VMware distributed virtual switch and its associated portgroups.
- Maximum Transmission Unit (MTU) configuration.
- Health check operations for NIC teaming and VLAN/MTU support.
- Monitoring switch traffic using Internet Protocol Flow Information Export (IPFIX).
- Link Layer Discovery Protocol (LLDP).
- Virtual network segmentation using a Private VLAN (PVLAN).
- VLAN-based SPAN (VSPAN) for virtual distributed port mirroring.
- Link Aggregation Control Protocol (LACP) defined for uplink portgroups.

Distributed Virtual Switch Configuration

To use a distributed virtual switch, you create a switch and portgroups on a vCenter Server, and add hosts as members of the switch.

Procedure

- 1 Use the `Folder.CreateDVS_Task` method to create a distributed virtual switch. Use a `DVSConfigSpec` to create a switch for a third-party implementation. Use a `VMwareDVSConfigSpec` to create a VMware distributed virtual switch.
- 2 Use the `CreateDVPortgroup_Task` method to create portgroups for host and virtual machine network connections and for the connection between proxy switches and physical NICs. A `DistributedVirtualPortgroup` specifies how virtual ports (`DistributedVirtualPort`) will be used. When you create a distributed virtual switch, the vCenter Server automatically creates one uplink portgroup (`config.uplinkPortgroup`). Uplink portgroups are distributed virtual portgroups that support the connection between proxy switches and physical NICs.

Port creation on a distributed switch is determined by the portgroup type (`DVPortgroupConfigSpec.type`):

- If a portgroup is early binding (static), then `DVPortgroupConfigSpec.numPorts` determines the number of ports that get created when the portgroup is created. This number can be increased if `DVPortgroupConfigSpec.autoExpand` is true.
- If a portgroup is ephemeral (dynamic), then `numPorts` is ignored and ports are created as needed.

You can also specify standalone ports that are not associated with a port group and uplink ports that are created on ESXi hosts (`DVSConfigSpec.numStandalonePorts`).

The `DVPortgroupConfigInfo.numPorts` property is the total number of ports for a distributed virtual switch. This total includes the ports generated by the static and dynamic portgroups and the standalone ports.

- 3 If you have created additional uplink portgroups, use the `ReconfigureDvs_Task` method to add the portgroup(s) to the `DVConfigSpec.uplinkPortgroup` array.
- 4 Retrieve physical NIC device names from the host (`HostSystem.config.network.pnic[].device`).
- 5 Add host member(s) to the distributed virtual switch. To configure host members:
 - Specify hosts (`DVConfigSpec.host[]`).
 - For each host, specify one or more physical NIC device names to identify the pNIC(s) for the host proxy connection to the network (`DistributedVirtualSwitchHostMemberConfigSpec.backing.pnicSpec[].pnicDevice`).
 - Use the `DistributedVirtualSwitch.ReconfigureDvs_Task` method to update the switch configuration.

When you add a host to a distributed virtual switch

(`DistributedVirtualSwitch.config.host`), the host automatically creates a proxy switch. The proxy switch is removed automatically when the host is removed from the distributed virtual switch.

- 6 Connect hosts and virtual machines to the distributed virtual switch.

Host connection	Specify port or portgroup connections in the host virtual NIC spec (<code>HostVirtualNicSpec.distributedVirtualPort</code> or <code>HostVirtualNicSpec.portgroup</code>).
Virtual machine connection	Specify port or portgroup connections in the distributed virtual port backing (<code>VirtualEthernetCardDistributedVirtualPortBackingInfo</code>) for the virtual Ethernet cards on the virtual machine (<code>VirtualEthernetCard.backing</code>).

Backup, Rollback, and Query Operations

If you are using a `VmwareDistributedVirtualSwitch`, you can perform backup and rollback operations on the switch and its associated distributed virtual portgroups.

When you reconfigure a VMware distributed virtual switch (`ReconfigureDvs_Task`), the server saves the current switch configuration before applying the configuration updates. The saved switch configuration includes portgroup configuration data. The server uses the saved switch configuration as a checkpoint for rollback operations. You can rollback the switch or portgroup configuration to the saved configuration, or you can rollback to a backup configuration (`EntityBackupConfig`).

- To backup the switch and portgroup configuration, use the `DistributedVirtualSwitchManager.DVSManagerExportEntity_Task` method. The export method produces a `EntityBackupConfig` object. The backup configuration contains the switch and/or portgroups specified in the `SelectionSet` parameter. To backup the complete configuration you must select the distributed virtual switch and all of its portgroups.
- To rollback the switch configuration, use the `DVSRollback_Task` method to determine if the switch configuration has changed. If it has changed, use the `ReconfigureDvs_Task` method to complete the rollback operation.
- To rollback the portgroup configuration, use the `DistributedVirtualPortgroup.DVPortgroupRollback_Task` method to determine if the portgroup configuration has changed. If it has changed, use the `ReconfigureDVPortgroup_Task` method to complete the rollback operation.

To perform query operations on a distributed virtual switch, use the `DistributedVirtualSwitchManager` methods.

VMware Standard Virtual Switch

Network setup for ESXi hosts can consist of several parts:

- Setting up one or more virtual switches. Virtual switches provide the connectivity between virtual machines on the same host or on different hosts. Virtual switches also support VMkernel network access for VMotion, iSCSI, and NFS. You set up virtual switches independently on each host. See [Adding a Standard Virtual Switch](#).
- Adding virtual machine port groups. A virtual machine always accesses the network through a port group. See [Adding a Virtual Port Group](#).
- Specifying the adapter for the virtual machine. This adapter is specified as a virtual device, configured as part of virtual machine setup, and discussed in [Configuring a Virtual Machine](#).
- Adding VMkernel network interfaces, for example, to support iSCSI storage or VMotion. See [Adding a VMkernel Network Interface](#).
- Configuring a physical adapter (pnict), the actual connection from the host to the network. You can configure the pnict through the `HostNetworkSystem.pnict` property, which is a `PhysicalNic` data object. You can specify the set of pnicts associated with a virtual switch through the `VirtualSwitch.pnict` property, which takes an array of physical network adapters.

- Network configuration for the host (IP routing, DNS, SNMP). See [Adding Networking Services](#).

Configuring a Standard Virtual Switch

To use a VMware standard virtual switch, you use the following elements to configure the switch on an ESXi host.

- `HostNetworkSystem` – Managed object that represents the host's networking configuration. This object's properties point to the networking data objects you can use for network management, including `HostDnsConfig` and `HostIpRouteConfig`.

`HostNetworkSystem` properties allow you to access `HostNetCapabilities` and `HostNetworkInfo` data objects, and access and modify the `HostNetworkConfig` data object.

`HostNetworkSystem` includes methods for retrieving and changing the network configuration. See the *API Reference* for a complete list of methods and the permissions required to run them.

- `HostNetworkConfig` – Allows you to specify the network configuration for the host. You can apply the configuration by running the `HostNetworkSystem.UpdateNetworkConfig` method.
- `Network` – Represents a network accessible by either hosts or virtual machines. This can be a physical network or a logical network, such as a VLAN.

When you add a host to a vCenter Server system, or when you add a virtual machine to an ESX/ESXi host, a `Network` is added automatically.
- `HostSystem.QueryHostConnectionInfo` and `Datacenter.QueryConnectionInfo` both return a `HostConnectInfo` data object, which describes the current network configuration.

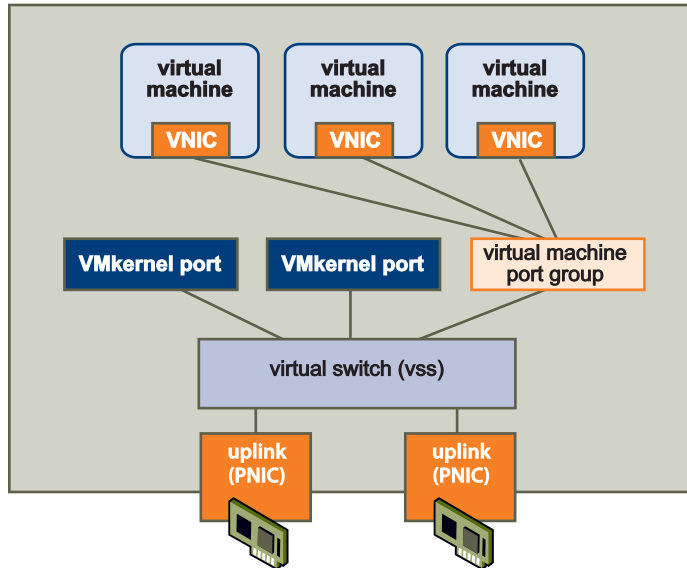
`HostSNMPSystem` – Supports SNMP setup. See [Setting Up SNMP](#).

vNetwork Standard Switch Environment

A vNetwork Standard Switch (vSS) can route traffic internally between virtual machines and can link virtual machines to external networks.

The following diagram shows the elements of a vSS environment.

Figure 10-1. vSS Environment



Virtual Switches

At the center of networking with vSS is the virtual switch itself. The vSS can send network traffic between virtual machines on the same host (private network) or network traffic to an external network (public network). The public network uses the Ethernet adapter associated with the physical host (uplink adapter).

When two or more virtual machines are connected to the same vSS, network traffic between them is routed locally. If an uplink adapter is attached to the vSS, each virtual machine can access the external network that the adapter is connected to.

Setting Up Networking with vSS

You can use the `HostNetworkSystem` managed object to access and manipulate the elements of an ESXi network.

Retrieving Information About the Network Configuration

You can retrieve information about the network configuration as follows:

- The properties of the `HostNetworkConfig` object, which you access through `HostNetworkSystem.networkConfig`, allow you to retrieve configuration information. This information is comprehensive and includes the physical adapters, virtual switches, virtual network interfaces, and so on.

You can also use `HostNetworkConfig` to make changes to the configuration.

- The properties of the `HostNetworkInfo` object, which you access through `HostNetworkSystem.networkInfo`, allow you to retrieve runtime information.

Adding a Standard Virtual Switch

You call the `HostNetworkSystem.AddVirtualSwitch` method to add one or more virtual switches. Pass in the name of the virtual switch and a `HostVirtualSwitchSpec` data object as parameters.

Inside `HostVirtualSwitchSpec` you can specify the MTU, number of ports, network policy, and bridge specification. The bridge specifies how the virtual switch connects to the physical adapter. The currently supported bond bridge provides network adapter (NIC) teaming capabilities through the use of a list of physical devices and, optionally, a beacon probe to test connectivity with physical adapters.

After you have created the virtual switch, you can connect it to a `pnic` for connection to the outside, and to a `VMkernel` port or a port group.

To add a virtual switch, use the following steps.

Procedure

- 1 Obtain information about the current networking configuration.

You can use a property collector to retrieve the `HostNetworkSystem` managed object and several of its properties, such as `networkInfo`.

- 2 Define a `HostVirtualSwitchSpec` that specifies the attributes of the virtual switch. You can specify the number of ports (56 to 4088 on ESXi systems) and the `HostNetworkPolicy`. See “Defining the Host Network Policies” on page 122.

- 3 Call `HostNetworkSystem.AddVirtualSwitch` to add a virtual switch. Specify a unique name and a `HostVirtualSwitchSpec` that defines the switch attributes.

The following fragment from `AddVirtualSwitch.java` illustrates this.

Example: Adding a Virtual Switch

```
vswitchId = vSwitch42;
...
ManagedObjectReference nwSystem = configMgr.getNetworkSystem();
HostVirtualSwitchSpec spec = new HostVirtualSwitchSpec();
spec.setNumPorts(8);
service.addVirtualSwitch(nwSystem, vswitchId, spec);
System.out.println( " : Successful creating : " + vswitchId);
```

Adding a Virtual Port Group

Port groups allow you to differentiate between different kinds of traffic passing through a virtual switch. You can also use port groups as a boundary for communication or for security policy configuration. The default port groups for ESXi systems are `Management Network` and `VM Network`.

When you create a port group, you can specify a VLAN ID for it. VLANs are an important part of ESXi networking because they allow you to group traffic. For example, you could create separate network segments for VMotion, for management and for development. Using VLANs, you only need to have a separate uplink adapter for each network segment and a single virtual switch connecting to that adapter. That setup can greatly reduce the number of switches you need.

To add a virtual port group, use the following steps.

Procedure

- 1 Define a `HostPortgroupSpec`. For each port group, you can specify the network policy, the VLAN ID, and the virtual switch to which the port group belongs.
- 2 Call `HostNetworkSystem.AddPortGroup`, passing in the `PortGroupSpec`.

Adding a VMkernel Network Interface

VMkernel network interfaces provide the network access for the VMkernel TCP/IP stack. You must create new VMkernel ports for your ESXi system if you plan on using VMotion, VMware FT, or iSCSI and NAS storage. A VMkernel port consists of a port on the virtual switch and a VMkernel interface.

To add a VMkernel Network Interface to your ESXi system, use the following steps.

Procedure

- 1 Create a `HostVirtualNicSpec` data object. Inside the object, you can specify the IP configuration in a `HostIpConfig` data object. For vSS, specify the `portgroup` property. For vDS, specify the `distributedVirtualPort` property.
- 2 Call `HostNetworkSystem.AddVirtualNic`, passing in the `HostVirtualNicSpec`.
- 3 You can then use the VMkernel network interface for software iSCSI or NAS, or call the `HostVmotionSystem.SelectVnic` method to use this VMkernel NIC for VMotion.

[Adding a VMkernel Network Interface](#), a code fragment from the `AddVirtualNic` example, illustrates this. The sample retrieves the IP address from the command line using the `cb.get_option` call.

Example: Adding a VMkernel Network Interface

```
private HostVirtualNicSpec createVNicSpecification() {
    HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
    HostIpConfig ipConfig = new HostIpConfig();
    ipConfig.setDhcp(false);
    ipAddr = cb.get_option("ipaddress");
    ipConfig.setIpAddress(ipAddr);
    ipConfig.setSubnetMask("255.255.255.0");
    vNicSpec.setIp(ipConfig);
    return vNicSpec;
}

....
```

```
HostVirtualNicSpec vNicSpec = createVNicSpecification();
service.addVirtualNic(nwSystem, portGroup, vNicSpec);
```

Defining the Host Network Policies

When you configure host networks, you can define specific policies for the network. The `HostNetworkPolicy` data object type describes network policies for both virtual switches and port groups. If the settings are not specified for the port group explicitly, the port group inherits policy settings from the virtual switch with which it is associated.

The policies are defined by the following data objects available as properties of `HostNetworkPolicy`.

- `HostNicTeamingPolicy` – Defines the connection to the physical network. This includes failure criteria, active and standby NICs, and failover and load balancing information. See [NIC Teaming](#).
- `HostNetworkSecurityPolicy` – Defines the security policies for the network. See the *ESXi Configuration Guide*.
- `HostNetworkTrafficShapingPolicy` – Establishes parameters for three traffic characteristics: average bandwidth, peak bandwidth, and maximum burst size.

You can also specify the VLAN policy by assigning an integer to the `HostPortgroupSpec.vlanid` property. The VMkernel takes care of tagging and untagging the packets as they pass through the virtual switch. See the `HostPortgroupSpec` and `HostNetworkPolicy` data objects in the *API Reference*.

NIC Teaming

Virtual machines connect to the public network through a virtual switch, which, in turn, connects to the physical network interface (pnic). When the physical adapter or the adapter's network connection fails, connectivity for the associated virtual switch and all port groups and virtual machines is lost.

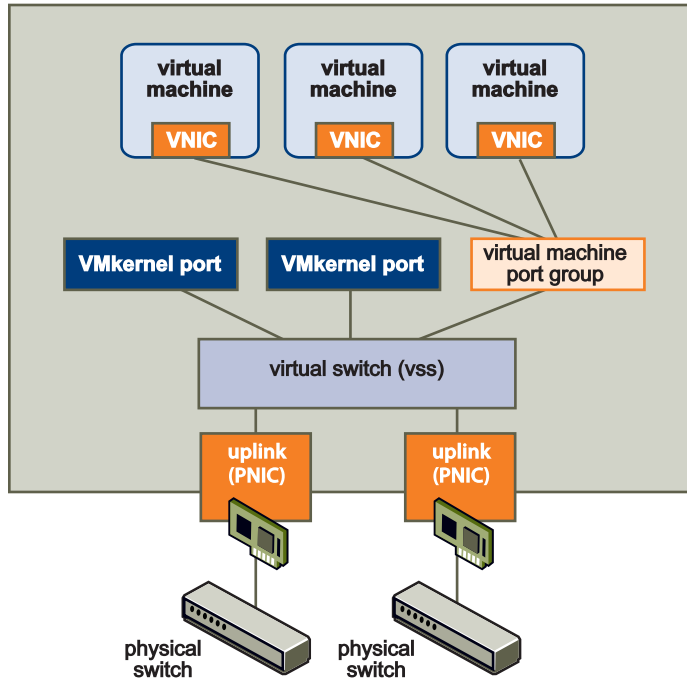
To resolve this issue, you can set up your environment so each virtual switch connects to two uplink adapters. Each uplink adapter connects to two different physical switches. The teams can then either share the load of traffic between physical and virtual networks among some or all of its members, or provide passive failover in the event of a hardware failure or a network outage.

You set up NIC teaming by setting the `HostNetworkPolicy`. The path to the `HostNicTeamingPolicy` is:

```
HostConfigSpec.network.vswitch[].spec.policy.nicTeaming
```

If you specify NIC teaming for a virtual switch, the `HostVirtualSwitchSpec.bridge` property must be set to `HostVirtualSwitchBondBridge`.

Figure 10-2. NIC Teaming



Setting Up IPv6 Networking

vSphere supports both Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) environments. With IPv6, you can use vSphere features such as NFS in an IPv6 environment.

An IPv6-specific configuration in vSphere involves providing IPv6 addresses, either by entering static addresses or by using DHCP for all relevant vSphere networking interfaces. IPv6 addresses can also be configured using stateless autoconfiguration sent by router advertisement.

You can set up IPv6 networking for a host by changing the `HostIpConfig.ipV6Config` property, which is a `HostIpConfigIpV6AddressConfiguration` data object.

`HostIpConfigIpV6AddressConfiguration` allows you to specify whether auto-configuration is enabled, whether DHCP for IPv6 addresses is enabled, and an array of IPv6 addresses (`HostIpConfigIpV6Address` data objects).

`HostIpConfigIpV6Address` allows you to specify all aspects of the IPv6 address including the state of the address, the address (unless DHCP is enabled), life time, operation, origin, and prefix length. See the *API Reference*. The following code fragment illustrates setting the VMkernel NIC to get an automatic IPv6 address from router advertisements and through DHCP. The user provides the IP address on the command line when calling the program from which the fragment is taken. The sample retrieves the address using the `cb.get_option` utility applications call.

Example: IPv6 Setup

```
private HostVirtualNicSpec createVNicSpecification() {
    HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
    HostIpConfig ipConfig = new HostIpConfig();
```

```
//setting the vnic to get an automatic ipv6 address from router advertisements
// and through dhcp

ipV6Config = new HostIpConfigIPv6AddressConfiguration();
ipV6Config.setAutoConfigurationEnabled(true);
ipV6Config.setDhcpV6Enabled(true);
ipConfig.setIpV6Config(ipV6Config);
vNicSpec.setIp(ipConfig);
return vNicSpec;

....
```

Adding Networking Services

You can set up network services for your ESXi system by using `HostConfigManager` properties and methods.

Adding an NTP Service

The `HostConfigManager.dateTimeSystem` property contains a `HostDateTimeSystem` data object. This object allows you to perform NTP and date and time related configuration.

- Query and update the date and time information by using one of the methods defined in `HostDateTimeSystem`.
- Modify the `HostDateTimeSystem.dateTimeInfo` property, which contains a `HostDateTimeInfo` object, to set up NTP. The NTP information is stored in the `HostDateTimeInfo.ntpConfig` property, which is a `HostNtpConfig` object. The `HostNtpConfig` objects's `server` property contains a list of time servers, specified by IP address or fully qualified domain name.

Important You can start and stop the NTP daemon and retrieve information about it by using the `HostServiceSystem` object.

Setting Up the IP Route Configuration

You can use the `HostNetworkSystem.UpdateIPRouteConfig` method to specify the IP route configuration for an ESXi system. The method takes a `HostIPRouteConfig` data object as an argument. In this object, you can specify the default gateway address and the IPv6 gateway address.

Setting Up SNMP

Simple Network Management Protocol (SNMP) allows management programs to monitor and control networked devices. vCenter Server and ESXi systems include different SNMP agents:

- The SNMP agent included with vCenter Server can send traps when the vCenter Server system is started or when an alarm is triggered on vCenter Server. The vCenter Server SNMP agent functions only as a trap emitter and does not support other SNMP operations such as GET.

- ESXi 4.0 includes an SNMP agent embedded in the host daemon (`hostd`) that can send traps and receive polling requests such as `GET` requests.

Versions of ESX released before ESX/ESXi 4.0 included a Net-SNMP-based agent. You can continue to use this Net-SNMP-based agent in ESX 4.x with MIBs supplied by your hardware vendor and other third-party management applications. However, to use the VMware MIB files, you must use the embedded SNMP agent. To use the NET-SNMP based agent and embedded SNMP agent at the same time, make one of the agents listen on a nondefault port. By default, both agents use the same port.

The SDK supports SNMP agent configuration through the `HostSnpSystem` managed object. This object includes two methods, `ReconfigureSnpAgent` and `SendTestNotification`.

- `HostSnpSystem.ReconfigureSnpAgent` allows you to specify agent properties through a `HostSnpConfigSpec`. That data object allows you to specify the SNMP port, read only communities, and the trap targets in an `HostSnpDestination` object. The `HostSnpDestination` object allows you to specify the community, and a host and port listening for notification.
- `HostSnpSystem.SendTestNotification` allows you to test your configuration.

A `HostSnpSystemAgentLimits` data object in the `HostSnpSystem.limits` property specifies limits of the agent.

Sample Code Reference

The following table lists the sample applications included with the vSphere SDK that demonstrate how to use some of the managed objects discussed in this chapter.

Java
(SDK\vsphere- ws\java\JAXWS\samples\com\vmware\host)
AddVirtualNic.java
AddVirtualSwitch.java
AddVirtualSwitchPortGroup.java
RemoveVirtualNic.java
RemoveVirtualSwitch.java
RemoveVirtualSwitchPortGroup.java

Virtual Machine Configuration

11

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. Virtual machines are compatible with all standard x86 computers. Each virtual machine encapsulates a complete computing environment and runs independently of underlying hardware.

This chapter includes the following topics:

- [VirtualMachine Management Objects and Methods](#)
- [Creating Virtual Machines and Virtual Machine Templates](#)
- [Configuring a Virtual Machine](#)
- [Adding Devices to Virtual Machines](#)
- [Performing Virtual Machine Power Operations](#)
- [Registering and Unregistering Virtual Machines](#)
- [Customizing the Guest Operating System](#)
- [Installing VMware Tools](#)
- [Upgrading a Virtual Machine](#)

VirtualMachine Management Objects and Methods

Virtual machines are the central elements of your vSphere environment.

You create a virtual machine by calling `Folder.CreateVM_Task`, and configure the virtual machine by using properties and methods of the `VirtualMachine` managed object. Most of the properties point to data objects that the methods use as input. [Figure 11-1. VirtualMachine Managed Object with Some Properties and Methods](#) shows some of the properties and methods.

Client applications commonly access and manipulate the following virtual machine related objects:

- `VirtualMachine` – Managed object used for most virtual machine manipulation. Includes methods that create templates, clones, or snapshots of a virtual machine, perform power operations and guest OS management, and install VMware Tools.

- `VirtualMachineConfigInfo` – Data object which allows you to retrieve configuration-specific information from a virtual machine.
- `VirtualMachineCloneSpec` – Data object which allows you to specify virtual machine properties for a clone operation. Argument to `VirtualMachine.CloneVM_Task`.

Figure 11-1. VirtualMachine Managed Object with Some Properties and Methods

VirtualMachine	ManagedEntity
<code>capability : VirtualMachineCapability</code> <code>config : VirtualMachineConfigInfo</code> <code>datastore : MOR[] Datastore[]</code> <code>environmentBrowser : MOR EnvironmentBrowser</code> <code>guest : GuestInfo</code> <code>guestHeartbeatStatus : ManagedEntityStatus</code> <code>network : MOR[] Network[]</code> <code>resourceConfig : ResourceConfigSpec</code> <code>resourcePool : MOR ResourcePool</code> <code>runtime : VirtualMachineRuntimeInfo</code> <code>snapshot : VirtualMachineSnapshotInfo</code> <code>storage : VirtualMachineStorageInfo</code> <code>summary : VirtualMachineSummary</code>	<code>alarmActionsEnabled : boolean</code> <code>configIssue : Event []</code> <code>configStatus : ManagedEntityStatus</code> <code>customValue : CustomFieldValue []</code> <code>declaredAlarmState : AlarmState []</code> <code>disabledMethod : string []</code> <code>effectiveRole : int []</code> <code>name : string</code> <code>overallStatus : ManagedEntityStatus</code> <code>parent : MOR[] ManagedEntity</code> <code>permission : Permission []</code> <code>recentTask : MOR[] Task []</code> <code>tag : Tag []</code> <code>triggeredAlarmState : AlarmState []</code>
<code>AcquireMksTicket (_this) : VirtualMachineMksTicket</code> <code>AnswerVM (_this, questionId, answerChoice) : void</code> <code>CheckCustomizationSpec (_this, spec) : void</code> <code>CloneVM_Task (_this, folder, name, spec) : MOR Task</code> <code>CreateScreenshot_Task (_this) : MOR Task</code> <code>CreateSecondaryVM_Task (_this, host) : MOR Task</code> <code>CreateSnapshot_Task (_this, name, description, memory, quiesce) : MOR Task</code> <code>CustomizeVM_Task (_this, spec) : MOR Task</code> <code>DefragmentAllDisks (_this) : void</code> <code>DisableSecondaryVM_Task (_this, vm) : MOR Task</code> <code>EnableSecondaryVM_Task (_this, vm, host) : MOR Task</code> <code>ExportVm (_this) : MOR HttpNfcLease</code> <code>ExtractOvfEnvironment (_this) : string</code> <code>MakePrimaryVM_Task (_this, vm) : MOR Task</code> <code>MarkAsTemplate (_this) : void</code> <code>MarkAsVirtualMachine (_this, pool, host) : void</code> <code>...</code>	<code>Destroy_Task (_this) : MOR Task</code> <code>Reload (_this) : void</code> <code>Rename_Task (_this, newName) : MOR Task</code>

Creating Virtual Machines and Virtual Machine Templates

To create a virtual machine, you use the `Folder.CreateVM_Task` method. The method takes a `VirtualMachineConfigSpec` data object as input argument. `VirtualMachineConfigSpec` allows you to specify the attributes of the virtual machine you are creating.

If you need several identical virtual machines, you can convert an existing virtual machine to a template and create multiple copies (clones) from the template. You can also create multiple virtual machines by cloning an existing virtual machine directly.

Creating a Virtual Machine Using VirtualMachineConfigSpec

Use the `Folder.CreateVM_Task` method to create a virtual machine by specifying its attributes. You must specify either a host or a resource pool (or both). The virtual machine uses the CPU and memory resources from the host or resource pool.

Calling the CreateVM_Task Method

Create a virtual machine by calling the `Folder.CreateVM_Task` method with the following arguments:

- `_this`—Folder where you want to place the virtual machine.
- `config`—`VirtualMachineConfigSpec` data object that specifies CPU, memory, networking, and so on. See [Specifying Virtual Machine Attributes with VirtualMachineConfigSpec](#)
- `pool`—Resource pool for the virtual machine to draw resources from.
- `host`—`HostSystem` managed object that represents the target host on which to run the virtual machine. If you invoke this method on a standalone host, omit this parameter. If the target host is part of a VMware DRS cluster, this parameter is optional; if no host is specified, the system selects one.

Important All objects must be located in the same datacenter.

Specifying Virtual Machine Attributes with VirtualMachineConfigSpec

The actual customization of the virtual machine happens through the properties of the `VirtualMachineConfigSpec` that is passed in as an argument to `Folder.CreateVM_Task`. For example, you can specify the name, boot options, number of CPUs, and memory for the virtual machine. All properties of `VirtualMachineConfigSpec` are optional to support incremental changes. See the *API Reference*.

The following example fragment from the `VMCreate` sample program illustrates how to define a `VirtualMachineConfigSpec`.

Example: Defining a VirtualMachineConfigSpec Data Object

```
VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
...
vmConfigSpec.setName("MyVM");
vmConfigSpec.setMemoryMB(new Long(Integer.parseInt(500)));
vmConfigSpec.setNumCPUs(Integer.parseInt(4));
vmConfigSpec.setGuestId(cb.get_option("guestosid"));
...
```

The VMware SDK `SDK/samples/Axis/java/com/vmware/apputils/vim/VMUtils.java` sample defines a more comprehensive virtual machine that also includes a Floppy, CD-ROM, disk, and virtual NIC. See [Configuring a Virtual Machine](#) for a discussion of commonly set properties.

When you create a virtual machine, the virtual machine files are added at the virtual machine's storage location. See [Table 11-1. Default Files](#).

Additional Configuration Information

The `VirtualMachineConfigInfo` and `VirtualMachineConfigSpec` objects provide the `extraConfig` property for additional configuration information. The `extraConfig` property is an array of key/value pairs that identify configuration options. The Server stores the `extraConfig` options in the `.vmx` file for the virtual machine. As the vSphere API evolves from version to version, an `extraConfig` option may become a standard configuration property that is part of the defined inventory data model. In this case, you must use the standard data model property for access; you cannot use the `extraConfig` property.

Creating Virtual Machine Templates

Templates allow you to create multiple virtual machines with the same characteristics, such as resources allocated to CPU and memory, or type of virtual hardware. A virtual machine template is a virtual machine that cannot be powered on and that is not associated with a resource pool.

You can convert any powered off virtual machine to a template by calling `VirtualMachine.MarkAsTemplate`. After the conversion, the original virtual machine no longer exists. You can use the template to create multiple clones of the same configuration.

Cloning a Virtual Machine

A clone is a copy of a virtual machine. The main difference between a virtual machine and a clone is that the `VirtualMachine.config.template` property is set to `true`.

You can create a clone in one of the following ways:

- If you no longer need a specific instance of a virtual machine, but you want to use the virtual machine's configuration as a template, use the `VirtualMachine.MarkAsTemplate` method. This method sets the `config.template` property to `true`, and disables the virtual machine.
- If you want to use an existing virtual machine as a template, but keep the virtual machine, call the `VirtualMachine.CloneVM_Task` method to create a duplicate of the virtual machine.

If you use the `VirtualMachine.CloneVM_Task` method, you can customize certain attributes of the clone by specifying them in the `VirtualMachineCloneSpec` data object you pass in when you call the method.

The following code fragment from `VMClone.java` illustrates how you can customize a clone and specify a new location for it.

Example: Cloning a Virtual Machine

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineRelocateSpec relocSpec = new VirtualMachineRelocateSpec();
cloneSpec.setLocation(relocSpec);
cloneSpec.setPowerOn(false);
cloneSpec.setTemplate(false);

String clonedName = cloneName;
```

```
ManagedObjectReference cloneTask
    = service.cloneVM_Task(vmRef, vmFolderRef, clonedName, cloneSpec);
```

The `VirtualMachine.CloneVM_Task` method takes the source virtual machine, target folder, name, and `VirtualMachineCloneSpec` as arguments.

The `VirtualMachineCloneSpec` data object includes the location, power state, and whether the clone should be a template. The location, in turn, is a `VirtualMachineRelocateSpec` data object that specifies the target location (datastore, disk, and host or resource pool) and any transformation to be performed on the disk.

Converting a Template to a Virtual Machine

You can change a template back to an operational virtual machine.

- To convert the template to a virtual machine, call the `MarkAsVirtualMachine` method on the template. You must specify a resource pool and, optionally, a host for the virtual machine. Host and resource pool must be under the same `ComputeResource`. When the operation completes, the template no longer exists.
- To keep the template, clone the template by calling the `CloneVM_Task` method on the template. In the `VirtualMachineCloneSpec` (the `spec` parameter), set the `template` property to `false`.

Accessing Information About a Virtual Machine

After you have created a virtual machine, you can retrieve information about the virtual machine through the `VirtualMachineConfigInfo` properties. See the *API Reference* for a complete list.

Checking Default Files

After you have created a virtual machine, several files are generated and placed in the directory specified in the `VirtualMachineConfigSpec.files` property.

Table 11-1. Default Files

File	Usage	File Description	File Format
<code>.vmx</code>	<code>.vmname.vmx</code>	Virtual machine configuration file.	ASCII
<code>.vmxf</code>	<code>vmname.vmx</code>	Additional virtual machine configuration files, available, for example, with teamed virtual machines.	ASCII
<code>.vmdk</code>	<code>vmname.vmdk</code>	Virtual disk file.	ASCII
<code>.flat.vmdk</code>	<code>vmname.flat.vmdk</code>	Preallocated virtual disk in binary format.	Binary
<code>.vswp</code>	<code>vmname.vswp</code>	Swap file.	
<code>.nvram</code>	<code>vmname.nvram</code> or <code>nvram</code>	Non-volatile RAM. Stores virtual machine BIOS information.	
<code>.vmss</code>	<code>vmname.vmss</code>	Virtual machine suspend file.	

Table 11-1. Default Files (continued)

File	Usage	File Description	File Format
.log	vmware.log	Virtual machine log file.	ASCII
#.log	vmware-#.log	Old virtual machine log files. # is a number starting with 1.	ASCII
.vmtx	<i>vmname.vmtx</i>	Virtual machine template file.	ASCII

If you are using snapshots, the following additional files might be available. See [Snapshots](#).

File Extension	Usage	File Description
.vmsd	<i>vmname.vmsd</i>	Virtual machine snapshot file.
.vmsn	<i>vmname.vmsn</i>	Virtual machine snapshot data file.
**delta.vmdk		Snapshot difference file. A number preceding the extension increases with more snapshots.
**vmdk		Metadata about a snapshot.
-Snapshot#.vmsn		Snapshot of virtual machine memory. Snapshot size is equal to the size of your virtual machine's maximum memory.

Checking Default Devices

When you create a virtual machine, you are also creating a set of default devices, based on the hardware version associated with your SDK. You can see these devices using the `EnvironmentBrowser.QueryConfigOption` method. For example, the IDE controllers are created by default. Many of these default devices contain properties that you cannot change.

However, you can add the following optional devices to the default set: `VirtualSerialPort`, `VirtualParallelPort`, `VirtualFloppy`, `VirtualCdrom`, `VirtualUSB`, `VirtualEthernetCard`, `VirtualDisk`, and `VirtualSCSIPassthrough`. See the `VirtualDevice` Data Object in the *API Reference* for more information about each of these optional devices.

Caution Do not try to change default device properties using the `VirtualMachineConfigSpec.deviceChange` method discussed in [Adding Devices to Virtual Machines](#), because the `deviceChange` method is not applicable to default device properties.

Configuring a Virtual Machine

You can configure a virtual machine during creation (`Folder.CreateVM_Task`) or cloning (`VirtualMachine.CloneVM_Task`). You can also reconfigure a virtual machine using `VirtualMachine.ReconfigVM_Task`.

In vSphere 5.5 and later, the `ReconfigVM_Task` method will throw an error when it attempts to change certain virtual machine properties while the virtual machine is powered on. In previous releases, the server would modify the properties in the configuration specification and the changes would take effect after the virtual machine resets, reboots, or performs a fast suspend and resume.

Since vSphere 5.5, a reconfigure operation modifies the virtual machine in real time, so that the virtual machine properties have been changed by the time the method returns control to the client.

You cannot reconfigure a powered on virtual machine successfully if you specify any of the following properties in the `VirtualMachineConfigSpec` when you call the `ReconfigVM_Task` method.

Property	VirtualMachineConfigSpec Path
<code>VirtualDevice.unitNumber</code>	<code>deviceChange.device.unitNumber</code>
<code>VirtualDevicePciBusSlotInfo.pciSlotNumber</code>	<code>deviceChange.device.slotInfo.pciSlotNumber</code>
<code>VirtualDiskFlatVer1BackingInfo.diskMode</code> <code>VirtualDiskFlatVer2BackingInfo.diskMode</code> <code>VirtualDiskRawDiskMappingVer1BackingInfo.diskMode</code> <code>VirtualDiskSeSparseBackingInfo.diskMode</code> <code>VirtualDiskSparseVer1BackingInfo.diskMode</code> <code>VirtualDiskSparseVer2BackingInfo.diskMode</code>	<code>deviceChange.device.backing.diskMode</code>
<code>VirtualDiskFlatVer2BackingInfo.digestEnabled</code> <code>VirtualDiskSeSparseBackingInfo.digestEnabled</code>	<code>deviceChange.device.backing.digestEnabled</code>
<code>VirtualMachineConfigSpec.changeTrackingEnabled</code>	<code>changeTrackingEnabled</code>
<code>VirtualEthernetCard.addressType</code>	<code>deviceChange.device.addressType</code>
<code>VirtualEthernetCard.macAddress</code>	<code>deviceChange.device.macAddress</code>
<code>VirtualEthernetCard.wakeOnLanEnabled</code>	<code>deviceChange.device.wakeOnLanEnabled</code>
<code>VirtualSCSIController.sharedBus</code>	<code>deviceChange.device.sharedBus</code>
<code>VirtualSerialPort.yieldOnPoll</code>	<code>deviceChange.device.yieldOnPoll</code>
<code>VirtualUSBController.autoConnectDevices</code> <code>VirtualUSBXHCIController.autoConnectDevices</code>	<code>deviceChange.device.autoConnectDevices</code>
<code>VirtualUSBController.ehciEnabled</code>	<code>deviceChange.device.ehciEnabled</code>
<code>VirtualMachineVideoCard.useAutoDetect</code>	<code>deviceChange.device.useAutoDetect</code>
<code>VirtualMachineVideoCard.videoRamSizeInKB</code>	<code>deviceChange.device.videoRamSizeInKB</code>
<code>VirtualMachineVideoCard.numDisplays</code>	<code>deviceChange.device.numDisplays</code>
<code>VirtualMachineVideoCard.use3dRendererSupported</code>	<code>deviceChange.device.use3dRendererSupported</code>

The *API Reference* lists all properties and includes information about required permissions for these configuration methods. The following sections describe some commonly specified attributes.

Name and Location

You can specify the display name for the virtual machine by setting the `VirtualMachineConfigSpec.name` property. Any % (percent) character used in this `name` parameter must be escaped, unless it is used to start an escape sequence. Clients can also escape any other characters in this `name` parameter.

Use the `annotation` field to provide a description of the virtual machine. To remove an existing description, specify the empty string as the value of `annotation`.

The location of the virtual machine is determined implicitly during creation because you call a `Folder.CreateVM_Task` method and specify resource pool and optional target host the virtual machine should belong to. See [Chapter 16 Resource Management](#) for a discussion of resource pools and virtual machine location.

Hardware Version

The hardware version of a virtual machine indicates the lower-level virtual hardware features a virtual machine supports, such as BIOS, number of virtual slots, maximum number of CPUs, maximum memory configuration, and other hardware characteristics.

For a newly created virtual machine, the default hardware version is the most recent version available on the host where the virtual machine is created. To increase compatibility, you might want to create a virtual machine with a hardware version older than the highest supported version. You can do so by specifying the `VirtualMachineConfigSpec.version` property during virtual machine creation. For existing virtual machines, call the `VirtualMachine.UpgradeVM_Task` method.

The hardware version of a virtual machine can be lower than the highest version supported by the ESXi host it is running on under the following conditions:

- You migrate a virtual machine to a newer version of ESXi after it was created on a host that was running an earlier version of ESXi.
- You create a virtual machine on a newer version of ESXi by using an existing virtual disk that was created on a host that was running an earlier version of ESXi.
- You add a virtual disk created on a host that was running an earlier version of ESXi to a virtual machine created on a newer version of ESXi.

Virtual machines with hardware versions lower than 4 can run on ESX/ESXi 4.x hosts but have reduced performance and capabilities. In particular, you cannot add or remove virtual devices on virtual machines with hardware versions lower than 4 when they reside on an ESX/ESXi 4.x host. To make full use of these virtual machines, upgrade the virtual hardware.

Boot Options

You can control a virtual machine's boot behavior by setting the `VirtualMachineConfigSpec.bootOptions` property. The `VirtualMachineBootOptions` data object in that property allows you to specify the following properties:

- `bootDelay` – Delay before starting the boot sequence, in milliseconds.
- `bootRetryDelay` – Delay before a boot retry, in milliseconds. This property is only considered if the `bootRetryEnabled` property is set to `true`.
- `bootRetryEnabled` – If set to `true`, a virtual machine that fails to boot tries again after the `bootRetryDelay` time period has elapsed.
- `enterBIOSSetup` – If set to `true`, the virtual machine enters BIOS setup the next time it boots. The virtual machine resets this flag to `false` so subsequent boots proceed normally.

Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs. You specify the guest operating system in the following two properties:

- `guestosid` – Specify one of the constants in the `VirtualMachineGuestOsIdentifier` as a string.
- `alternateGuestName` – Full name for the guest operating system. Use this property if `guestosid` is one of the values of `VirtualMachineGuestOsIdentifier` starting with `other*`.

CPU and Memory Information

To manage compute and memory resources at the virtual machine level, you use several properties of the `VirtualMachineConfigSpec` data object. These properties specify the CPU and memory configuration for the virtual machine. You can also specify the operational boundaries for resource allocation, and the priority of the virtual machine in case of resource contention.

Configuring Virtual CPUs and Memory

To configure the capabilities of a virtual machine, you use properties of the `VirtualMachineConfigSpec` data object to specify the machine characteristics available to the guest operating system. The guest uses these capabilities in the same way as resources on a physical machine.

Configuring Virtual CPUs

Set the number of virtual CPU cores for the virtual machine with the `VirtualMachineConfigSpec.numCPUs` property. Legal values for this property change depending on the `guestosid` value you specify. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

The guest operating system acts as if it had `numCPUs` cores available at all times, but the host's physical resources are shared by all its virtual machines. The host allocates physical cores in time slices as backing for virtual cores. For information about how to specify guidance for resource allocation, see [Configuring Resource Allocation Constraints for Virtual Machines](#).

Configuring Multi-Core CPUs

Set the number of cores per CPU chip with the `VirtualMachineConfigSpec.numCoresPerSocket` property. The value must be an integral divisor of `VirtualMachineConfigSpec.numCPUs`. The default value is 1 if the property is omitted for virtual machine creation. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

Configuring Memory

Set the RAM size for a virtual machine with the `VirtualMachineConfigSpec.memoryMB` property. If you use `VirtualMachineConfigSpec` to update the virtual machine properties, you can omit this property to leave it unchanged.

The guest operating system acts as if it had `memoryMB` available at all times, but the host's physical resources are shared by all its virtual machines. The amount of physical memory available as backing for virtual machines can vary over time, and it can affect virtual machine performance. For information about how to specify guidance for resource allocation, see [Configuring Resource Allocation Constraints for Virtual Machines](#).

CPU Processors and Memory Affinity

If your virtual machine is on an ESXi system, and if you have a license that supports Symmetric Multiprocessors (SMP), you can configure the virtual machine to have multiple virtual CPUs by setting `cpuAffinity` and `memoryAffinity`. You define a set of integers that represents the processors (for CPU) and NUMA nodes (for memory). If you are reconfiguring the affinity setting and leave the array empty, any existing affinity is removed. See the *Resource Management Guide* for a discussion of NUMA nodes and affinity.

CPU Features

You can use the `VirtualMachineConfigSpec.cpuFeatureMask[].info` property to represent the CPU features requirements for a virtual machine or guest operating system. See the `HostCpuIdInfo` data object discussion in the *API Reference* for a detailed discussion.

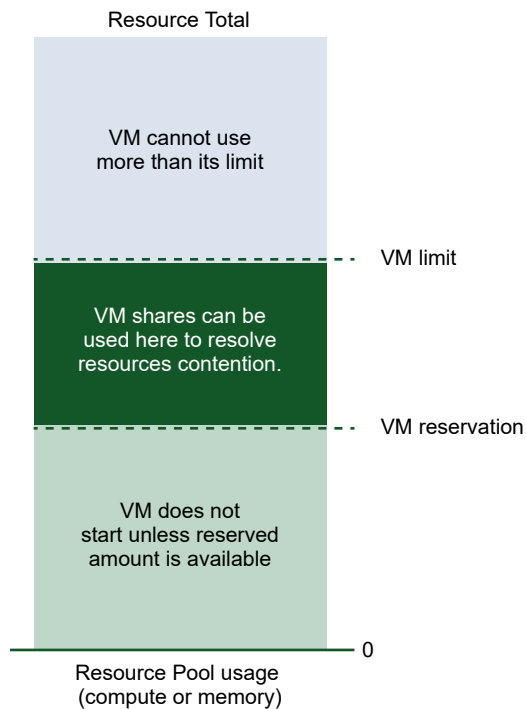
CPU and Memory Modification for Running Virtual Machines

Set `CpuHotAddEnabled` and `CpuHotRemoveEnabled` to specify whether virtual processors can be added to or removed from a virtual machine while the virtual machine is running. Set `MemoryHotAddEnabled` to specify whether memory can be added while the virtual machine is running.

Configuring Resource Allocation Constraints for Virtual Machines

The host hypervisor allocates physical resources as backing for the virtual resources needed by guest operating systems. You can specify resource allocation preferences that guide and constrain the hypervisor when allocating resources to running virtual machines.

To specify allocation preferences for CPU and memory, use the `cpuAllocation` and `memoryAllocation` properties of `VirtualMachineConfigSpec`. Both properties are data objects of type `ResourceAllocationInfo` objects. The `ResourceAllocationInfo` object has properties to specify upper and lower allocation limits, and properties to specify relative priorities when virtual machines contend for resources.



To specify boundaries for resource allocation, use these properties of `ResourceAllocationInfo`:

- `reservation` – Lower limit for resource allocation to the virtual machine. If the hypervisor cannot reserve this amount, it does not start the virtual machine. If the running virtual machine uses less than its reserved amount, other running virtual machines can use a part of the reserved resources temporarily.

- `limit` – Upper limit for CPU or memory resources assigned to this virtual machine. The virtual machine does not exceed this limit, even if unused resources are available. This property is typically used to limit the impact of the virtual machine on other running virtual machines. If the property is set to -1, the configured virtual memory size or the configured number of virtual cores limits the resource allocation.

Note Although the number of virtual CPU cores limits the compute resource allocated, the virtual bandwidth allocated also depends on the speed of the physical CPU cores assigned as backing.

To specify preferences for resource allocation in case of contention, use the `shares` property of `ResourceAllocationInfo`. The `shares` property is a nested data object of type `SharesInfo`, which specifies a relative priority for resolving resource contention between virtual machines. The `SharesInfo` data object has two properties, `level` and `shares`:

- `level` – An enum type with four potential values: `high`, `normal`, `low`, and `custom`. If you specify the value `custom`, you must also specify the `shares` property. If you specify any other value for `level`, the `shares` property is ignored.
- `shares` – In a situation of resource contention, the `shares` value is compared against the `shares` values of other virtual machines, and resources are allocated first to the virtual machine with the highest `shares` value.

The most direct way to specify resource priority is to set the `shares` values manually. This is most useful in environments where all virtual machines use custom `shares` values. In a mixed environment, the configuration settings do not compare directly, which can lead to unexpected results.

A more sophisticated way of specifying priority is to specify `high`, `normal`, or `low` for the `level` property. These settings cause the host to calculate the shares of a virtual machine in a way that factors in the configured memory size or the configured virtual CPU cores of the virtual machine. The result gives more priority shares to virtual machines with larger configurations.

Note Both the custom shares values and the calculated shares values prioritize virtual machines only among their siblings running in the same resource pool. For a system of virtual machines running only in the default root resource pool, determining priority is fairly simple. For data centers with nested resource pools, a virtual machine's priority in the larger context of all running virtual machines depends also on the priority of its parent resource pool.

For a detailed examination of resource pools and how they affect priority shares calculations, see [Resource Management](#). The *Resource Management Guide* includes a detailed discussion of resource allocation in the vSphere environment.

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines.

Virtual Network Interfaces

You can add a virtual network interface to a virtual machine using a subclass of `VirtualEthernetCard`, you can set the `addressType` to `Manual`, `Generated`, or `Assigned`. If you choose `Assigned`, you can specify a MAC address explicitly.

For more information about virtual devices in general, see [Adding Devices to Virtual Machines](#).

The number of virtual network interfaces depends on the hardware version you specify for a virtual machine. Hardware version 7 virtual machines support up to ten virtual NICs. Hardware version 4 virtual machines support up to four virtual NICs.

Virtual Machine MAC Address

Upon virtual machine creation, ESXi or vCenter Server systems assign each virtual network interface its own unique MAC address. The first three bytes of the MAC address that is generated for each virtual network adapter consists of a manufacturer-specific Organizationally Unique Identifier (OUI). The MAC address-generation algorithm produces the other three bytes. vSphere generates MAC addresses that are checked for conflicts. After the MAC address has been generated, it does not change unless the virtual machine is moved to a different location.

All MAC addresses that have been assigned to virtual network interfaces of running and suspended virtual machines on a given physical machine are tracked. The MAC address of a powered off virtual machine is not checked against those of running or suspended virtual machines. It is possible that a virtual machine acquires a different MAC address after a move.

The *ESXi Configuration Guide* discusses virtual machine MAC addresses in detail.

Fibre Channel NPIV Settings

N-port ID virtualization (NPIV) supports sharing a single physical FC HBA port among multiple virtual ports, each with unique identifiers. This capability lets you control virtual machine access to LUNs on a per-virtual machine basis.

Each virtual port is identified by a pair of world wide names (WWNs): a world wide port name (WWPN) and a world wide node name (WWNN). These WWNs are assigned by vCenter Server. For detailed information on how to configure NPIV for a virtual machine, see the *Fibre Channel SAN Configuration Guide*.

NPIV support is subject to the following limitations:

- NPIV must be enabled on the SAN switch. Contact switch vendors for information about enabling NPIV on their devices.
- NPIV is supported only for virtual machines with RDM disks. Virtual machines with regular virtual disks continue to use the WWNs of the host's physical HBAs.
- Virtual machines on a host have access to a LUN using their NPIV WWNs if the physical HBAs on the ESXi host have access to a LUN using its WWNs. Ensure that access is provided to both the host and the virtual machines

You can set up NPIV with the `VirtualMachineConfigSpec` properties that start with `npiv`.

File Locations

File locations for a virtual machine are specified in the following properties:

- `VirtualMachineConfigSpec.files` is a `VirtualMachineFileInfo` data object that allows you to specify the log directory, snapshot directory, suspend directory, and configuration file location. Most locations have a default that you can change as needed.
- `VirtualMachineConfigSpec.locationID` is a 128-bit hash based on the virtual machine's configuration file location and the UUID of the host the virtual machine is assigned to. This property is not usually set by developers; however, clearing this property by setting it to an empty string is recommended if you move the virtual machine.

If a virtual machine's `VirtualMachineCapability.swapPlacementSupported` property is `true` for a virtual machine, you can specify a value for the `VirtualMachineConfigSpec.swapPlacement` property. The value must be one of the values of the `VirtualMachineConfigInfoSwapPlacementType` enumeration, as a string.

Adding Devices to Virtual Machines

You can add devices to a virtual machine during creation using the `VirtualMachineConfigSpec.deviceChange` property, which is a `VirtualDeviceSpec`. You specify the host device that the virtual device should map to by using a backing object. A backing object represents the host device associated with a virtual device.

- Backing option objects – You can find out which devices the host supports by extracting the relevant backing option object.
- Backing information object – The backing information object allows you to supply data for virtual device configuration. You access a `VirtualDeviceBackinInfo` object as follows:

```
VirtualMachineConfigSpec.deviceChange[].device.backing
```

To add a device to a virtual machine, you must first find out which devices are supported on the corresponding ESXi host, and then specify a `VirtualDevice` object. Perform these tasks to add a device to a virtual machine:

Procedure

- 1 Find out which devices your ESXi system supports by calling the `QueryConfigOption` method, which you can access through the `VirtualMachine.environmentBrowser` property. The method returns a `VirtualMachineConfigOption` data object that specifies what the ESXi supports. For example, `VirtualMachineConfigOption.hardwareOptions` includes information about supported CPU and memory and an array of `VirtualDeviceOption` data objects.

Note You cannot use the `QueryConfigOption` method to create another instance of a default device. If you attempt to add a default device, such as an IDE controller, the server ignores the operation.

- 2 Specify the backing information object for the device. The actual process for defining the object differs for different objects. For example, for a CD-ROM passthrough device, you use a `VirtualCdromPassthroughBackingInfo` device. The `VirtualDevice.backing` property is a `VirtualDeviceBackingInfo` object which is extended by devices.

The following code fragment adds a CD-ROM passthrough device:

```
VirtualCdromPassthroughBackingInfo vcpbi = new VirtualCdromPassthroughBackingInfo();
// Does the virtual device have exclusive access to the CD-ROM device?
vcpbi.setExclusive(false);
// Specifies the device name.
vcpbi.setDeviceName('cdrom0');
```

- 3 Specify connection information for the device.

The `VirtualDevice.connectable` property is a `VirtualDeviceConnectInfo` data object. This object provides information about restrictions on removing the device while a virtual machine is running. This property is `null` if the device is not removable.

```
VirtualDeviceConnectInfo vdc_i = new VirtualDeviceConnectInfo();
// Allow the guest to control whether the virtual device is connected?
vdc_i.setAllowGuestControl(false);
// Is the device currently connected?
vdc_i.setConnected(true);
// Connect the device when the virtual machine starts?
vdc_i.setStartConnected(true);
```

- 4 Define the controller key, the virtual device key, and the unit number.

You define these items with the integer properties: `controllerKey`, `key`, and `unitNumber`. See the `VirtualDevice` data object in the *API Reference*.

5 Specify device Information.

The `deviceInfo` property is a `Description` data object that has a `name` property and a `summary` property. You can supply a string value for each, describing the device.

```
Description vddesc = new Description();
vddesc.setLabel('CD-ROM Device cdrom0');
vddesc.setSummary('The CD-ROM device for this virtual machine.');
```

6 Specify the virtual device as the device property of a `VirtualDeviceConfigSpec`.

7 Specify the `VirtualDeviceConfigSpec` as the `deviceChange` property to the `VirtualMachineConfigSpec` that you pass in to a `Folder.CreateVM_Task` or `VirtualMachine.ReconfigVM_Task` method.

Example

Here's the complete code fragment for a CD-ROM passthrough device:

```
VirtualDevice vd = new VirtualDevice();
vd.setBacking(vcpbi);
vd.setConnectable(vdci);
vd.setControllerKey(257);
vd.setDeviceInfo(vddesc);
vd.setKey(2);
vd.setUnitNumber(25);
```

Performing Virtual Machine Power Operations

Just like physical machines, virtual machines have power states.

- **Powered on** – The virtual machine is running. If no OS has been installed, you can perform OS installation as you would for a physical machine.
- **Powered off** – The virtual machine is not running. You can still update the software on the virtual machine's physical disk, which is impossible for physical machines.
- **Suspended** – The virtual machine is paused and can be resumed; like a physical machine in standby or hibernate state.

Important Before you power on a virtual machine, you must make sure that the host has sufficient resources. You must have enough memory for the virtual machine, and some memory overhead. See [Querying Virtual Machine Memory Overhead](#).

`VirtualMachine` power operations allow you to change the power state. Each operation is sensitive to the current power state, for example, powering on a powered off virtual machine has the desired result while powering on a powered on virtual machine results in an error. You must check the current state before you run one of these tasks.

- **PowerOnVM_Task** – Powers on a virtual machine. If the virtual machine is suspended, this method resumes execution from the suspend point.

- `PowerOffVM_Task` – Powers off a virtual machine.
- `ResetVM_Task` – Resets power on this virtual machine. If the current state is `poweredOn`, `ResetVM_Task` first performs a hard `powerOff` operation. After the power state is `poweredOff`, `ResetVM_Task` performs a `powerOn` operation.

Although this method functions as a `powerOff` followed by a `powerOn`, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the reset method completes.

- `SuspendVM_Task` – Suspends the virtual machine. You can later power on the suspended virtual machine to the same state.

Virtual machines are often configured to start up the guest operating system when they are started, and try to shut down the guest operating system when being shut down. However, starting and stopping a virtual machine differs from starting and stopping the guest operating system (see [Customizing the Guest Operating System](#)).

Important Power operations might affect other virtual machines that are participating in a DRS cluster or VMware HA. See [Chapter 16 Resource Management](#) for information about DRS clusters and VMware HA.

You can use the `Datacenter.PowerOnMultiVM_Task` to power on multiple virtual machines in a datacenter. Pass an array of `VirtualMachine` managed object references and an array of option values to the method. If any of the virtual machines in the list is manually managed by VMware DRS, the system generates a DRS recommendation that the user needs to apply manually. Standalone or DRS disabled virtual machines are powered on for the current host. Virtual machines managed by DRS, to be placed by DRS, are powered on for the recommended host.

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the inventory (inside the folder from which you called the creation method by default), and it is registered. If you copy virtual machine files to relocate the virtual machine, or if you remove the files from the inventory using the vSphere Client, it becomes unregistered and unusable. You cannot power on a virtual machine that is not part of the inventory.

To restore the virtual machine to the inventory, and make it usable again, you can use the `RegisterVM_Task` method, defined in the `Folder` managed object. You can register the virtual machine to a host or to a resource pool. You can register the virtual machine as a template if you want to use it to clone other virtual machines from.

The `ColdMigration.java` sample illustrates both registering and reconfiguring a virtual machine. At the heart of the sample is the following call, which registers the virtual machine. Arguments include the virtual machine's current folder, datastore path, and name, whether to register as a template, and the resource pool or host to register the machine in.

```
ManagedObjectReference taskmor = cb.getConnection().getService().registerVM_Task(
    vmFolderMor, vmxPath, getVmName(), false, resourcePool, host);
```

After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

The `RemoveManagedObject.java` sample illustrates unregistering a virtual machine.

Customizing the Guest Operating System

You install the guest operating system on the virtual machine just as you would install it on a physical machine. Afterwards, you can use the Web Services API to retrieve information and perform some customization if VMware Tools is installed on top of the guest operating system.

`VirtualMachine` includes the following methods for managing the guest operating system:

- `ShutdownGuest` and `RebootGuest` shut down and reboot the guest OS, and `StandbyGuest` puts the guest in hibernate mode. In each case, you perform the action on the guest OS. For example, you might shut down Windows but leave the virtual machine running.
- `ResetGuestInformation` clears cached guest information. Guest information can be cleared only if the virtual machine is powered off. Use this method if stale information is cached, preventing reuse of an IP address or MAC address.
- `SetScreenResolution` sets the console screen size of the guest operating system. When you call this method, the change is reflected immediately the virtual machine console you can access in the vSphere Client.

You can customize the identity and network settings of the guest OS with the `CustomizationSpec` data object that is a parameter to `VirtualMachine.CustomizeVM_Task`. The `CustomizationSpec` is also a property of the `VirtualMachineCloneSpec` you pass in when cloning a virtual machine.

The settings you customize with this method are primarily virtual machine settings, but because the virtual machine and the guest OS share the information, you are also customizing the guest OS with this method.

The `CustomizationSpec` allows you to set the following properties:

- `encryptionKey` – Array of bytes that can be used as the public key for encrypting passwords of administrators.
- `globalIPSettings` – Contains a `CustomizationGlobalIPSettings` data object which specifies a list of DNS servers and a list of name resolution suffixes for the virtual network adapter.

- `identity` – Allows you to specify the network identity and settings, similar to the Microsoft Sysprep tool.
- `nicSettingMap` – Custom IP settings that are specific to a particular virtual network adapter.
- `options` – Optional operations (either `LinuxOptions` or `WinOptions`).

Installing VMware Tools

VMware Tools is a suite of utilities that enhances the performance of a virtual machine's guest operating system and improves virtual machine management. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools. The SDK requires that you install VMware Tools, or some operations related to the guest operating system fail.

Important You must install the guest operating system before you install VMware Tools.

With VMware Tools installed on the guest OS, the virtual machine obtains its DNS (domain name server) name and an IP address and is therefore reachable over the network.

`VirtualMachine` includes three methods for automating installation and upgrade of VMware Tools.

- `MountToolsInstaller` – Mounts the VMware Tools CD installer as a CD-ROM for the guest operating system. To monitor the status of the tools installation, check `GuestInfo.toolsStatus`. Check `GuestInfo.toolsVersionStatus` and `GuestInfo.toolsRunningStatus` for related information.
- `UnmountToolsInstaller` – Unmounts the VMware Tools installer CD.
- `UpgradeToolsTask` – Performs an upgrade of VMware Tools. This method assumes VMware Tools has been installed and is running. The method takes one argument, `InstallerOptions`, which allows you to specify command-line options passed to the installer to modify the installation procedure for tools.

Use the `ToolsConfigInfo` data object in `VirtualMachineConfigSpec.toolsInfo` property to specify the settings for the VMware Tools software running on the guest operating system.

Upgrading a Virtual Machine

You can upgrade virtual machine hardware by running the `VirtualMachine.UpgradeVM_Task` method. The method upgrades this virtual machine's virtual hardware to the latest revision that is supported by the virtual machine's current host. You can specify the version number as an argument. This method is useful if you want to run your virtual machine on a newer hypervisor that supports newer versions of the hardware.

Virtual Machine Management

12

Virtual machines can perform like physical computers and can be configured like physical computers. Virtual machines also support special features that physical computers do not support. This chapter discusses some of these features: migrating virtual machines, using snapshots, and using linked virtual machines.

For more information about configuring virtual machines, see the chapter [Configuring a Virtual Machine](#).

This chapter includes the following topics:

- [Virtual Machine Migration](#)
- [Snapshots](#)
- [Linked Virtual Machines](#)

Virtual Machine Migration

Migration is the process of moving a virtual machine from one host or storage location to another. Copying a virtual machine creates a new virtual machine. It is not a form of migration. vSphere supports the following migration types:

Migration Type	Description
Cold migration	Moves a powered-off virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage locations.
Migration of a suspended virtual machine	Moves a suspended virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage location.
Migration with vMotion	Moves a powered-on virtual machine to a new host. Migration with vMotion allows you to move a virtual machine to a new host without interruption in the availability of the virtual machine.
Migration with Storage vMotion	Moves the virtual disks or configuration file of a powered-on virtual machine to a new datastore. Migration with Storage vMotion allows you to move a virtual machine's storage without interruption in the availability of the virtual machine.

Migration of a suspended virtual machine and migration with vMotion are both sometimes called hot migration, because they allow migration of a virtual machine without powering it off.

You can move virtual machines manually or set up a scheduled task to perform the cold migration.

Cold Migration

If a virtual machine is shut down, you can move it to a different cluster, resource pool, or host by copying all virtual machine files to a different directory. The `ColdMigration` example illustrates this.

Migration with vMotion

VMware vMotion supports the live migration of running virtual machines from one physical server to another with no downtime. The source and destination physical servers may be in the same datacenter or in different datacenters.

When calling the `VirtualMachine` object's `MigrateVM_Task` method, you can specify a host or resource pool to migrate to, and optionally the task priority and power state of the virtual machine. The `vMotion` example makes the following queries, and performs the migration if possible:

- Uses `QueryVMotionCompatibility_Task` to check that two hosts are compatible.
- Uses `CheckMigrate_Task` to check whether migration is feasible. If two hosts are not compatible, virtual machines cannot be migrated from one to the other.
- Uses `CheckRelocation_Task` to check whether relocation is possible.

`MigrateVM_Task` was deprecated in vSphere 6.5 and can be replaced with `RelocateVM_Task`.

Using Storage vMotion

Storage vMotion allows you to move a running virtual machine from one storage cluster to another. Taking the virtual machine or its associated storage offline is not required. All datastore types are supported, including local storage, VMFS, NAS (network attached storage), and VVols (virtual volumes).

You can place the virtual machine and all its disks in a single location, or select separate locations for the virtual machine configuration file and each virtual disk. The virtual machine remains on the same host during Storage vMotion.

To perform Storage vMotion, call the `VirtualMachine` object's `RelocateVM_Task` method. The `RelocateVMSpec` passed in to the method allows you to specify the target datastore and target host or resource pool.

As of vSphere 6.5, you should call `RelocateVM_Task` for all types of vMotion:

- cold relocate, whether VM storage moves or not
- vMotion within a cluster
- Storage vMotion

- cross-datacenter vMotion (XvMotion)
- Folder moves not involving VM moves

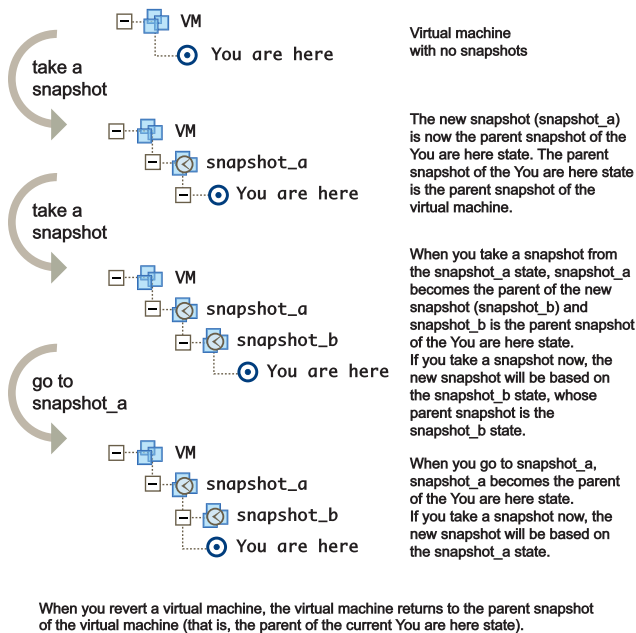
Snapshots

A snapshot is reproduction of the virtual machine just as it was when you took the snapshot. The snapshot includes the state of the data on all virtual machine disks and the virtual machine power state (on, off, or suspended). You can take a snapshot when a virtual machine is powered on, powered off, or suspended.

When you create a snapshot, the system creates a delta disk file for that snapshot in the datastore and writes any changes to that delta disk. You can later revert to the previous state of the virtual machine.

The `VirtualMachine` object has methods for creating snapshots, reverting to any snapshot in the tree, and removing snapshots.

Figure 12-1. Virtual Machine Snapshots



Snapshot hierarchies can become fairly complex. For example, assume that, in the example in [Figure 12-1. Virtual Machine Snapshots](#), you revert to `snapshot_a`. You might then work with and make changes to the `snapshot_a` virtual machine, and create a new snapshot, creating, in effect, a branching tree.

Creating a Snapshot

The `VirtualMachine.CreateSnapshot_Task` method creates a new snapshot of a virtual machine. As a side effect, the current snapshot becomes the parent of the new snapshot.

The method allows you to specify a name for the snapshot and also requires you set the `memory` and `quiesce` properties:

- `memory` – If `true`, a dump of the internal state of the virtual machine (basically a memory dump) is included in the snapshot. Memory snapshots consume time and resources, and take a while to create. When set to `false`, the power state of the snapshot is set to powered off.
- `quiesce` – If `true` and the virtual machine is powered on when the snapshot is taken, VMware Tools is used to quiesce the file system in the virtual machine. This ensures that a disk snapshot represents a consistent state of the guest file systems. If the virtual machine is powered off or VMware Tools is not available, the quiesce flag is ignored.

The `VMSnapshot.java` example calls this method as follows:

```
ManagedObjectReference taskMor = service.createSnapshot_Task(
    vmMor, snapshotName, desc, false, false);
```

The method returns MOR to a `Task` object with which to monitor the operation. The `info.result` property in the `Task` contains the newly created `VirtualMachineSnapshot` upon success.

Reverting to a Snapshot

When you revert to a snapshot, you restore a virtual machine to the state it was in when the snapshot was taken. The `VirtualMachine.RevertToSnapshot_Task` allows you to specify a target host and whether the virtual machine should be powered on.

If the virtual machine was running when the snapshot was taken, and you restore it, you must either specify the host to restore the snapshot to, or set the `SupressPowerOn` flag to `true`.

Deleting a Snapshot

You can delete all snapshots by calling `VirtualMachine.RemoveAllSnapshots` or by calling the `VirtualMachineSnapshot.RemoveSnapshot_Task` method. The `VirtualMachineSnapshot` object was previously returned in the task returned by the `CreateSnapshot_Task` method.

Linked Virtual Machines

Linked virtual machines are two or more virtual machines that share storage and support efficient sharing of duplicated data.

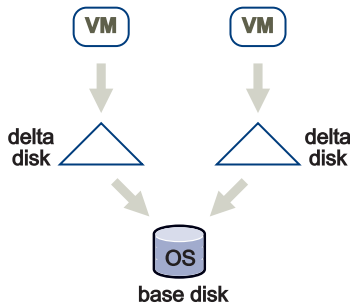
Linked Virtual Machines and Disk Backings

In its simplest form, shared storage is achieved through the use of delta disk backings. A delta disk backing is a virtual disk file that sits on top of a standard virtual disk backing file. Each time the guest operating system on a virtual machine writes to disk, the data is written to the delta disk. Each time the guest operating system on a virtual machine reads from disk, the virtual

machine first targets the disk block in the delta disk. If the data is not on the delta disk, the virtual machine looks for it on the base disk.

Linked virtual machines can be created from a snapshot or from the current running point. After you create a set of linked virtual machines, they share the base disk backing and each virtual machine has its own delta disk backing, as shown in [Figure 12-2. Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing](#).

Figure 12-2. Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing



Caution We recommend a limit of up to eight host virtual machines accessing the same base disk in a linked virtual machine group. However, you can have an unlimited number of linked virtual machines within each host virtual machine in the group.

Limitation for HA Clusters

Virtual machines in a linked clone group can be part of a VMware HA (high availability) cluster. The number of hosts in a cluster might affect HA's ability to restart a failed virtual machine.

- Clusters that contain ESXi 5.0 or earlier hosts – If a cluster has eight or fewer hosts, then linked virtual machines restart properly. However, if the cluster has more than eight hosts and any of the hosts are ESXi 5.0 or earlier, HA might not be able to restart a virtual machine after it fails. HA is not aware that virtual machines in the linked clone group are subject to the eight host limit. In this case, when HA responds to a failure, it might try to restart the virtual machine on a host that cannot participate in the group due to the maximum host limit. HA will attempt failover five times to different hosts. Thus, in clusters with 13 or more hosts, it is possible that HA will never try a host that is associated with the linked clone group.
- Clusters that contain only ESXi 5.1 or later hosts – The maximum host limit for a linked clone group is the maximum number of hosts allowed in a cluster. In this case, the number of hosts in the cluster does not affect the ability to restart failed virtual machines.

Creating a Linked Virtual Machine

You can create linked virtual machines in one of two ways:

- Clone the virtual machine from a snapshot.

- Clone the virtual machine from the current virtual machine state. This state might differ from the snapshot point.

Creating a Linked Virtual Machine From a Snapshot

You first create a snapshot, and then create the linked virtual machine from the snapshot.

Procedure

- 1 To create the snapshot, call the `CreateSnapshot_Task` method for the virtual machine. The virtual machine can be in any power state. The following pseudo code creates a snapshot named `snap1`. The code does not include a memory dump. VMware Tools is used to quiesce the file system in the virtual machine if the virtual machine is powered on.

```
myVm.CreateSnapshot("snap1", "snapshot for creating linked virtual machines", False, True)
```

- 2 To create the linked virtual machine, specify the snapshot you created and use a `VirtualMachineRelocateDiskMoveOptions.diskMoveType` of `createNewDeltaDiskBacking`, as illustrated in [Creating a Linked Virtual Machine from a Snapshot](#). Creating linked virtual machines from a snapshot works with virtual machines in any power state.

Example: Creating a Linked Virtual Machine from a Snapshot

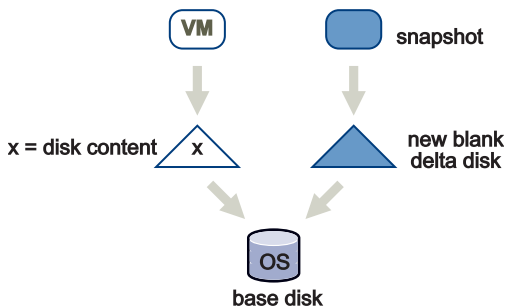
```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.createNewChildDiskBacking

cloneSpec = new VirtualMachineCloneSpec()
cloneSpec.powerOn = False
cloneSpec.template = False
cloneSpec.location = relSpec
cloneSpec.snapshot = myVm.snapshot.currentSnapshot

myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)
```

The result is a virtual machine with the same base disk as the original, but a new delta disk backing.

Figure 12-3. Creating a Linked Virtual Machine from a Snapshot



Creating a Linked Virtual Machine From the Current Running Point

To create a virtual machine from the current running point, clone the virtual machine, as in , but use a `diskMoveType` of `moveChildMostDiskBacking`. The virtual machine can be in any power state.

For more information about cloning a virtual machine, see [Creating a Linked Virtual Machine from a Snapshot](#).

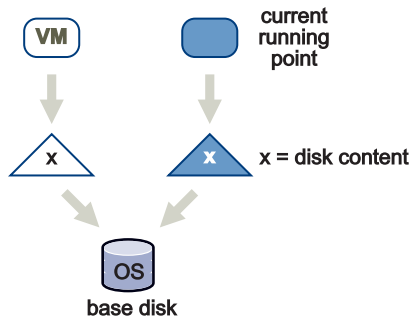
Example: Creating a Linked Virtual Machine from the Current Running Point

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking

cloneSpec = new VirtualMachineCloneSpec()
cloneSpec.powerOn = False
cloneSpec.template = False
cloneSpec.location = relSpec

myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)
```

Figure 12-4. Creating a Linked Virtual Machine from the Current Running Point



Instant Clone Virtual Machines

The instant clone operation creates a running virtual machine that continues running from the current state of the source virtual machine. There is no delay to power on the cloned virtual machine.

An instant clone virtual machine is a kind of linked clone with these characteristics:

- The clone starts out powered on and running.
- The clone replicates the running state of the source virtual machine.
- The clone shares memory pages with the source virtual machine as long as they remain unmodified.

The instant clone operation facilitates rapid deployment of virtual machines for low-latency response to peak computing demand. It also uses memory and disk space as efficiently as possible.

Because the clone replicates the running state of the source virtual machine, it usually requires customization of some network properties in the guest operating system to eliminate conflicts with the source. An instant clone virtual machine may also require new backings for some dedicated virtual devices, such as ISO files that back virtual optical drives.

Instant Clone Terminology

The instant clone feature has its own set of terminology, which uses some terms in ways specific to the feature.

stun

To stop the clock of a virtual machine temporarily, preventing guest operations during an instant clone operation.

freeze

To stop the clock of a virtual machine indefinitely, allowing a series of instant clone operations that reproduce the exact process state of the source virtual machine.

suspend

Save the state of a virtual machine so applications can resume, and put it to sleep.

resume

Continue running the virtual machine that was suspended, including its applications.

unstun

To release a stunned source virtual machine, allowing I/O operations to proceed.

source

A virtual machine that is cloned during an instant clone operation.

instant clone

- 1 An operation that quickly creates a new virtual machine that is an exact copy of a source virtual machine.
- 2 A new virtual machine created by an instant clone operation.

Run State of the Instant Clone Source

An instant clone can be created either from a source virtual machine in a frozen state, or from the current running point of a source virtual machine. Certain characteristics of the process and of the result can affect your choice of frozen virtual machine or current running point for the source virtual machine.

Advantages of Cloning from the Current Running Point

An instant Clone created from the current running point of a source virtual machine does not require an agent running in the guest operating system to invoke the frozen state.

If you create an instant clone from the current running point, the source virtual machine continues running from that point after a very brief delay, whereas a frozen source virtual machine requires intervention by the host to thaw it.

Advantages of Cloning from a Frozen Virtual Machine

A source virtual machine in the frozen state allows you to deploy any number of instant clones in exactly the same state, because the guest operating system retains its state after the clone operation.

When you create instant clones from the current running point of a source virtual machine, each cloning operation adds another delta disk to the source virtual machine. Each delta disk adds a small performance penalty, but a long chain of delta disks causes a noticeable performance penalty for virtual disk accesses.

The Instant Clone Process from a Frozen Source

The process to create an instant clone from a frozen source virtual machine requires one-time initial preparation, but after that it proceeds faster than creating an instant clone from the current running point. The source virtual machine must be booted and running, in a state that you want to replicate to other virtual machines. VMware Tools must be installed and running in the guest operating system so that you can run the command to freeze the source virtual machine .

Freezing the Source Virtual Machine for an Instant Clone Operation

Before you start the instant clone operation, you issue a VMware Tools command in the guest operating system, which freezes the source virtual machine. This is only needed one time, prior to the first instant clone operation. After that, the source virtual machine remains frozen. When VMware Tools receives the command to freeze the virtual machine, it passes the command through a back door to the host hypervisor.

The command to freeze the virtual machine depends on the guest operating system.

- For a Linux guest operating system, either of the following commands will freeze the virtual machine:
 - `vmware-rpctool "instantclone.freeze"`
 - `vmtoolsd -cmd "instantclone.freeze"`
- For a Windows guest operating system, the following command will freeze the virtual machine:
 - `C:\Program Files\VMware\VMware Tools\vmtoolsd.exe --cmd "instantclone.freeze"`

When a virtual machine is frozen, the hypervisor prevents future context switches to the guest operating system, which in effect stops the virtual machine's clock, although the virtual machine status continues to indicate that it is running. You can determine whether the virtual machine is frozen by examining the `runtimeinfo.InstantCloneFrozen` flag.

Instant Clone API Methods

The vSphere Web Services API offers several methods that implement different parts of the instant clone operation. By dividing the functionality into separate methods, the API provides flexibility in the timing and customization of the operation.

You can think of the instant clone operation as having three divisions: preparation, cloning, and customization.

Preparation includes these options:

- Installing the guest customization engine. The engine assists with customization of the guest network from the Web Services API. For more information, see [Installing the Guest Customization Engine](#).
- Disconnecting virtual NICs. You have the option to disconnect the NICs in the source virtual machine before you start the clone operation, to avoid network collisions between the source and its clones. For more information, see [Avoiding Network Identity Collisions after Instant Clone Operations](#).
- Freezing the source virtual machine. If you need to create a large number of clones, consider cloning from a frozen source. For more information, see [Run State of the Instant Clone Source](#).

Creating a clone of the source virtual machine requires a single method call. The `VirtualMachine.InstantClone_Task` method takes a parameter of type `InstantCloneSpec`, which enables you to give the clone a new name and configuration values. By using the key-value pairs in the `config` array, you can specify that the virtual NICs of the clone will start in a disconnected state, which avoids network conflicts with the source virtual machine. The `location` property of `InstantCloneSpec` is ignored.

Customization of a clone virtual machine takes place after the `InstantClone_Task` method completes. To customize, you invoke methods of the `VirtualMachineGuestCustomizationManager` managed object, as well as optional steps for application-dependent customization. For more information, see [Guest Network Customization for Instant Clone Virtual Machines](#).

How the Instant Clone Operation Works

The instant clone operation involves the following sequence of steps.

Stunning the Instant Clone Source Virtual Machine

When you issue the API command to initiate an instant clone operation, the host begins by doing a stun operation on the source virtual machine. The stunned condition lasts a very short time, and is not directly visible to the API user.

The stunned condition blocks all I/O operations of the source virtual machine while the instant clone is created. The stun is redundant in the case of a frozen virtual machine. During the time the source virtual machine is stunned, the host creates a delta disk for the source virtual machine, which remains empty as long as the source virtual machine is stunned or frozen.

Copying Virtual Disks During the Instant Clone Operation

Then the host creates an empty delta disk for the instant clone virtual machine. This delta disk links to the base disk of the source virtual machine. More precisely, if the source virtual machine already had one or more delta disks, the instant clone delta disk links to the second delta disk in the disk chain of the source. The second delta disk is the delta disk that was the topmost delta disk when the `stun` operation was invoked.

Copying Memory During the Instant Clone Operation

The host creates a new virtual memory paging file for the instant clone virtual machine, whose page tables link to the paging file of the source virtual machine. When the instant clone virtual machine is complete and is allowed to run instructions, all its memory pages will be shared with the source virtual machine until either virtual machine does a write to memory.

Configuring the Instant Clone Virtual Machine

Then the host creates a virtual machine configuration file for the instant clone, which links to its delta disk and memory paging file. At this point, the instant clone virtual machine is capable of running instructions, and the host unstuns it. Both the memory and the virtual disks are shared with the source virtual machine, so the instant clone is identical to the source virtual machine, from the guest operating system point of view.

When you create virtual machines with identical guest operating systems, the network settings conflict. Usually you will want to customize these settings and reset the network stack for each instance of the guest operating system.

Limitations of Instant Clones

The instant clone feature provides good performance and scalability, but it has a few limitations:

- The instant clone feature is not available when you connect directly to ESXi. Instant clone depends on functionality available only in vCenter Server.
- Delta disks created during the instant clone operation are not visible in the snapshot trees of the virtual machines. They cannot be managed with the `VirtualMachineSnapshot` managed object. Instead, you must manage the delta disks using the `VirtualMachine.Reconfigure()` method.
- The instant clone operation is partly parallel and partly serialized. You can start several operations concurrently, but part of the process is exclusive and requires serialization of requests.
- The instant clone operation responds to errors by backing out changes, such as delta disks, to restore the previous state of the file system and the source virtual machine. However, there is a small window of failure at the end of the operation where the cloned virtual machine is independent and cannot be backed out to recover from an error. This is a low risk situation.

- Instant clones create a delta disk for both the source and generated virtual machine, so resources can run out after vSphere raises a warning. The source virtual machine may be frozen to avoid excess growth. For current limits and ways to deal with failures, see the KB article *Instant Clone fails due to exceeding maximum number of delta disks*.
- The instant clone operation does not allow you to specify a different host for the clone. The clone must be on the same host as the source virtual machine because they share memory and virtual disk files.

Guest Customization of Instant Clone Virtual Machines

The instant clone operation produces a virtual machine whose memory is exactly the same as the source virtual machine. This can cause conflicts in network addressing, if two different virtual machines respond concurrently with the same network identity. The conflicting network attributes are usually the host name, IP address, and MAC address.

In the case of an instant clone derived from a frozen source virtual machine, a conflict might not immediately arise, because the source virtual machine is not active on the network. The instant clone simply assumes the identity of the source virtual machine when the host unstuns it. However, a second clone derived from the same frozen virtual machine could create a network conflict by duplicating the same identity as the first instant clone.

For an instant clone derived from the current running point of the source virtual machine, the problem is more acute because the source virtual machine resumes operation simultaneously with the instant clone.

To avoid network conflicts, you customize the instant clone to adopt a new identity, before allowing both the source virtual machine and the instant clone to run concurrently. In a situation where temporary network disruption is tolerable, you can ignore the conflict while you customize the network settings in the guest operating system of the instant clone.

Otherwise, you have two options to avoid the race condition that results when both virtual machines resume running from the same point. You can either disable the virtual NIC before the instant clone operation, or else you can specify that the virtual NIC in the instant clone will be disabled at the time when it resumes running after the operation. You specify the latter by setting the `config` property of the `spec` parameter when you invoke the `InstantClone_Task` method.

Avoiding Network Identity Collisions after Instant Clone Operations

To avoid a network conflict between a source virtual machine and its instant clone, you must disable at least one of the two conflicting virtual NICs until you complete network customization in the guest operating system of the instant clone. Here are some common approaches:

If the source virtual machine is frozen, its virtual NIC is effectively disabled, and you can reconfigure the guest network settings of the instant clone at any time before the source virtual machine is thawed.

If you derive the instant clone from the current running point of the source virtual machine, you can set the `connected` property of the virtual NIC backing to `false`, which disables it in the source before you start the instant clone operation. Then you re-enable the virtual NIC after the instant clone operation completes. After you customize the network settings in the guest operating system of the instant clone, you must re-enable the virtual NIC in the instant clone as well. This approach requires that you tolerate a network interruption until the instant clone operation is complete.

To avoid a network interruption, you can issue the instant clone operation with an `InstantCloneSpec` that specifies that the virtual NIC in the instant clone will be disconnected. The virtual NIC in the source virtual machine is not affected. After you complete guest operating system customization in the instant clone, you set the `connected` property of its virtual NIC back to `true`, and the instant clone operates under its new network identity.

In any case where you change the network identity settings in the guest operating system of the instant clone, you must reset the network stack for the virtual NIC to cause it to adopt the new settings. You can do this by invoking the `VirtualMachineGuestCustomizationManager.StartGuestNetwork_Task` method after you customize the clone's network settings, or you do the reset from the guest operating system, as with a physical machine. For more information, see [Restarting the Guest Network After Customization](#) and [Resetting the Network Stack in a Running Virtual Machine](#).

Removing Snapshots and Deleting Linked Virtual Machines

After you have created a group of linked virtual machines, you can remove a snapshot that was the basis for a linked virtual machine, or delete a virtual machine. Those actions affect disks in the linked virtual machine group. Perform the actions when connected to a vCenter Server system for disk consolidation or deletion.

- Snapshot removal – During snapshot removal, the snapshot metadata is also removed, and the virtual machine from which the snapshot was taken is no longer shown as having snapshots. If you remove a snapshot while connected to the ESXi host directly, shared disks are not consolidated and unnecessary levels of delta disks might result. If you remove a snapshot while connected to a vCenter Server system, shared disks are not consolidated, but unshared disks are consolidated.
- Virtual machine deletion – When you delete a virtual machine by directly connecting to the ESXi host, shared disks are not deleted. When you delete a virtual machine by connecting to a vCenter Server system, shared disks are not deleted, but unshared disks are deleted.

Caution Delete all linked virtual machines before deleting the source virtual machine from which they were created, so that you don't have orphaned or corrupt disk files on your file system.

Relocating a Virtual Machine in a Linked Virtual Machine Group

You can move the virtual machines in a linked virtual machine group between datastores and save storage. The contents of the delta disk might not be as important as the contents of the base, and you can save storage by removing the delta disk.

For more information, see [Figure 12-3. Creating a Linked Virtual Machine from a Snapshot](#).

Example: Relocating a Linked Virtual Machine

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking
relSpec.datastore = localDatastore
myVm.Relocate(relSpec)
```

You can relocate multiple linked virtual machines to a new datastore, but keep all shared storage during the relocation. To achieve the relocation, relocate the desired virtual machines one by one, giving the option to allow reattaching to an existing disk, as shown in [Relocating Multiple Linked Virtual Machines](#).

Example: Relocating Multiple Linked Virtual Machines

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType =
VirtualMachineRelocateDiskMoveOptions.moveAllDiskBackingsAndAllowSharing
relSpec.datastore = targetDatastore
myVm.Relocate(relSpec)
```

Promoting a Virtual Machine's Disk

Promoting a virtual machine's disk improves performance.

Important You can use the `PromoteDisks` API only when connected to a vCenter Server system.

You can use `PromoteDisks` to copy disk backings or to consolidate disk backings.

- **Copy** – If the `unlink` parameter is `true`, any disk backing that is shared by multiple virtual machines is copied so that this virtual machine has its own unshared version. Files are copied into the home directory of the virtual machine. This setting results in improved read performance, but higher space requirements. The following call copies and shares disks, and then collapses all unnecessary disks.

```
myVm.PromoteDisks(True, [])
```

- **Consolidate** – If the `unlink` parameter is `false`, any disk backing that is not shared between multiple virtual machines and not associated with a snapshot is consolidated with its child backing. The net effect is improved read performance at the cost of inhibiting future sharing. The following call eliminates any unnecessary disks:

```
myVm.PromoteDisks(False, [])
```

Promoting a virtual machine's disk might also be useful if you end up with disk backings that are not needed for snapshots or for sharing with other virtual machines.

Both uses of `PromoteDisks` take an optional second argument, which allows you to apply the method to only a subset of disks. For example, you can unshare and consolidate only the virtual disk with key 2001 as follows:

```
for any of my VMs in dev
  if (dev.key == 2001)
    disk2001 = dev

myVm.PromoteDisks(True, [disk2001])
```

Performing Advanced Manipulation of Delta Disks

For advanced manipulation of delta disks, you can use `VirtualDeviceConfigSpec` methods such as `VirtualDeviceConfigSpec.create` and `VirtualDeviceConfigSpec.add`.

Together `add` and `create` allow you to create a blank delta disk on top of an existing disk. For the `VirtualDeviceConfigSpec` you specify `add` to prepare the VM for a new delta disk, then `create` to initiate the file operation for the delta disk whose `parent` property is an existing disk. These methods create a new delta disk whose parent is the pre-existing disk.

One use case is adding a delta disk on top of an existing virtual disk in a virtual machine without creating a snapshot. [Adding a Delta Disk Backing](#) illustrates how to add the delta disk for the first virtual disk in the virtual machine.

Example: Adding a Delta Disk Backing

```
disk = None
for any of my VMs in dev
  if (VirtualDisk.isinstance == dev):
    disk = dev

# Remove the disk
removeDev = new VirtualDeviceConfigSpec()
removeDev.operation = "remove"
removeDev.device = disk

# Create a new delta disk which has the
# original disk as its parent disk
addDev = new VirtualDeviceConfigSpec()
addDev.operation = "add"
addDev.fileOperation = "create"
```

```
addDev.device = copy.copy(disk)
addDev.device.backing = copy.copy(disk.backing)
addDev.device.backing.fileName = "[" + disk.backing.datastore.name + "]"
addDev.device.backing.parent = disk.backing

spec = new VirtualMachineConfigSpec()
spec.deviceChange = [removeDev, addDev]
```

For working code with a similar use case, see the JAXWS sample program `VMDeltaDisk.java` in the SDK under `vsphere-ws/java`.

Virtual Machine Guest Operations

13

The vSphere Web Services SDK enables you to do operations inside a virtual machine, by communicating with the guest operating system.

This chapter includes the following topics:

- [Authenticating with the Guest Operating System](#)
- [Running Guest OS Operations](#)
- [Guest Operating System Customization](#)
- [Guest Customization for Stopped Virtual Machines](#)
- [Guest Customization Using cloud-init](#)
- [Guest Network Customization for Instant Clone Virtual Machines](#)
- [Resetting the Network Stack in a Running Virtual Machine](#)

Authenticating with the Guest Operating System

Authentication is required for most guest operations. Before you invoke a method that affects the guest, you must establish your credentials with the guest operating system and acquire a `GuestAuthentication` object that you pass to the method.

The `GuestAuthentication` type has several subclasses that hold guest credentials acquired by different methods. You can choose from several ways to authenticate:

- `NamePasswordAuthentication`
- `SAMLTokenAuthentication`
- `SSPIAuthentication`
- `TicketedSessionAuthentication`

Running Guest OS Operations

Guest OS operations manipulate processes, files, folders, and environment variables in a virtual machine's guest operating system.

The vSphere API offers the following managed object types for guest operations:

- GuestAuthManager – authenticate to acquire credentials in the guest OS.
- GuestFileManager – manipulate files, directories, and remote copying in the guest OS.
- GuestProcessManager – manipulate processes in the guest OS.
- GuestAliasManager – support single sign-on for guest operations; create and delete user aliases.
- GuestWindowsRegistryManager – manipulate keys and values in the Windows registry.
- VirtualMachineGuestCustomizationManager – customize guest settings, especially for instant clone virtual machines.

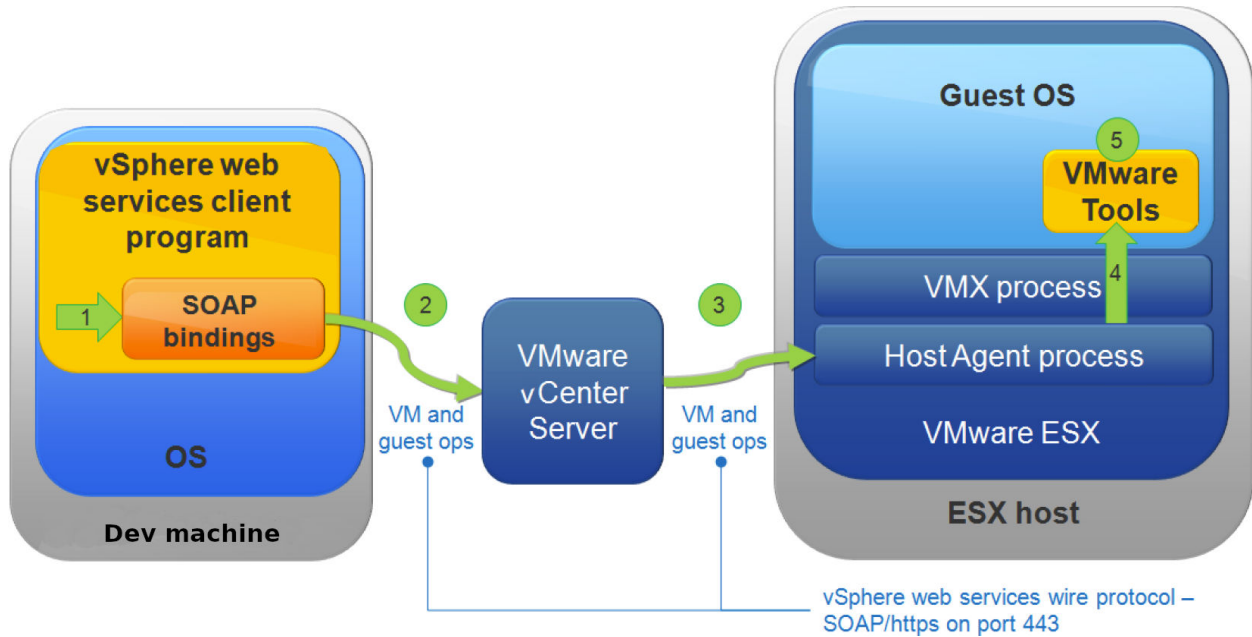
All the above managed objects are subclasses that inherit properties from GuestOperationsManager. In the vSphere API, the VirtualMachine and GuestInfo managed objects contain information about what guest operations might be running and relevant virtual machine state:

```
vim.VirtualMachine.guest()  
vim.vm.GuestInfo.guestOperationsReady  
vim.vm.GuestInfo.interactiveGuestOperationsReady
```

VMware Tools must be present to run guest operations. To perform interactive guest operations, the user must be logged into the console, for example through the vSphere Client. Steps involved are:

- 1 Java program translates to SOAP bindings.
- 2 Directives of vSphere API pass through vCenter Server.
- 3 Directives are relayed to ESXi host agent process.
- 4 Virtual machine executable passes guest operations to VMware Tools.
- 5 Guest OS performs guest operations.

Figure 13-1. Guest operations control flow



The following table summarizes the methods available for guest operations.

Table 13-1. GuestOperationsManager methods

Managed Object	Methods	Description
GuestAliasManager	AddGuestAlias	define alias for guest account
	ListGuestAliases	list guest aliases for specified user
	ListGuestMappedAliases	list alias map for in-guest user
	RemoveGuestAliasByCert	remove certificate associated aliases
GuestAuthManager	AcquireCredentialsInGuest	authenticate, return session object
	ReleaseCredentialsInGuest	release session object
	ValidateCredentialsInGuest	check authentication data or timeout
GuestFileManager	ChangeFileAttributesInGuest	change attributes of file in guest
	CreateTemporaryDirectoryInGuest	make a temporary directory
	CreateTemporaryFileInGuest	create a temporary file
	DeleteDirectoryInGuest	remote directory in guest OS
	DeleteFileInGuest	remove file in guest OS
	InitiateFileTransferFromGuest	start file transfer from guest OS
	InitiateFileTransferToGuest	start file transfer to guest OS
	ListFilesInGuest	list files or directories in guest

Table 13-1. GuestOperationsManager methods (continued)

Managed Object	Methods	Description
	MakeDirectoryInGuest	make a directory in guest
	MoveDirectoryInGuest	move or rename a directory in guest
	MoveFileInGuest	rename a file in guest
GuestWindowsRegistryManager	CreateRegistryKeyInGuest	create a registry key
	DeleteRegistryKeyInGuest	delete a registry key
	DeleteRegistryValueInGuest	delete a registry value
	ListRegistryKeysInGuest	list registry subkeys for a given key
	ListRegistryValuesInGuest	list registry values for a given key
	SetRegistryValueInGuest	set or create a registry value
GuestProcessManager	ListProcessesInGuest	list processes running in guest OS
	ReadEnvironmentVariableInGuest	read environment variable in guest
	StartProgramInGuest	start running program in guest
	TerminateProcessInGuest	stop a running process in guest

`InitiateFileTransferFromGuest` and `InitiateFileTransferToGuest` are useful for transferring small files between the host and guest. For large file transfers, virtual machines should be connected to the network because networking transfers are much faster.

Java Source Code Samples

Four Java code samples based on JAX-WS are available in the vSphere SDK for Web services, in this directory:

```
SDK/vsphere-ws/java/JAXWS/samples/com/vmware/guest
```

`CreateTemporaryFile.java` creates a temporary file inside a virtual machine, by calling the following method:

```
vimPort.createTemporaryFileInGuest(fileManagerRef, vmMOR, auth, prefix, suffix,
directoryPath);
```

`DownloadGuestFile.java` downloads a file from the guest to a specified path on the host where the client is running. The destination, a local file on the client host, is specified on the command line as `--localfilepath`.

```
vimPort.initiateFileTransferFromGuest(fileManagerRef, vmMOR, auth, guestFilePath);
```

`RunProgram.java` runs a specified program inside a guest operating system, with output re-directed to a temporary file, and downloads the resulting output to a file on the local client. The program must already exist on the guest, and is specified on the command line as `--guestprogrampath`. The output file to store on the client host is specified on the command line as `--localoutputfilepath`.

```
vimPort.startProgramInGuest(processManagerRef, vmMOR, auth, spec);
```

`UploadGuestFile.java` uploads a file from the client machine to a specified location inside the guest. The source, a local file on the client host, is specified on the command line as `--localfilepath`.

```
vimPort.initiateFileTransferToGuest(fileManagerRef, vmMOR, auth, guestFilePath,
    guestFileAttributes, fileSize, optionsmap.containsKey("overwrite"));
```

Guest Operating System Customization

You can use the vSphere Web Services SDK to customize several attributes of the guest operating system. The `CustomizationSpec` data object allows you to specify network and identity settings in the guest operating system.

Use the `CustomizationSpec` data object to specify the settings that you choose to modify. You can specify the following areas:

- IP addresses and gateway addresses for virtual NIC devices
- DNS servers
- Virtual hostname
- Domain name
- Time zone

You can customize a guest operating system while the virtual machine is stopped, by invoking the `VirtualMachine.CustomizeVM_Task` method. You can customize the guest operating system while the virtual machine is running, by invoking the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The instant clone feature requires that you customize the virtual machine while it is running. To initialize a virtual machine for a cloud environment, you must customize the virtual machine while it is stopped.

The two ways to customize a virtual machine are compared in the following table.

Table 13-2. Comparing Guest Operating System Customization

	Customization While Stopped	Instant Clone Customization
vSphere Compatibility	<ul style="list-style-type: none"> ■ vSphere 5 and later ■ Both vSphere and VMC 	<ul style="list-style-type: none"> ■ vSphere 7 and later ■ vSphere only (not VMC)
API Method	<code>VirtualMachine.CustomizeVM_Task()</code>	<ul style="list-style-type: none"> ■ <code>VirtualMachineGuestCustomizationManager.CustomizeGuest_Task()</code> ■ <code>VirtualMachineGuestCustomizationManager.StartGuestNetwork_Task()</code> ■ <code>VirtualMachineGuestCustomizationManager.AbortCustomization_Task()</code>
Creating CustomizationSpec	Create customization specifications in vSphere Client or in API.	Create customization specifications in API.
Virtual Machine State	Must be powered off.	Must be powered on.
vmtools	Must be installed. For Linux cloud-init customization, use open-vm-tools.	Must be installed and running.
cloud-init	Must be installed if using cloud-init customization.	Not used.
Customization Engine	Not used.	Required.
Supported Guests	Both Linux and Windows. For supported versions, see https://partnerweb.vmware.com/programs/guestOS/guest-os-customization-matrix.pdf .	Linux only. Supported distributions are listed in Customizing Guest Network Settings for Running Virtual Machines .
Network Adapter State	<p>For Sysprep or Linuxprep customization, <code>CustomizeVM_Task()</code> disconnects virtual adapters and prepares customization. The next time the virtual machine powers on, the customization will take effect and <code>vmtools</code> will reconnect virtual adapters.</p> <p>For CloudinitPrep customization, <code>CustomizeVM_Task()</code> does not disconnect virtual adapters.</p>	<code>InstantClone_Task()</code> disconnects virtual adapters before creating a clone. Use <code>StartGuestNetwork()</code> to restart virtual adapters after customization.
Rebooting the Guest	<p><code>vmtools</code> reboots the guest operating system after reconnecting virtual network adapters.</p> <p>No reboot needed after Linux cloud-init customization, either LinuxPrep or CloudinitPrep.</p>	<p>No reboot needed.</p> <p><code>StartGuestNetwork()</code> restarts network services after customization.</p>

Guest Customization for Stopped Virtual Machines

You can customize certain settings in the guest operating system with the help of the vSphere Web Services SDK. Customization is simpler for a virtual machine that is not running.

Use this general procedure to customize a guest operating system that you can shut down without a significant service impact. However, to initialize a virtual machine in a cloud environment, by using the cloud-init utility, see [Guest Customization Using cloud-init](#).

Prerequisites

A guest operating system can be easily customized while it is not running. If stopping the guest will cause a significant service impact, such as in an instant clone virtual machine, see [Guest Network Customization for Instant Clone Virtual Machines](#) instead.

Procedure

- 1 Shut down the guest operating system by using the `ShutdownGuest` method or the `PowerOffVM_Task` method.
- 2 Prepare a `CustomizationSpec` data object that specifies the desired global IP settings, virtual NIC settings, and so on.
- 3 Invoke the `CustomizeVM_Task` method of the `VirtualMachine` managed object.
- 4 Start the guest operating system by using the `PowerOnVM_Task` method of the `VirtualMachine` managed object.

What to do next

For more information about customizing the guest operating system, in the context of virtual machine configuration, see [Customizing the Guest Operating System](#).

Guest Customization Using cloud-init

You can customize certain settings in a Linux guest operating system with the help of the vSphere Web Services SDK. The cloud-init utility is a standardized way to initialize a virtual machine when you deploy it in a cloud environment.

Guest operating systems can be customized in one of several ways, depending on certain cloud-init configuration flags and on the subtype of the `identity` property of the `CustomizationSpec` data object. The following table shows the combinations needed to achieve raw data cloud-init customization, LinuxPrep with traditional customization, or LinuxPrep with cloud-init customization. Other combinations are not allowed.

Table 13-3. Enabling Guest Customization Types

Flags	CustomizationSpec.identity	Result
<code>allow_raw_data: true</code> or absent (default= <code>true</code>)	CloudInitPrep	VMware Tools will work with cloud-init to customize the guest with the unprocessed metadata and userdata from CloudinitPrep.
<code>disable_vmware_customization: true</code> or absent (default= <code>true</code>)	LinuxPrep	VMware Tools will do traditional customization on the guest.
<code>disable_vmware_customization: false</code>	LinuxPrep	VMware Tools will work with cloud-init to customize the guest with configuration derived from LinuxPrep.

Use the following procedures to apply settings to a virtual machine by passing raw configuration data to the cloud-init utility. You will use the `CustomizeVM_Task` method with a `CustomizationSpec` that contains configuration data compatible with the cloud-init standard.

Prepare a Virtual Machine Template for Raw Data cloud-init Customization

The following steps prepare a virtual machine template for customization with raw cloud-init data.

- 1 In the guest operating system, install open-vm-tools version 11.3.0 or later. Configure the Linux guest to run open-vm-tools services when the guest operating system starts.

Example for an Ubuntu 20.04 guest operating system:

```
$ sudo apt update
$ sudo apt install open-vm-tools
$ sudo vmtoolsd -v
$ sudo systemctl is-enabled open-vm-tools.service
```

- 2 In the guest operating system, install cloud-init version 21.1 or later. Configure the Linux guest to run cloud-init services when the guest operating system starts.

Example for an Ubuntu 20.04 guest operating system:

```
$ sudo apt update
$ sudo apt install cloud-init
$ sudo cloud-init -v
$ sudo systemctl is-enabled cloud-init.service
$ sudo systemctl is-enabled cloud-init-local.service
```

- 3 Configure cloud-init to accept the OVF data source for VMware raw cloud-init configuration.

Edit the cloud-init configuration file in `/etc/cloud/cloud.cfg` or in `/etc/cloud/cloud.cfg.d/*`, depending on the Linux distribution. Add `OVF` to the list of datasources accepted by cloud-init.

Example line for cloud-init version 21.3:

```
datasource_list: [ NoCloud, ConfigDrive, Azure, OVF, OpenStack, Ec2 ]
```

- 4 Configure cloud-init to enable VMware customization with raw cloud-init data.

Edit the cloud-init configuration file in `/etc/cloud/cloud.cfg` or in `/etc/cloud/cloud.cfg.d/*`, depending on the Linux distribution. Set the flag `disable_vmware_customization` to `true` and set the flag `allow_raw_data` to `true`. If customization time-out is an issue, you can set the value of `vmware_cust_file_max_wait` (default: 15 seconds).

Example lines for cloud-init version 21.3:

```
disable_vmware_customization: true
datasource:
  OVF:
    allow_raw_data: true
vmware_cust_file_max_wait: 25
```

- 5 Run the cloud-init clean step to remove any artifacts from previous configuration failures.

```
$ sudo cloud-init clean
```

- 6 Power off the virtual machine template.

```
$ sudo poweroff
```

Prepare a cloud-init Metadata File

Use the following information to prepare your cloud-init metadata file. The metadata file is required for cloud-init.

- The metadata file is a plain-text file formatted as YAML or JSON.
- The maximum file size is 512KB.
- All standard `v1` keys are accepted, such as:
 - `distro`
 - `local_hostname`
 - `platform`
 - `public_ssh_keys`
- The `instance-id` key is required. All other keys are optional.
- For network configuration, both `v1` and `v2` keys are accepted.

This example shows a metadata file in YAML format that includes a minimal network configuration:

```
instance-id: cloud-vm-example-1
local hostname: cloud-vm
network:
  version: 2
  ethernets:
    nics:
      match:
        name: eth*
      dhcp4: yes
```

This example shows a metadata file in JSON format that specifies a fixed IP network configuration:

```
{
  "instance-id": "cloud-vm-example-2",
  "local-hostname": "cloud-vm",
  "cloud-name": "vSphere",
  "distro": "ubuntu",
  "distro-version": "20.04",
  "distro-release": "focal",
  "network": {
    "version": 2,
    "ethernets": {
      "nics": {
        "match": {
          "name": "eth*"
        },
        "nameservers": {
          "addresses": ["203.0.113.1"],
          "search": ["www.example.com", "example.com"]
        },
        "gateway4": "192.0.2.253",
        "dhcp4": false,
        "addresses": [ "192.0.2.0/24" ]
      }
    }
  }
}
```

Prepare a cloud-init Userdata File

Use the following information to prepare your cloud-init userdata file. The userdata file is optional for cloud-init.

- The userdata file is a plain-text file in raw cloud-init format, without compression or base64 encoding.
- The maximum file size is 512KB.

- Userdata configuration can be used to accomplish a number of configuration tasks in the guest operating system. For instance, some of the more common tasks include:
 - Add user accounts
 - Install software
 - Run scripts
 - Import SSH keys

This example shows a userdata file that adds a new user to the guest operating system:

```
#cloud-config

users:
  - default
  - name: testuser
    primary_group: testgroup
    sudo: ALL=[ALL] NOPASSWD:ALL
    groups: sudo, wheel
    ssh_import_id: None
    lock_passwd: true
    ssh_authorized_keys:
      - ssh-rsa myRsaPubKey
```

Initiate Raw Data cloud-init Customization with the vSphere Web Services API

Use the following steps to initiate customization with raw cloud-init data by using the vSphere Web Services API.

- 1 Build a customization specification of type `CustomizationSpec`. Set the following properties in the spec:
 - a The `globalIPSettings` property must be present but need not contain its optional child properties. If the child properties are present, they will be ignored during the method call.
 - b The `identity` property must be present and must have type `CloudinitPrep`.
 - The child property `identity.metadata` is required. It contains the entire text of the metadata file, less any final new line character.
 - The child property `identity.userdata` is optional. If present, it contains the entire text of the userdata file, less any final new line character.

Example:

```
CloudinitPrep cloudinit = new CloudinitPrep();
cloudinit.setMetadata(metadataText);
cloudinit.setUserdata(userdataText);
CustomizationGlobalIPSettings ipSettings = new CustomizationGlobalIPSettings();
CustomizationSpec custSpec = new CustomizationSpec();
custSpec.setIdentity(cloudinit);
custSpec.setGlobalIPSettings(ipSettings);
```

Note The `nicSettingsMap` property, if present, will be ignored for raw data customization. The cloud-init tool accepts network customization only from the `identity.metadata` property.

- 2 Invoke the method `VirtualMachine.CustomizeVM_Task`, passing the `customizationSpec` as parameter. The method returns a `Task` object that you can use to monitor completion of the customization steps.

Example:

```
ManagedObjectReference custTask = service.CustomizeVM_Task(vmRef, custSpec);
```

- 3 Power on the virtual machine. As it boots, cloud-init applies the metadata configuration settings to the guest operating system. After the guest finishes booting and before it opens a shell window, cloud-init applies the userdata configuration.

Guest Network Customization for Instant Clone Virtual Machines

You can customize the network settings in the guest operating system with the help of the vSphere Web Services SDK. For instant clone operations, you must customize the clone without stopping the guest operating system.

Guest network customization for a running virtual machine involves several steps:

- Installation of the guest customization engine
- Disconnecting virtual NICs
- Customizing guest network settings
- Reconnecting virtual NICs
- Restarting the guest network
- Recovering from guest customization errors
- Running scripts for application-dependent customization

You can use these steps independently and you can run your own scripts in between steps as needed to customize, reset status, or restart processes in the guest.

Note You must authenticate using credentials accepted by the guest operating system before you can run guest operations.

Installing the Guest Customization Engine

The guest customization engine needs to be installed in a virtual machine before using the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. You can download the installer from the VMware Customer Connect web site.

The guest customization engine is available for most popular Linux distributions, using either an RPM package or a DEB package. To download and install it within the guest operating system, use the following steps.

Prerequisites

Start a web browser in a running virtual machine. For instant clone operations, prepare the source virtual machine by installing the customization engine before creating clones.

Procedure

- 1 Login to <https://my.vmware.com>.
- 2 Navigate to the Download VMware vSphere page for vSphere7.0.
- 3 Click the **Drivers & Tools** tab.
- 4 Find the entry for the instant clone customization engine.
- 5 Download the installation package appropriate for your Linux distribution:
 - The `.rpm` file for RHEL or SLES
 - The `.deb` file for Ubuntu
- 6 Run the command to install the package in the guest:
 - For the `.rpm` package, use this command:

```
sudo /bin/rpm -i --force yourdpgFilePath >> ./rpm.log 2>&1
```

- For the `.deb` package, use this command:

```
sudo /usr/bin/dpkg -i --force-all yourdpgFilePath >> ./dpkg.log 2>&1
```

What to do next

After the guest customization engine is installed, the guest is prepared to invoke `VirtualMachine.InstantClone_Task` or `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task`. You can delete the installation package to save virtual disk space.

Disconnecting Virtual NICs

Before you customize the guest network, you should disconnect the virtual NICs to quiesce network traffic during the operation. Disconnecting virtual NICs is a key step in the instant clone process.

You can disconnect a virtual NIC in one of these ways:

- In the `spec` parameter to the `InstantClone_Task` method you can specify that the virtual NIC should be disconnected during the instant clone operation. This way leaves the NIC in the source virtual machine active immediately after the clone operation, while the NIC in the clone is idle and ready for customization.
- In the `spec` parameter to the `ReconfigVM_Task` method you can change the connection state of a virtual NIC. This way is available either with or without an instant clone operation.

Disconnecting a Virtual NIC with `InstantClone_Task`

To disconnect a virtual NIC during an instant clone operation, prepare a `VirtualMachineInstantCloneSpec` data object that specifies the device should be in the disconnected state after the clone operation completes. You pass the clone specification in the `spec` parameter of `InstantClone_Task`. Assign a value of `disconnect` to the property `spec.location.deviceChange[].device.connectable.migrateConnect`.

Disconnecting a Virtual NIC with `ReconfigVM_Task`

To disconnect a virtual NIC by reconfiguring the virtual machine, prepare a `VirtualMachineConfigSpec` data object that specifies a disconnected virtual NIC. You pass the config specification in the `spec` parameter of the `ReconfigVM_Task` method. Assign a value of `False` to the property `spec.deviceChange[].device.connectable.connected`.

Customizing Guest Network Settings for Running Virtual Machines

You can use a method of the `VirtualMachineGuestCustomizationManager` managed object to apply new settings to the virtual NICs in the guest. The `VirtualMachineGuestCustomizationManager` uses the guest customization engine designed for instant clone operations to customize the guest while it continues running.

The customization step applies new settings for IP address, DNS server, and gateway server of one or more virtual NICs.

Note DNS settings can be specific to individual NICs in Windows operating systems. DNS settings are global in a Linux operating system.

The guest customization engine must be installed in the virtual machine before you call this method. See [Installing the Guest Customization Engine](#) for instructions.

Customization applies the new guest network settings that you specify in the `spec` parameter to the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The `spec` parameter is a data object of type `CustomizationSpec`, which contains a `nicSettingMap` property that holds an array that maps MAC address to network settings for one or more virtual NICs in the guest.

There are two ways to use the `nicSettingsMap` array in the `spec` parameter. You can do one of the following:

- Specify each virtual NIC explicitly by its MAC address, in `spec.nicSettingMap[i].macAddress`, and specify the virtual NIC's new settings in the properties of `spec.nicSettingMap[i].adapter`.
- Omit `spec.nicSettingMap[i].macAddress` for all virtual NICs and specify the new settings for each NIC in PCI bus order.

In addition to virtual NIC settings, you can customize these properties of a virtual machine:

- You can set the `CustomizationSpec.identity` property to a data object derived from the type `CustomizationIdentitySettings`. You must choose a subclass of the type that corresponds to the guest operating system installed in the virtual machine. In the `identity` object you can specify a new host name, domain name, and system time settings.
- You can set the `CustomizationSpec.globalIPSettings` property to a data object of type `CustomizationGlobalIPSettings` in which you specify DNS settings for all virtual NICs. Use this for Linux guest operating systems.

The `CustomizeGuest_Task` method supports the following guest operating systems:

- Red Hat Enterprise Linux 6.8 and higher
- Red Hat Enterprise Linux 7.4 and higher
- CentOS 7.4 and higher
- SUSE 11SP4
- SUSE 12SP3 and higher
- Ubuntu 16.04 and higher (LTS distributions)

The `toolsd` service must be running and available for this customization step.

Note The `CustomizeGuest_Task` method is asynchronous, but a critical part of the operation, which uses the customization engine, is synchronous.

You can pipeline the instant clone and customization operations to minimize the time needed to clone a virtual machine.

Note During an instant clone operation, the `toolsd` service is briefly offline while it resets connections in the clone. This results in no noticeable delay, but you might need to allow for a retry if you run pre-customization code.

The customization data is stored in the Namespace database in a `cust.cfg` format containing name-value pairs. This makes the customization process resilient to vMotion operations. It also means that you can access the data if needed as input for user-supplied customization scripts.

Reconnecting Virtual NICs in a Running Virtual Machine

After you customize a guest network in a running virtual machine, you must reconnect the virtual NICs before the guest operating system can use them.

To reconnect a virtual NIC by reconfiguring the virtual machine, prepare a `VirtualMachineConfigSpec` data object that specifies a connected virtual NIC. You pass the config specification in the `spec` parameter of the `ReconfigVM_Task` method. Assign a value of `True` to the property `spec.deviceChange[].device.connectable.connected`.

Restarting the Guest Network After Customization

After customizing guest network settings, you can restart the guest network manually from within the guest operating system, or by using a method of the `VirtualMachineGuestCustomizationManager` managed object. This method is important for instant clone operations, but it can be used for other purposes as well.

To restart the guest network manually, see [Resetting the Network Stack in a Running Virtual Machine](#).

To restart the guest network using the vSphere Web Services API, use the `VirtualMachineGuestCustomizationManager.StartGuestNetwork_Task` method. This method both reconnects the virtual NICs to the network and restarts the guest network service.

The `StartGuestNetwork_Task` method is supported on most popular Linux distributions.

Recovering from Guest Network Customization Errors

Guest network customization is a long-running operation with several potential failure points. If you encounter a failure, consider the following steps.

The guest network customization engine holds a lock on the `VirtualMachineGuestCustomizationManager` interface while it applies changes to the guest. A failure during customization or while restarting the network might leave a hung process or an orphaned lock.

If any step of the network customization returns an error, use the `VirtualMachineGuestCustomizationManager.AbortCustomization_Task` method to stop all running instances of the customization engine and to release the lock.

To troubleshoot a failure, you can examine the log file used by the `VirtualMachineGuestCustomizationManager.CustomizeGuest_Task` method. The log file is on the host file system at `/var/log/vmware-gosc/instant_clone_customization.log` for Linux hosts.

When you abort a guest customization operation, the guest network can be left in an inconsistent state. After the abort operation completes, you should check your customization settings and retry the `CustomizeGuest_Task` method in case of transient error conditions.

Note When you use the `VirtualMachineGuestCustomizationManager`, the customization procedure tolerates vMotion without failure.

Application-Dependent Customization

After you customize and restart the guest network, the virtual machine might need additional customization steps, depending on the applications that run in the guest operating system. For example, you might need to generate new encryption keys for an instant clone, or signal an application to close and re-open network sockets when you have configured new network IP addresses.

To do additional customization steps in the guest, use the `GuestFileManager` and the `GuestProcessManager` interfaces in the `GuestOperationsManager` managed object.

Resetting the Network Stack in a Running Virtual Machine

In any case where you change the network identity settings in the guest operating system of a running virtual machine, you must reset the network stack for the virtual NIC to cause it to adopt the new settings. You can do this by invoking the `VirtualMachineGuestCustomizationManager.RestartNetwork_Task` method after you customize the network settings, or you can do the reset from the guest operating system, as with a physical machine. The way to reset a NIC differs, depending on the operating system.

For information about the `RestartNetwork_Task` method, see [Restarting the Guest Network After Customization](#).

Resetting the Network Stack in a Linux Virtual Machine

When you change network settings in an instant clone, you must reset the network stack of the virtual NIC to adopt the new settings. For a Linux guest operating system, you must unbind and rebind the network driver so it adopts a new MAC address.

The following shell script rebinds all network drivers:

```
#!/bin/bash
for NETDEV in /sys/class/net/*
do
    DEV_LABEL=`basename `readlink -f $NETDEV/device``
    DEV_DRVR=`basename `readlink -f $NETDEV/device/driver``
    echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRVR/unbind
    echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRVR/bind
done
```

If the guest operating system was configured to use DHCP, you must also force a DHCP refresh after the MAC addresses refresh has completed.

Tip: Avoid resetting the network stack in the guest operating system while you are working over a network connection, such as SSH. You can run the shell commands to reset the stack as part of a customization script installed in the guest operating system of the source virtual machine before running the instant clone operation.

Resetting the Network Stack in a Windows Virtual Machine

When you change network settings in a Windows virtual machine, you must disable and re-enable the network object. There are several ways to do that in Windows.

Reset the Network Stack in a Windows Virtual Machine Using the Control Panel

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. You can do this using the Control Panel.

Procedure

- 1 In the Control Panel, navigate to **Control Panel > Networks and Internet > Network Connections**.
- 2 Right-click the network adapter icon and choose **disable**.
- 3 Right-click the network adapter icon and choose **enable**.
- 4 Repeat these steps for each network adapter.

Reset the Network Stack in a Windows Virtual Machine Using PowerShell

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task with PowerShell, use the `Restart-NetAdapter` cmdlet.

The cmdlet is described in more detail at <https://technet.microsoft.com/en-us/itpro/powershell/windows/netadapter/restart-netadapter>.

Procedure

- 1 List the virtual network adapters in the guest operating system.

```
Get-NetAdapter
```

The names of the virtual network adapters appear in the first column of output.

- 2 Restart the network adapter representing the external interface.

```
Restart-NetAdapter -Name "Ethernet0"
```

- 3 If you have more than one virtual interface, repeat the previous step for each one.

Reset the Network Stack in a Windows Virtual Machine Using netsh

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task, run `netsh` with `interface` command.

Run the following commands in a command prompt (Run > `cmd`).

Procedure

- 1 List the network interfaces.

```
netsh interface show interface
```

- 2 Choose the interface that represents the external network connection and disable it.

```
netsh interface set interface name="Ethernet0" admin=DISABLED
```

- 3 Re-enable the interface with a similar command.

```
netsh interface set interface name="Ethernet0" admin=ENABLED
```

- 4 Repeat the previous two commands if you have more than one NIC that needs to be reset.

Encryption in vSphere protects virtual machines, virtual disks, and related files. First you set up a trusted connection between vCenter Server and a key management server (KMS), then vCenter Server can retrieve keys from the KMS as needed. Physical device security and advanced options are additional concerns.

Various aspects of virtual machine encryption are handled differently. You manage setup of the KMS trusted connection and perform most encryption workflows from the vSphere Client. You manage automation of some advanced features using the vSphere Web Services SDK, as discussed in this chapter. You use the `crypto-util` command-line tool directly on ESXi hosts for some special cases, for example, to decrypt the core dumps in a vm-support bundle.

This chapter includes the following topics:

- [How Virtual Machine Encryption Protects a Datacenter](#)
- [vSphere Virtual Machine Encryption Components](#)
- [Prerequisites and Required Privileges for Encryption Tasks](#)
- [API Methods for vSphere Virtual Machine Encryption](#)
- [Workflows for vSphere Virtual Machine Encryption](#)
- [Best Practices for Virtual Machine Encryption](#)
- [Configuring Advanced Options](#)
- [Physical Platform Resources](#)

How Virtual Machine Encryption Protects a Datacenter

With vSphere virtual machine encryption, you can create encrypted virtual machines and encrypt existing ones. Because all virtual machine files with sensitive information are encrypted, the virtual machine is protected. Only administrators with encryption privileges can perform encryption and decryption tasks.

What Keys are Used

Two types of keys are used for encryption.

- The ESXi host generates and uses internal keys to encrypt virtual machines and disks. These keys are used as the disk encryption key (DEK) and are XTS-AES-256 keys.
- The key management server (KMS) sends keys to the vCenter Server upon request. These keys are used as the key encryption key (KEK) and are AES-256 keys. vCenter Server stores only the ID of each KEK, but not the key itself.
- ESXi hosts use the KEK to encrypt their internal keys, and store only the encrypted internal keys on disk, but not the KEK itself. When an ESXi host reboots, vCenter Server requests the necessary KEKs by sending the corresponding IDs to the KMS, and upon receipt, make the KEKs available to the ESXi host, which can then decrypt its internal keys as needed.

What Is Encrypted

Virtual machine encryption supports encrypting virtual machine files, virtual disk files, and core dump files.

Virtual Machine Files

Most virtual machine files, in particular guest data that are not stored in the VMDK file, are encrypted. This set of files includes but is not limited to the NVRAM (memory), VSWP (swap), and VMSN (snapshot) files. The key that vCenter Server retrieves from the KMS unlocks an encrypted bundle in the VMX file that contains internal keys and other secrets.

If you use the vSphere Client to create an encrypted virtual machine, all virtual disks are encrypted by default. For other encryption tasks, such as encrypting an existing virtual machine, you can encrypt and decrypt virtual disks separate from virtual machine files.

Note You cannot associate an encrypted virtual disk with an unencrypted virtual machine.

Virtual Disk Files

Data in an encrypted virtual disk (VMDK) file are never written in cleartext to storage or physical disk, and is never transmitted over the network in cleartext. The VMDK descriptor file is mostly cleartext, but contains a key ID for the KEK and the internal key (DEK) in the encrypted bundle.

You can use the vSphere API to perform either a shallow recrypt operation with a new KEK, or a deep recrypt operation with new internal keys.

Core Dump Files

Core dumps on an ESXi host that has encryption mode enabled are always encrypted. You can decrypt and password protect ESXi core dumps using the `crypto-util` command-line tool on the ESXi host.

Note Core dumps on the vCenter Server (Appliance) are not encrypted. Be sure to protect access to all vCenter Server systems.

What Is Not Encrypted

Some files that are associated with a virtual machine are not encrypted or partially encrypted.

Log Files

Log files are not encrypted because they should not contain sensitive data.

Virtual Machine Configuration Files

Most of the virtual machine configuration information, stored in the VMX and VMDS files, is not encrypted. Information about the KMS and the key ID is visible in those files.

Virtual Disk Descriptor File

To support disk management without a key, most of the virtual disk descriptor file is not encrypted.

Who Can Perform Cryptographic Operations

Only users who are assigned the Cryptographic Operations privileges can perform cryptographic operations. The privilege set is fine grained; see the *vSphere Security* guide. The default Administrator system role includes all Cryptographic Operations privileges. A new system role, No Cryptography Administrator, supports all Administrator privileges except for the Cryptographic Operations privileges.

You can create additional custom roles, for example, to allow a group of users to encrypt virtual machines but to prevent them from decrypting virtual machines.

For a full list of privileges, see the section “Cryptographic Operations Privileges” in the *vSphere Security* manual.

How Can I Perform Cryptographic Operations

The vSphere Client supports many cryptographic operations. For other tasks, you must use the API.

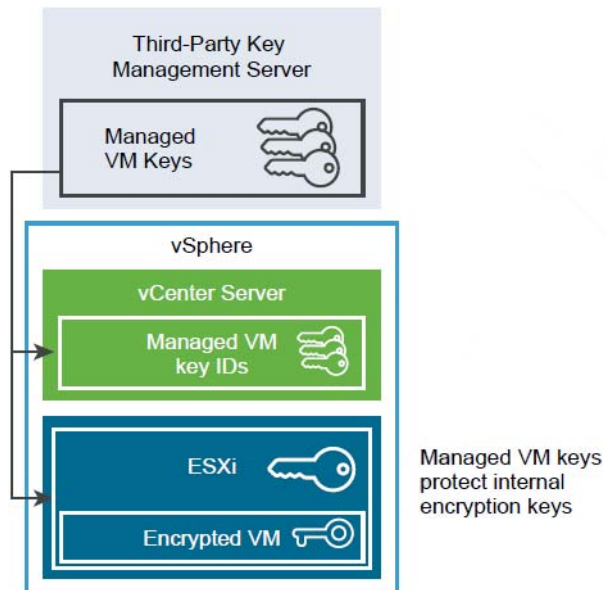
Table 14-1. Interfaces for performing cryptographic operations

Interface	Operations	Information
vSphere Client	Create encrypted virtual machines Encrypt and decrypt virtual machines	The <i>vSphere Security</i> guide.
vSphere Web Services SDK	Create encrypted virtual machines Encrypt and decrypt virtual machines Perform a deep reencrypt of virtual machines (use a different KEK and DEK). Perform a shallow reencrypt of virtual machines (use a different KEK).	This chapter.
<code>crypto-util</code>	Decrypt encrypted core dumps, check whether files are encrypted, and perform other management tasks directly on the ESXi host,	Command-line help and the <i>vSphere Security</i> guide.

vSphere Virtual Machine Encryption Components

An external key management server (KMS), the vCenter Server system, and ESXi hosts all contribute to the vSphere virtual machine encryption solution.

Figure 14-1. vSphere Virtual Machine Encryption Architecture



Key Management Server

The vCenter Server requests keys from an external KMS. The KMS generates and stores the keys, and passes them to vCenter Server for distribution.

You can use the vSphere Client or the vSphere API to add a cluster of KMS instances to the vCenter Server system. If you use multiple KMS instances in a cluster, all instances must be from the same vendor and must replicate keys.

If your environment uses different KMS vendors in different environments, you can add a KMS cluster for each KMS and specify a default KMS cluster. The first cluster that you add becomes the default cluster. You can explicitly change the default later.

As a KMS client, vCenter Server uses the Key Management Interoperability Protocol (KMIP) and makes it easy to use the KMS of your choice.

vCenter Server

Only vCenter Server has credentials for logging in to the KMS. The ESXi hosts do not have those credentials. The vCenter Server obtains keys from the KMS and pushes them to the ESXi hosts. The vCenter Server does not store the KMS keys, it merely keeps a list of key IDs.

The vCenter Server checks the privileges of users who perform cryptographic operations. You can use the vSphere Client to assign cryptographic privileges or to assign the No cryptography administrator custom role to groups of users. See [Prerequisites and Required Privileges for Encryption Tasks](#).

The vCenter Server adds cryptography events to the list of events that you can view and export from the vSphere Client Event Console. Each event includes the user, time, key ID, and cryptographic operation.

ESXi Hosts

ESXi hosts are responsible for several aspects of the encryption workflow.

- Accepting keys and storing them in memory (never on disk). If a host has encryption mode enabled, and the current user's role includes cryptographic operations privileges, vCenter Server pushes keys to the ESXi host upon request. See [Prerequisites and Required Privileges for Encryption Tasks](#).
- Ensuring that guest data for encrypted virtual machines are encrypted when stored on disk.
- Ensuring that guest data for encrypted virtual machines are never sent unencrypted over the network.

The keys that the ESXi host generates are called internal keys in this document. These keys are typically act as data encryption keys (DEKs).

Encryption Process Flow

After vCenter Server is connected to the KMS, users with the required privileges can create encrypted virtual machines and disks. Those users can also perform other encryption tasks such as encrypting existing virtual machines and decrypting encrypted virtual machines. During the encryption process, different components interact as follows.

- 1 When the user performs an encryption task, for example, creating an encrypted virtual machine, vCenter Server requests a new key from the default KMS. This key will be used as the KEK.

- 2 The vCenter Server stores the key ID and passes the key to an ESXi host. If the ESXi host is part of a cluster, vCenter Server sends the KEK to each host in the cluster. The key itself is not stored on the vCenter Server system. Only the key ID is stored.
- 3 The ESXi host generates internal keys (DEKs) for the virtual machine and its disks, using the KEK that it received from vCenter Server to encrypt the internal keys. The internal keys are kept in memory only. Only the encrypted data are stored on disk.
- 4 The ESXi host encrypts the virtual machine and its disks with the encrypted internal key.

ESXi hosts that have the KEK and can access the encrypted key file can perform operations on an encrypted virtual machine or disk. Because they come from the KMS, ESXi hosts can use the same KEK across reboots.

If you later want to decrypt a virtual machine, you change its storage policy either for the virtual machine or for its disks. If you want to decrypt individual components, first decrypt selected disks, then decrypt the virtual machine by changing the storage policy for VM Home. Both keys are required for decryption of each component, virtual disks or VM Home.

When you encrypt an existing virtual machine, you need at least twice the space that the virtual machine is currently using, in most cases.

Prerequisites and Required Privileges for Encryption Tasks

Users who perform encryption related tasks must have the appropriate privileges. Additional privileges are required if virtual machine encryption tasks require changing the host encryption mode. An extensive number of Cryptographic Operations privileges allow fine-grained control.

Encryption tasks are possible only in environments that include a vCenter Server. Additionally, the ESXi host must have encryption mode enabled for most encryption tasks. The user who performs an encryption task must have the appropriate privileges. Additional privileges are required if virtual machine encryption tasks require changing the host encryption mode. An extensive number of Cryptographic Operations privileges allow fine-grained control.

Cryptography Privileges and Roles

By default, the user with the vCenter Server Administrator role has all Cryptographic Operations privileges. You can assign the No cryptography administrator role to all vCenter Server administrators who do not need cryptographic privileges.

The user with the vCenter Server Administrator role has all privileges by default. You can assign the No cryptography administrator role to vCenter Server users who do not need Cryptographic Operations privileges. The No cryptography administrator lacks the following privileges for cryptographic operations:

- Add Cryptographic Operations privileges
- Global.Diagnostics
- Host.Inventory.Add host to cluster

- `Host.Inventory.Add` standalone host
- `Host.Local operations.Manage` user groups

To further limit what users can do, you can clone the No cryptography administrator role and create a custom role with only some of the Cryptographic Operations privileges. For example, you can create a role that allows users to encrypt but not to decrypt virtual machines, or that does grant privileges for management operations. See the *vSphere Security* manual for details.

Host Encryption Mode

You can encrypt virtual machines only if host encryption mode is enabled for the ESXi host. Host encryption mode is often enabled automatically, but it can be enabled explicitly.

You can check and explicitly set the current host encryption mode from the vSphere Client or by using the vSphere API; see [API Methods to Prepare an ESXi Host](#).

After host encryption mode is enabled, it cannot be disabled easily. See the *vSphere Security* guide for details.

Automatic changes occur when encryption operations attempt to enable host encryption mode. For example, suppose that you add an encrypted virtual machine to an ESXi host, and host encryption mode is not enabled. If you have the required privileges on the host, encryption mode automatically changes to enabled.

Assume a cluster that includes three ESXi hosts, host A, B, and C. You add an encrypted virtual machine to host A. What happens depends on several factors. If all three hosts have encryption enabled, you can create an encrypted virtual machine if you have `Encrypt new` privileges. If none of the hosts has encryption enabled, and you have `Register host` privileges on host A, then the virtual machine creation process enables host encryption on that host; otherwise an error results. The scenario is more complicated if host B or C is not enabled for encryption; see the *vSphere Security* guide for details.

Encrypted vSphere vMotion

vSphere vMotion always tries to use encryption when migrating encrypted virtual machines. You cannot disable encrypted vMotion for encrypted virtual machines in a cluster.

For virtual machines that are not encrypted, you can set encrypted vMotion to `Opportunistic` (use encrypted vMotion if supported) or `Required` (do not migrate if unsupported). See [Encrypted vSphere vMotion](#).

API Methods for vSphere Virtual Machine Encryption

Methods for managing vSphere virtual machine encryption are in the vSphere API, part of the Web Services SDK. In addition, you need a few Storage Policy APIs to create and assign encryption storage policies.

vSphere API Methods for KMS Management

Cryptographic key management interfaces are defined in the following hierarchy.

- **CryptoManager** – managed object, parent of **CryptoManagerKmp**.

CryptoManager defines methods that are covered in [vSphere API Methods for Cryptographic Operations](#).

- **CryptoManagerKmp** – managed object for handling the cryptographic keys.
 - **CertificateInfo** – basic information of a certificate.
 - **ClusterStatus** – status of a KMS cluster.
 - **ServerCertInfo** – information about the KMS certificate.
 - **ServerStatus** – status of a KMS.

CryptoManagerKmp defines the following methods:

- **String generateClientCsr**(cluster) – generate a certificate signing request with its private key.
- **String generateSelfSignedClientCert**(cluster) – generate self-signed client certificate with its private key.
- **KmpClusterInfo[] kmipServers**() – get a list of registered KMS, grouped by clusters.
- **KmpClusterInfo[] listKmpServers**(limit) – get a list of registered KMS.
- **void markDefault**(clusterId) – set the default KMS cluster.
- **void registerKmpServer**(server) – register a KMS.
- **void removeKmpServer**(clusterId, serverName) – remove a KMS, even if in use.
- **String retrieveClientCert**(cluster) – get the client certificate of the KMS cluster.
- **String retrieveClientCsr**(cluster) – get the generated client certificate signing request.
- **ServerCertInfo retrieveKmpServerCert**(keyProvider, server) – get the server certificate.
- **ClusterStatus[] retrieveKmpServersStatus**(clusters[]) – get status of the KMS instances.
- **String retrieveSelfSignedClientCert**(cluster) – get generated self signed client certificate.
- **void updateKmpServer**(server) – update a KMS.
- **void updateKmsSignedCsrClientCert**(cluster, certificate) – Set KMS signed certificate as the client certificate for a KMS cluster.

- `void updateSelfSignedClientCert(cluster, certificate)` – set a self-signed certificate as the client certificate for the KMS cluster.
- `void uploadClientCert(cluster, certificate, privateKey)` – set a client certificate with private key for the KMS cluster.
- `void uploadKmpServerCert(cluster, certificate)` – upload a server certificate.
- `CryptoKeyResult generateKey(keyProvider)` – generate a new encryption key.

The `generateKey` method is out of alphabetic order because it is called after the others.

API Methods to Prepare an ESXi Host

Encryption interfaces for ESXi hosts are additions to a previously existing managed object, `HostSystem`.

- `HostSystem` – managed object providing access to the ESXi host, including these additions:
 - `HostCryptoState` – enumeration indicating whether the ESXi host is incapable of, prepared for encryption mode, or cryptography safe with its host key already set. Safe means that the ESXi host can encrypt virtual machines and will not leak keys.
 - `ConfigureCryptoKey` – vCenter Server calls this method to set or change the key used for core dump encryption, and to place the host in safe mode. It is like calling `PrepareCrypto` and `EnableCrypto` in quick succession. Not supported if called on an ESXi host. As of vSphere 7.0 you can disable encryption on a host, after next reboot, by setting `pendingIncapable` in `HostCryptoState`.
 - `PrepareCrypto` – this method puts the ESXi host in prepared mode so it can receive sensitive data, after being enabled.
 - `EnableCrypto` – this method sets or changes the key for core dump encryption and puts the ESXi host in safe mode. It must be called in sequence after `PrepareCrypto`.

The following data objects are used by the `ConfigureCryptoKey` and `EnableCrypto` methods above, and by the `CryptoManager` methods below in [vSphere API Methods for Cryptographic Operations](#).

- `CryptoKeyPlain` – data object representing a plain text cryptographic key.
- `CryptoKeyId` – data object representing a cryptographic key.
- `CryptoKeyResult` – data object representing the result of a cryptographic key operation.

vSphere API Methods for Cryptographic Operations

Cryptographic operations are defined in the following hierarchy.

- `CryptoManager` – managed object for handling cryptographic keys.

`CryptoManager` defines the following methods:

- `void addKey(CryptoKeyPlain key)` – add plain key to the vCenter Server.

- `CryptoKeyResult[]` **addKeys**(`CryptoKeyPlain[]` keys) – add multiple plain keys to vCenter.
- `boolean` **enabled**() – indicate if the encryption feature is enabled.
- `CryptoKeyId[]` **listKeys**(`int` limit) – list keys.
- `void` **removeKey**(`CryptoKeyId` key, `boolean` force) – remove a key (only its ID is needed).
- `CryptoKeyResult[]` **removeKeys**(`CryptoKeyId[]` keys, `boolean` force) – multiple keys.
- `VirtualMachineConfigSpec` – previously existing data object passed as parameter to `CreateVM_Task` and `ReconfigVM_Task`. One of its newly added properties is `crypto`, a `CryptoSpec` with one of the following options, which is inherited by all virtual disks and virtual machine configuration files (VM home).
 - `CryptoSpecEncrypt` – indicates that the virtual machine should be encrypted.
 - `CryptoSpecDecrypt` – indicates that the virtual machine should be decrypted.
 - `CryptoSpecDeepReencrypt` – indicates that all KEKs and DEKs should be replaced.
 - `CryptoSpecShallowReencrypt` – indicates that only KEKs should be replaced.
 - `CryptoSpecNoOp` – indicates that encryption settings should not be changed.
 - `CryptoSpecRegister` – indicates that the operation should send keys but should not modify the encryption settings of the virtual machine or virtual disk. When an encrypted disk is hot attached, the program must pass `CryptoSpecRegister` with the key ID that encrypted the disk. The key can be obtained from the Datastore Browser.

These data objects are informational properties of `VMConfigFileInfo` and `VMDiskFileInfo`, respectively. They can be used to check whether the VM home and its virtual disks are encrypted.

- `VmConfigFileEncryptionInfo` – the encryption information of a virtual machine configuration.
- `VmDiskFileEncryptionInfo` – the encryption information of a virtual disk.

The enumeration `EncryptedVMotionModes` controls whether encrypted vMotion is disabled, required, or opportunistic (fall back to unencrypted vMotion if necessary, the default option).

More information about the encryption interfaces is available in vSphere Management SDK Documentation under vSphere Web Services SDK, in the *vSphere API Reference*.

SPBM API Methods for Encryption

The managed object `PbmProfileProfileManager` (`pbm.profile.ProfileManager`) provides methods to set encryption related storage policies, including create and update.

- `PbmCreate(createSpec)` – create a capability-based storage profile defining storage requirements.
- `PbmUpdate(profileId, createSpec)` – update a storage profile with new capability requirements.

Workflows for vSphere Virtual Machine Encryption

The sections below present vSphere API methods for key acquisition and encryption.

Set Up the Key Management Server Cluster

Before you can start encrypting virtual machines, you must set up the Key Management Server (KMS) cluster. This includes adding the KMS, establishing trust with the KMS, and setting the default KMS cluster. The KMS must support the KMIP (Key Management Interoperability Protocol) 1.1 standard, and it must be capable of acting as a symmetric key foundry and server.

Add Key Management Server(s) to Your Datacenter

Your organization's security administrator may be responsible for this task. This person will need to provide the cluster name (if it already exists), the KMS server name, its IP address, its connection port number, and possibly a proxy address, proxy port number, KMS user name, and corresponding password.

The `CryptoManager.java` code to add a KMS and set the default KMS cluster appears in [CryptoManager Java program to add KMS and set default cluster](#). The user running this program must have `Cryptographic operations.Manage key servers` privileges.

Establish a Trusted Connection by Exchanging Certificates

There are four ways to establish trust between a KMS and vCenter Server. Different KMS vendors require different ways.

Java examples for these four ways follow.

- 1 Upload your Root CA Certificate to the KMS. Obtain it manually and upload with the self-signed call.
- 2 Upload a self-signed vCenter Certificate to the KMS.
- 3 Have vCenter Server generate a certificate signing request (CSR), which the KMS signs and returns.
- 4 Download a security certificate and private key generated by the KMS to vCenter Server.

Example: CryptoManager Java program to add KMS and set default cluster

```

package com.vmware.general;
import com.vmware.common.annotations.Action;
import com.vmware.common.annotations.Option;
import com.vmware.common.annotations.Sample;
import com.vmware.connection.ConnectedVimServiceBase;
import com.vmware.vim25.CryptoKeyId;
import com.vmware.vim25.CryptoKeyResult;
import com.vmware.vim25.CryptoManagerKmpServerCertInfo;
import com.vmware.vim25.KeyProviderId;
import com.vmware.vim25.KmpServerInfo;
import com.vmware.vim25.KmpServerSpec;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.RuntimeFaultFaultMsg;
/**
 * CryptoManager. Demonstrates uses of the CryptoManager API. Parameters:
 * url [required] : web service url, for example https://10.9.8.7/sdk
 * username [required] : username for the authentication
 * password [required] : corresponding password
 * Command line to run CryptoManager code:
 * run.bat com.vmware.general.CryptoManager ^
 * --url webserviceurl --username name --password pass ^
 * --kmsname kms --kmsip ipaddr --kmsclusterid providerId
 */
@Sample(name = "CryptoManager", description = "Demonstrates uses of the CryptoManager API")
public class CryptoManager extends ConnectedVimServiceBase {
    private ManagedObjectReference cryptoManager = null;
    private KmpServerSpec kmpSpec = null;
    private String kmsName;
    private String kmsIp;
    private int kmsPort = 5696; // default
    private String kmsClusterId;

    public void initCryptoManager() throws RuntimeFaultFaultMsg {
        if (serviceContent != null) {
            cryptoManager = serviceContent.getCryptoManager();
            if (cryptoManager == null) {
                throw new RuntimeFaultFaultMsg("CryptoManager could not be obtained", null);
            }
        } else {
            throw new RuntimeFaultFaultMsg("ServiceContent could not be obtained", null);
        }
    }

    public void registerKmpServer() throws RuntimeFaultFaultMsg {
        KmpServerInfo serverInfo = new KmpServerInfo(); // Create KMS info
        serverInfo.setName(kmsName); // Set the name of your KMS here
        serverInfo.setAddress(kmsIp); // Set the IP addr of your KMS
        serverInfo.setPort(kmsPort); // Set KMS port, if different from default
        KeyProviderId providerId = new KeyProviderId(); // Set the name of KMS cluster here
        providerId.setId("KMScluster");
        kmpSpec = new KmpServerSpec();
        kmpSpec.setInfo(serverInfo); // KMS spec with server and cluster ID
        kmpSpec.setClusterId(providerId);
        vimPort.registerKmpServer(cryptoManager, kmpSpec); // Register server
    }
}

```

```

    }
    public void trustKmip() throws RuntimeFaultFaultMsg { // Get KMS certificate
        CryptoManagerKmipServerCertInfo certInfo = vimPort.retrieveKmipServerCert(
            cryptoManager, kmipSpec.getClusterId(), kmipSpec.getInfo());
        // Upload retrieved certificate to vCenter Server and trust it
        vimPort.uploadKmipServerCert(
            cryptoManager, kmipSpec.getClusterId(), certInfo.getCertificate());
    }
    public void establishTrust() throws RuntimeFaultFaultMsg {
        // Make KMS trust vCenter Server by uncommenting and calling one of these lines
        // - establishTrustUsingSelfSignedCert() - see Example 12-2 for source code
        // - establishTrustUsingSignedCsr() - see Example 12-3 for source code
        // - establishTrustUsingCertAndKey() - see Example 12-4 for source code
        trustKmip(); // Now make the vCenter Server trust KMS
    }
    public void setDefaultKmipCluster() throws RuntimeFaultFaultMsg {
        vimPort.markDefault(cryptoManager, kmipSpec.getClusterId()); // Mark cluster as default
    }
    public void generateNewKey() throws RuntimeFaultFaultMsg {
        CryptoKeyResult keyResult = vimPort.generateKey(cryptoManager, kmipSpec.getClusterId());
        CryptoKeyId keyId = keyResult.getKeyId(); // Generate new key for encryption
    }
    @Action
    public void action() throws RuntimeFaultFaultMsg {
        initCryptoManager();
        registerKmipServer();
        establishTrust();
        setDefaultKmipCluster();
        generateNewKey();
    }
    @Option(name = "kmsname", description = "Name of the KMS", required = true)
    public void setKMSName(String name) {
        this.kmsName = name;
    }
    @Option(name = "kmsip", description = "IP address of the KMS", required = true)
    public void setKMSIp(String ip) {
        this.kmsIp = ip;
    }
    @Option(name = "kmsport", description = "KMS port", required = false)
    public void setKMSPort(String port) {
        this.kmsPort = Integer.parseInt(port);
    }
    @Option(name = "kmsclusterid", description = "KMS cluster Id", required = true)
    public void setKMSClusterId(String clusterId) {
        this.kmsClusterId = clusterId;
    }
}

```

Example: Trust with self-signed certificate or Root CA certificate

This example method uploads a self-signed vCenter certificate, or the Root CA certificate, to the KMS.

```

public void establishTrustUsingSelfSignedCert() throws RuntimeFaultFaultMsg {
    // Generate self-signed cert, or obtain the Root CA certificate

```



```
String selfSignedCert = vimPort.generateSelfSignedClientCert(
    cryptoManager, kmipSpec.getClusterId());
// Follow steps for KMS to trust self-signed or Root CA cert, update vCenter to use it
vimPort.updateSelfSignedClientCert(
    cryptoManager, kmipSpec.getClusterId(), selfSignedCert);
}
```

Example: Trust with CSR then downloading KMS signed certificate

This example method generates a CSR and downloads the KMS signed certificate onto vCenter Server.

```
public void establishTrustUsingSignedCsr() throws RuntimeFaultFaultMsg {
    // Generate a Certificate Signing Request
    String csr = vimPort.generateClientCsr(cryptoManager, kmipSpec.getClusterId());
    String signedCert = null;
    // Follow steps for your KMS to sign CSR and get the signedCert to update on vCenter
    vimPort.updateKmsSignedCsrClientCert(
        cryptoManager, kmipSpec.getClusterId(), signedCert);
}
```

Example: Trust by downloading KMS certificate and private key

This example method downloads a certificate and private key generated by the KMS to vCenter Server.

```
public void establishTrustUsingCertAndKey() throws RuntimeFaultFaultMsg {
    String certFromKms = null;
    String privateKeyKms = null;
    // Follow steps for KMS to generate certificate and private key (certFromKms,
privateKeyKms)
    vimPort.uploadClientCert(
        cryptoManager, kmipSpec.getClusterId(), certFromKms, privateKeyKms);
}
```

Create an Encryption Storage Policy

Before any virtual machines can be encrypted, vCenter Server must contain an encryption storage policy. The policy only needs to be created once; it can be assigned to any virtual machines that you want to encrypt.

Example: Java program to set storage policy for encryption

```
package com.vmware.spbm.samples;
import java.util.ArrayList;
import java.util.List;
import com.vmware.common.annotations.Action;
import com.vmware.common.annotations.Option;
import com.vmware.common.annotations.Sample;
import com.vmware.pbm.InvalidArgumentFaultMsg;
import com.vmware.pbm.PbmCapabilityConstraintInstance;
import com.vmware.pbm.PbmCapabilityInstance;
import com.vmware.pbm.PbmCapabilityMetadata;
```

```

import com.vmware.pbm.PbmCapabilityMetadataPerCategory;
import com.vmware.pbm.PbmCapabilityProfileCreateSpec;
import com.vmware.pbm.PbmCapabilityPropertyInstance;
import com.vmware.pbm.PbmCapabilityPropertyMetadata;
import com.vmware.pbm.PbmCapabilitySubProfile;
import com.vmware.pbm.PbmCapabilitySubProfileConstraints;
import com.vmware.pbm.PbmCapabilityVendorNamespaceInfo;
import com.vmware.pbm.PbmCapabilityVendorResourceTypeInfo;
import com.vmware.pbm.PbmDuplicateNameFaultMsg;
import com.vmware.pbm.PbmFaultProfileStorageFaultFaultMsg;
import com.vmware.pbm.PbmProfileId;
import com.vmware.pbm.PbmServiceInstanceContent;
import com.vmware.spbm.connection.ConnectedServiceBase;
import com.vmware.spbm.connection.helpers.PbmUtil;
import com.vmware.vim25.ManagedObjectReference;
import com.vmware.vim25.RuntimeFaultFaultMsg;
/**
 * CreateVMEncryptionProfile
 * Create new Storage Profile with one rule-set based on vmwarevmcrypt capabilities.
 * Parameters:
 * vcurl      [required] : web service url, for example https://10.9.8.7/sdk
 * username   [required] : username for the authentication
 * password   [required] : corresponding password
 * profilename [required] : name of the storage profile
 * Command Line:
 * run.bat com.vmware.spbm.samples.CreateVMEncryptionProfile --vcurl [webserviceurl] ^
 *      --username [username] --password [password] --profilename [Storage Profile Name]
 */
@Sample(name = "CreateVMEncryptionProfile",
description = "Create a new storage profile with "
              + "one rule-set based on vmwarevmcrypt capabilities.")
public class CreateVMEncryptionProfile extends ConnectedServiceBase {
    private PbmServiceInstanceContent spbmisc;
    private String profileName;

    // Build capability instance based on capability name associated with vmwarevmcrypt
    provider
    PbmCapabilityInstance buildCapability(String capabilityName,
                                         List<PbmCapabilityMetadataPerCategory> metadata)
    throws InvalidArgumentFaultMsg {
        // Create Property Instance with capability vmwarevmcrypt
        PbmCapabilityMetadata capabilityMeta =
            PbmUtil.getCapabilityMeta(capabilityName, metadata);
        if (capabilityMeta == null)
            throw new InvalidArgumentFaultMsg("Specified Capability does not exist", null);
        // Create and associate Property Instances with a Rule
        PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();
        for (PbmCapabilityPropertyMetadata propMeta : capabilityMeta.getPropertyMetadata()) {
            PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();
            prop.setId(propMeta.getId());
            prop.setValue(propMeta.getDefaultValue());
            rule.getPropertyInstance().add(prop);
        }
        // Associate Rule with a Capability Instance
        PbmCapabilityInstance capability = new PbmCapabilityInstance();

```

```

        capability.setId(capabilityMeta.getId());
        capability.getConstraint().add(rule);
        return capability;
    }
    @Action
    public void createProfile() throws RuntimeFaultFaultMsg,
        com.vmware.pbm.RuntimeFaultFaultMsg, InvalidArgumentFaultMsg,
        PbmDuplicateNameFaultMsg, PbmFaultProfileStorageFaultFaultMsg {
        // Get PBM Profile Manager & Associated Capability Metadata
        spbmSvc = connection.getPbmServiceContent();
        ManagedObjectReference profileMgr = spbmSvc.getProfileManager();
        // Step 1: Check if there is a vmwarevmcrypt Provider
        Boolean encryptionCapable = false;
        List<PbmCapabilityVendorResourceTypeInfo> vendorInfo =
            connection.getPbmPort().pbmFetchVendorInfo(profileMgr, null);
        for (PbmCapabilityVendorResourceTypeInfo vendor : vendorInfo)
            for (PbmCapabilityVendorNamespaceInfo vnsi : vendor.getVendorNamespaceInfo())
                if (vnsi.getNamespaceInfo().getNamespace().equals("vmwarevmcrypt")) {
                    encryptionCapable = true;
                    break;
                }
        if (!encryptionCapable)
            throw new RuntimeFaultFaultMsg(
                "Cannot create storage profile. 'vmwarevmcrypt' Provider not found.", null);
        // Step 2: Get PBM Supported Capability Metadata
        List<PbmCapabilityMetadataPerCategory> metadata =
            connection.getPbmPort().pbmFetchCapabilityMetadata(profileMgr,
                PbmUtil.getStorageResourceType(),
                "com.vmware.iofilters");

        // Step 3: Add Provider Specific Capabilities
        List<PbmCapabilityInstance> capabilities = new ArrayList<PbmCapabilityInstance>();
        capabilities.add(buildCapability("vmwarevmcrypt@ENCRYPTION", metadata));
        // Step 4: Add Capabilities to a RuleSet
        PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();
        ruleSet.getCapability().addAll(capabilities);
        // Step 5: Add Rule-Set to Capability Constraints
        PbmCapabilitySubProfileConstraints constraints = new
PbmCapabilitySubProfileConstraints();
        ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));
        constraints.getSubProfiles().add(ruleSet);
        // Step 6: Build Capability-Based Profile
        PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();
        spec.setName(profileName);
        spec.setDescription("Storage Profile in SDK. Rule based on Encryption capability");
        spec.setResourceType(PbmUtil.getStorageResourceType());
        spec.setConstraints(constraints);
        // Step 7: Create Storage Profile
        PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);
        System.out.println("Profile " + profileName + " with ID: " + profile.getUniqueId());
    }
    @Option(name = "profileName", description = "Name of the storage profile", required = true)
    public void setProfileName(String profileName) {
        this.profileName = profileName;
    }
}

```

Create an Encrypted Virtual Machine

The Web Services SDK provides Java code to create a virtual machine. The Java code can be modified to create an encrypted virtual machine. Steps below show how to implement virtual machine encryption in the `VMCreate.java` sample program. The `crypto` property in `VirtualMachineConfigSpec` should be set to `CryptoSpecEncrypt`, and an encryption key provided. The storage profile must have been previously set to specify encryption. If the `CryptoSpec` is unset, but a storage profile with encryption is set, vCenter Server automatically generates new keys and sets `CryptoSpec`, if a default KMS has been configured.

To enhance the `com.vmware.spbm.samples.VMCreate` program for encryption, follow these steps:

- 1 Import the following classes:

```
import com.vmware.vim25.CryptoKeyId;
import com.vmware.vim25.CryptoSpecEncrypt;
import com.vmware.vim25.KeyProviderId;
import com.vmware.vim25.VirtualDeviceConfigSpecBackingSpec;
```

- 2 Declare the following variables:

```
private CryptoKeyId cryptoKeyId;
private String keyId;
private String providerId;
```

- 3 In the `createVirtualDisk()` function, find the following line:

```
diskSpec.setOperation(VirtualDeviceConfigSpecOperation.ADD);
```

- 4 In the `createVirtualDisk()` function, add the following code after the line you found in the previous step:

```
if ((keyId != null) && (providerId != null)) {
    CryptoSpecEncrypt cSpec = new CryptoSpecEncrypt();
    VirtualDeviceConfigSpecBackingSpec backingSpec =
        new VirtualDeviceConfigSpecBackingSpec();
    cSpec.setCryptoKeyId(buildCryptoKeyId());
    backingSpec.setCrypto(cSpec);
    diskSpec.setBacking(backingSpec);
}
```

- 5 In the `createVmConfigSpec()` function, find the following line:

```
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
```

- 6 In the `createVmConfigSpec()` function, add the following code after the line you found in the previous step:

```
if ((keyId != null) && (providerId != null)) {
    CryptoSpecEncrypt cryptoSpecForVMHome = new CryptoSpecEncrypt();
    cryptoSpecForVMHome.setCryptoKeyId(buildCryptoKeyId());
    configSpec.setCrypto(cryptoSpecForVMHome);
}
```

- 7 Add the following options toward the end of file:

```
@Option(name = "keyid", description = "Key Id", required = false)
public void setKeyId(String kid) {
    this.keyId = kid;
}
@Option(name = "providerid", description = "Cluster/Provider Id", required = false)
public void setProviderId(String pid) {
    this.providerId = pid;
}
```

The SPBM sample code is in the Storage Policy SDK, not in the vSphere Web Services SDK.

Clone an Encrypted Virtual Machine

Set the `crypto` property in `RelocateSpec` at `cloneSpec.location`. The virtual machine must be powered off, with no existing snapshots. The encrypted virtual machine will be cloned, still encrypted.

Example: Clone an encrypted virtual machine

```
// Step 5: Create Specs
VirtualMachineRelocateSpec relocSpec = new VirtualMachineRelocateSpec();
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
// Step 6: Associate Storage Profile
relocSpec.getProfile().add(getVMDefinedProfileSpec(profileName));
cloneSpec.setConfig(configSpec);
cloneSpec.setLocation(relocSpec);
cloneSpec.setPowerOn(false);
cloneSpec.setTemplate(false);
// Step 7: Clone VM
ManagedObjectReference
    cloneTask = connection.getVimPort().cloneVMTask(vmRef, vmFolderRef, cloneName,
cloneSpec);
```

Encrypt an Existing Virtual Machine or Disk

First set the `crypto` property in the `VirtualMachineConfigSpec` to `CryptoSpecEncrypt` and provide a key. The virtual machine must be powered off, with no existing snapshots.

Storage profiles must also be set to specify encryption. If the `CryptoSpec` is unset, but a storage profile with encryption is set, the vCenter Server automatically generates new keys and sets `CryptoSpec`, if the default KMS cluster has been configured.

Example: Encrypt a virtual machine based on storage profile

```
void Encrypt() throws Exception {
    // Create VirtualMachineConfigSpec
    VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
    // Create VirtualDeviceConfigSpec
    VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();
    // Get VirtualMachineProfileSpec for new encryption profile and name it encryptionProfile
    VirtualMachineProfileSpec encryptionProfile = new VirtualMachineDefinedProfileSpec();
    // Get VirtualDisk for disk to reconfigure as in com.vmware.vm.VMReconfig, name it disk
    diskSpec.setDevice(disk);
    diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);
    // Add encryption profile to VirtualDeviceConfigSpec
    diskSpec.getProfile().add(encryptionProfile);
    // Create CryptoSpec for encryption
    // Get Key Id from CryptoManager as keyId
    // See CryptoManager for details on generating or retrieving CryptoKeyId
    CryptoSpecEncrypt cryptoSpec = new CryptoSpecEncrypt();
    cryptoSpec.setCryptoKeyId(keyId);
    // Add CryptoSpecEncrypt to diskSpec backing
    VirtualDeviceConfigSpecBackingSpec backingSpec = new VirtualDeviceConfigSpecBackingSpec();
    backingSpec.setCrypto(cryptoSpec);
    diskSpec.setBacking(backingSpec);
    // When encrypting a VirtualDisk, the VM home needs to be encrypted also.
    // You can choose the same key to encrypt VM home and virtual disk, or use different keys.
    // Set cryptoSpec and profile for encrypting virtual machine home.
    vmConfigSpec.setCrypto(cryptoSpec);
    vmConfigSpec.getVmProfile().add(encryptionProfile);
    // Set the device changes
    vmConfigSpec.getDeviceChange().add(diskSpec);
    // Issue reconfigure - See reConfig() in com.vmware.vm.VMReconfig to reconfigure VM.
}
```

Decrypt an Encrypted Virtual Machine or Disk

This is similar to encrypting an existing virtual machine, but with a different `CryptoSpec`. First set the `crypto` property in the `VirtualMachineConfigSpec` to `CryptoSpecDecrypt`. The virtual machine must be powered off, and the storage profiles must be set not to specify encryption.

If the `CryptoSpec` is unset, but a storage profile without encryption is set, the vCenter Server sets `CryptoSpecDecrypt`. The `ConfigSpec.crypto` parameter must be set explicitly. Only encryption is deduced from the storage profile.

Example: Decrypt a virtual machine based on storage profile

```
void Decrypt() throws Exception {
    // Create VirtualMachineConfigSpec
    VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
```

```

// Create VirtualDeviceConfigSpec
VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();
// Create Empty VirtualMachineProfileSpec for decryption
VirtualMachineProfileSpec emptyProfile = new VirtualMachineEmptyProfileSpec();
// Get VirtualDisk for disk to reconfigure in com.vmware.vm.VMReconfig, name it disk.
diskSpec.setDevice(disk);
diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);
// Remove any encryption profile set for the disk
diskSpec.getProfile().add(emptyProfile);
// Create CryptoSpec for decryption
CryptoSpecDecrypt cryptoSpec = new CryptoSpecDecrypt();
// Add CryptoSpecEncrypt to diskSpec backing
VirtualDeviceConfigSpecBackingSpec backingSpec = new VirtualDeviceConfigSpecBackingSpec();
backingSpec.setCrypto(cryptoSpec);
diskSpec.setBacking(backingSpec);
// Decrypting virtual machine home is not necessary when decrypting a virtual disk.
// If no encrypted disk is present on virtual machine after that disk is decrypted,
// you can choose to decrypt virtual machine home.
// Set cryptoSpec and profile for decrypting virtual machine home
vmConfigSpec.setCrypto(cryptoSpec);
vmConfigSpec.getVmProfile().add(emptyProfile);
// Set the device changes
vmConfigSpec.getDeviceChange().add(diskSpec);
// Issue reconfigure - See reConfig() in com.vmware.vm.VMReconfig to reconfigure VM.
}

```

Encrypt with Different Keys for VM and Disk

This method shows how to use two different keys to encrypt the virtual machine (VM home) and its disk.

Example: Different encryption keys for VM home and virtual disk

```

void EncryptUsingDifferentKeys() throws Exception {
    // Create VirtualMachineConfigSpec and VirtualDeviceConfigSpec
    VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
    VirtualDeviceConfigSpec diskSpec = new VirtualDeviceConfigSpec();
    // Get VirtualMachineProfileSpec for encryption profile you created and name it
    VirtualMachineProfileSpec encryptionProfile = new VirtualMachineDefinedProfileSpec();
    // Get VirtualDisk for disk to be reconfigured as shown in VMReconfig and name it
    diskSpec.setDevice(disk);
    diskSpec.setOperation(VirtualDeviceConfigSpecOperation.EDIT);
    // Add encryption profile to VirtualDeviceConfigSpec
    diskSpec.getProfile().add(encryptionProfile);
    // Create CryptoSpec for disk encryption and get KeyId from CryptoManager
    // See CryptoManager for details on generating or retrieving CryptoKeyId
    CryptoSpecEncrypt cryptoSpecForDisk = new CryptoSpecEncrypt();
    cryptoSpecForDisk.setCryptoKeyId(keyIdForDiskEncryption);
    // Add CryptoSpecEncrypt to diskSpec backing
    VirtualDeviceConfigSpecBackingSpec backingSpec = new VirtualDeviceConfigSpecBackingSpec();
    backingSpec.setCrypto(cryptoSpecForDisk);
    diskSpec.setBacking(backingSpec);
    // When encrypting a VirtualDisk, the virtual machine home must also be encrypted.
}

```

```
// You can choose the same key to encrypt VM home and disk, or use different keys.
// Create CryptoSpec for VM Home encryption and get KeyId from CryptoManager.
CryptoSpecEncrypt cryptoSpecForVMHome = new CryptoSpecEncrypt();
cryptoSpecForVMHome.setCryptoKeyId(keyIdForVMHomeEncryption);
// Set cryptoSpec and profile for encrypting virtual machine home
vmConfigSpec.setCrypto(cryptoSpecForVMHome);
vmConfigSpec.getVmProfile().add(encryptionProfile);
// Set the device changes
vmConfigSpec.getDeviceChange().add(diskSpec);
}
```

Recrypt (Rekey) Encrypted Virtual Machines

You can recrypt virtual machines using either the vSphere API or the vSphere Client (as of 7.0.x). In the *vSphere Security* manual, recrypt is usually called rekey, or sometimes re-encrypt.

There are two kinds of recryption operations. Deep recrypt replaces all keys, rewriting encrypted data in a powered-off virtual machine and its disks. Shallow recrypt replaces only top-level keys and is comparatively fast.

For details on generating or retrieving the CryptoKeyId, see [CryptoManager code in CryptoManager Java program to add KMS and set default cluster](#).

Recrypt Only Key Encryption Keys

For shallow recrypt, which affects only the key encryption keys (KEKs), set the `crypto` property in the `VirtualMachineConfigSpec` to `CryptoSpecShallowRecrypt` and call the `Reconfigure` method.

Example: Shallow Recrypt

```
void ShallowRecrypt() throws Exception {
    // Shallow recrypt follows the same flow as encrypt. The two differences are:
    // - Instead of using a new encryption profile, just get the previously
    //   applied profile from the virtual machine to be reconfigured and use it.
    // - The type of CryptoSpec object created is CryptoSpecShallowRecrypt()
    // Create CryptoSpec for shallow recrypt
    // Get Key Id from CryptoManager as newKeyId
    CryptoSpecShallowRecrypt cryptoSpec = new CryptoSpecShallowRecrypt();
    cryptoSpec.setNewKeyId(newKeyId);
    // Follow steps from Encrypt()
}
```

Shallow recrypt can be done with the virtual machine powered on. Deep recrypt requires power off.

Recrypt Both Key and Disk Encryption Keys

For deep recrypt, which affects both KEKs and disk encryption keys (DEKs), set the `crypto` property in the `VirtualMachineConfigSpec` to `CryptoSpecDeepRecrypt` and call the `Reconfigure` method.

Example: Deep Recrypt

```
void DeepRecrypt() throws Exception {
    // Deep recrypt follows the same flow as encrypt. The two differences are:
    // - Instead of using a new encryption profile, just get the previously
    //   applied profile from the virtual machine to be reconfigured and use it.
    // - The type of CryptoSpec object created is CryptoSpecDeepRecrypt()
    // Create CryptoSpec for deep recrypt
    // Get Key Id from CryptoManager as newKeyId
    CryptoSpecDeepRecrypt cryptoSpec = new CryptoSpecDeepRecrypt();
    cryptoSpec.setNewKeyId(newKeyId);
    // Follow steps from Encrypt()
}
```

Query Crypto Key In-Use Status

Sometimes it is useful to determine key status, for example whether a key is available on vCenter Server, which virtual machines and ESXi hosts are using it, and if third party applications reference it.

As of vSphere 6.7 Update 2, the `queryCryptoKeyStatus` method is available to check use of a KMS key, such as if vCenter Server can access the key, if the key is being used by some virtual machines, or as a host key. The first parameter is a key ID or an array of key IDs to query. The second parameter is a bitmap defining items to check:

- 0x01 - ask if key data is available to vCenter Server
- 0x02 - query which virtual machines use this key
- 0x04 - check the ESXi hosts using this as a host key
- 0x08 - check third party programs using this key.

Key status results are returned in a `CryptoManagerKmp::CryptoKeyStatus` data object, called `keyStatusArray` in the code below. If your program queried multiple keys, an array of results is returned.

The `queryCryptoKeyStatus` method and its returned data object are new in vSphere 6.7 U2, so it does not work in earlier releases.

The following Java code checks all items above by passing 15, a bitwise OR of the settings above.

```
CryptoManagerKmpCryptoKeyStatus[] keyStatusArray =
    vimPort.queryCryptoKeyStatus(cryptoManager,
    [vmKeyId, diskKeyId, srcHostkey, dstHostkey], 15);
for (CryptoManagerKmpCryptoKeyStatus keyStatus : keyStatusArray[]) {
    System.out.println("keyId: " + keyStatus.getKeyId());
    System.out.println("keyAccessible: " + keyStatus.getKeyAvailable())
    System.out.println("reason: " + keyStatus.getReason())
    System.out.println("encryptedVMs: " + keyStatus.getEncryptedVMs())
    System.out.println("affectedHosts: " + keyStatus.getAffectedHosts())
    System.out.println("referencedByTags: " + keyStatus.getReferencedByTags())
}
```

The `keyAccessible` means the key is available on vCenter Server. The `reason` is either valid, or indicates why the key is not available. The `encryptedVMs` is an array of virtual machine MoRefs, and `affectedHosts` is an array of MoRefs to ESXi hosts. The `referencedByTags` field gives names of third party applications using the key.

Encrypted vSphere vMotion

When virtual machines and data move across the network, all sensitive data remain encrypted.

Enable Encrypted vMotion

The enumeration `VirtualMachineConfigSpecEncryptedVMotionModes` controls whether encrypted vMotion is disabled, opportunistic (fall back to default unencrypted vMotion if necessary), or required. If encrypted vMotion is required, or opportunistic and available, the vCenter Server can migrate either encrypted or unencrypted virtual machines within a disk cluster, or from one disk cluster to another using Storage vMotion.

Virtual Disk Manager

The following methods in managed object `VirtualDiskManager` are affected by encryption:

`copyVirtualDisk` – If the source disk is encrypted, its copied disk is encrypted with the same key, regardless of the `crypto spec`. When calling `copyVirtualDisk_Task` on vCenter Server, do not specify the `destSpec` parameter, which throws a Not Implemented fault; `destSpec` is supported only on ESXi hosts.

`createVirtualDisk` – Given a valid `crypto spec`, this task creates an encrypted virtual disk.

`moveVirtualDisk` – Encrypted virtual disk cannot be moved to an unencrypted virtual machine.

`createChildDisk` – The child disk of an encrypted disk remains encrypted with the same key.

`reparentDisks` – Encrypted child disk cannot be reparented to a new disk that is not in the same disk chain.

Best Practices for Virtual Machine Encryption

This section gives tips for optimum use of keys and virtual machine encryption.

Key Lifecycle and Removal

The `removeKey` and `removeKeys` methods delete key(s) from vCenter Server, but they do not delete keys from the KMS. Key lifecycle is managed entirely from the KMS, where stale keys persist. You can invoke the `listKeys` method to show keys in use on the vCenter Server, but there is currently no method to query whether a specific key is in use.

Be Careful with Force Remove

The `force` parameter of `removeKey` and `removeKeys` should be used judiciously. With the `force` option, the `removeKey` and `removeKeys` methods delete key(s) from both the vCenter Server and ESXi hosts, even if the key is currently in use. The result can leave virtual machines in a locked state until the key is replaced. The use case for the `force` option is for customers to prevent a key from being used anywhere, after it was compromised or expired.

Remove Keys on ESXi host

If you call `removeKey` on an ESXi host, even without the `force` option, the key gets deleted from the host's key cache, and the encrypted virtual machine becomes unusable. ESXi hosts do not track which keys are in use. Rebooting the ESXi host causes vCenter Server to push all keys to the host again, but the virtual machine may not be fully recoverable from its failed state.

Carefully Manage Differential Keys

When you encrypt both a virtual machine and its virtual disks from the vSphere Client, the same key is used for both. When you encrypt a virtual machine and its disks using the API, you can set different encryption keys for the virtual machine and each virtual disk. If one of the disk keys is missing, the power on operation may fail. If the missing key is for a non-boot virtual disk, you can remove that disk from the virtual machine and retry the power on operation. Take care when managing the lifecycle of different keys for a virtual machine and its disks.

Rename During Registration

The `registerVM_Task` method can rename a virtual machine at registration time. In vSphere 6.5 this required a two-step workaround for encrypted virtual machines, but as of vSphere 6.7, this operation is supported with encryption. Also as of vSphere 6.7, the `reloadVirtualMachineFromPath_Task` is supported for encrypted virtual machines. The reload API is equivalent to unregistering and registering a virtual machine on a different configuration path.

Encryption on vSAN Datastores

For security on vSAN datastores, you should use vSAN encryption instead of virtual machine encryption. This is because vSAN encryption is designed to be compatible with deduplication and compression, whereas virtual machine encryption causes vSAN deduplication and compression to become less effective.

Unlocking Encrypted Virtual Machines

There are many reasons why an encrypted virtual machine could be, in effect, locked. For solutions, see section “Resolve Missing Key Issues” in the *vSphere Security* manual.

In vSphere 6.7 and later you can unlock locked virtual machines with the `cryptoUnlock_Task` method. A vCenter Server alarm notifies you when an encrypted virtual machine is in a locked state. To unlock it you must have **Cryptographic operations.RegisterVM** privilege. Before unlocking, it is a good idea to troubleshoot and attempt to fix the cause of the locked virtual machine. The `cryptoUnlock_Task` method takes one parameter, the `MoRef` of a virtual machine.

Configuring Advanced Options

The advanced options settings of an ESXi host can affect virtual machine security.

Advanced options include welcome messages, `sshd` prompts, SOAP session timeout, eager zeroing of VM memory, login attempts before lockout, seconds delay after lockout, password history, password duration, password quality, DCUI timeout, shell timeout, and host client session timeout. Eager zeroing of memory can be helpful for security.

Setting advanced options on individual ESXi hosts using the UI can be impractical at scale. The vSphere API offers a programmatic interface to manipulate advanced options. The UI is built on top of the API calls.

Advanced options are controlled by the `OptionManager` managed object, a property of `HostConfigManager`. You can show advanced options with `QueryOptions`, and set them with `UpdateOptions`. For details, see the *vSphere API Reference*.

Querying Advanced Options

The full list of advanced options is available in the `OptionManager.supportedOption[]` field, and the list of non-default settings is available in the `OptionManager.setting[]` field. The contents of these arrays are fixed for a particular ESXi build and do not change at runtime.

You can use the `QueryOptions` method to get the setting for any supported option.

Setting Advanced Options

To set an advanced option, call the `UpdateOptions` method with the desired key and value. The following pseudo-code sets the advanced option for memory zeroing to "1" for true:

```
MoRef optionMgr = hostSystem.configManager.advancedOption;
opts = new OptionValue[]
opts[0].key = "Mem.MemEagerZero"
opts[0].value = "1"
optionMgr.UpdateOptions(changedValue=opts)
```

Physical Platform Resources

ESXi hosts allow direct access to physical devices in limited scenarios.

Many vSphere devices are implemented as virtual devices. Access attempts from the guest are trapped and implemented in software. For example, the keyboard is implemented by trapping I/O port access in the guest and transmitting keycode data obtained using a remote console connection; it is not possible to connect the physical keyboard to any VM.

Several categories of devices can be configured for direct access, including USB, storage, optical disc drives, virtual GPU, and PCI passthrough (DirectPath I/O) devices. Use of physical "passthrough" resources is uncommon, because it prevents vMotion. Some physical resources can be configured for access and still allow vMotion.

Physical device access can be revoked by removing an associated virtual device from its VM configuration.

USB

In its hardware devices field, data object `VirtualMachineConfigInfo` may contain the type `VirtualUSB`. Virtual devices can have a backing field, which specifies access to a particular physical device. For USB connectivity, the VM must have a compatible virtual controller configured for the VM. Programs can set the type of backing object:

- `vim.vm.device.VirtualUSB.USBBackingInfo` – USB device is attached to the ESXi host as a passthrough device, which disallows vMotion.
- `vim.vm.device.VirtualUSB.RemoteHostBackingInfo` – USB device is attached to a particular remote ESXi host, which remains attached to the device during vMotion.
- `vim.vm.device.VirtualUSB.RemoteClientBackingInfo` – USB device gets forwarded over the network along with the VM's remote console.

See data object `VirtualUSB` in the *vSphere API Reference* for details. You can use the `esxcli` command to find host-attached USB devices for connection to a VM. The following command prints available devices, including VID (Vendor ID) and PID (Product ID), which programs use to configure backing objects types above:

```
esxcli hardware usb passthrough device list
```

You can also attach USB devices to a VM using the Host Client. See procedure "Add a USB Device to a Virtual Machine in the VMware Host Client" on <https://docs.vmware.com>.

Storage

Passthrough storage devices are known as Raw Device Mappings (RDM) and represent essentially a passthrough of SCSI commands to a particular LUN. RDM can be used with vMotion if LUN IDs are kept consistent across participating ESXi hosts.

Configuring an RDM occurs through the `vim.vm.ConfigInfo.hardware.devices[]` array, with instances of type `vim.vm.device.VirtualDisk` that have backing field of type `vim.vm.device.VirtualDisk.RawDiskVer2BackingInfo`.

Physical CD-ROM or DVD

A physical disc drive on a host can be attached directly to a VM. Such drives are automatically detected by the host; no configuration is required. You can configure VMs to access a physical disc drive. See procedure "Add a CD or DVD Drive to a Virtual Machine in the vSphere Client" on <https://docs.vmware.com>.

Virtual GPU (vGPU)

When used in conjunction with an nVidia graphics card and the nVidia GRID software package, a vGPU can be passed through to one or more VMs for GPU or general purpose GPU (GPGPU) acceleration.

PCI Passthrough

ESXi allows specific PCI devices to be configured for passthrough use by a VM instead of by the host. This feature is called DirectPath I/O and is used in limited cases for improved low-latency performance. When operating in this configuration, the PCI device must be behind a VT-d bridge that implements an IOMMU. VMkernel arranges for this IOMMU to be programmed with only pages that the guest is allowed to access. This ensures that the guest can program the PCI device to access only pages that the guest should access. If the VM powers off or becomes unavailable, the PCI device receives a PCI "reset" to clear any pending DMA transactions before detaching the IOMMU domain from the device.

The host configuration that indicates which devices are eligible for assignment to VMs is through the `vim.host.PciPassthruSystem` managed object, accessible using the host's `ConfigManager.pciPassthruSystem`. The `UpdatePassthruConfig` method on this managed object can be used to enable or disable PCI passthrough for a particular PCI bus address (bus:slot:function). Making a device eligible for passthrough usually requires host reboot. Devices not configured for PCI passthrough on the host are instead claimed by ESXi and are thus unavailable to VMs.

The VM configuration uses `vim.vm.ConfigInfo.hardware.devices[]`. When an instance in that array is of type `vim.vm.device.VirtualPCIPassthrough`, the backing field has type `vim.vm.device.VirtualPCIPassthrough.DeviceBackingInfo` and this object describes the exact PCI device that is being passed through.

To enable power-on of a VM, the administrator must specify a PCI device that can be configured for assignment to the VM.

Physical Network

ESXi does not allow guests to have direct access to a physical network except when the administrator configures a virtual switch with a network interface attached to specific physical network interfaces. Any VMs expected to have external network access may be configured to have their virtual network interfaces connected to this virtual switch. See [Using a Distributed Virtual Switch](#) for information about configuring virtual switches and virtual network interface backings.

Alternatively, the administrator can configure a virtual switch to connect with a specific VLAN on the physical network and then connect VMs to that virtual switch. This provides logical isolation of the VMs from the physical network. See [Adding a Virtual Port Group](#) for information about configuration of the management network.

A virtual application consists of one or more virtual machines, which are deployed, managed, and maintained as a single unit. This chapter explains how to use the vSphere Web Services SDK for building and managing a virtual application.

This chapter includes the following topics:

- [About Virtual Applications](#)
- [Creating a VirtualApp](#)
- [Managing VirtualApp Children](#)
- [Exporting a Virtual Application](#)
- [Importing an OVF Package](#)
- [Virtual Application Life Cycle](#)

About Virtual Applications

A virtual application specifies and encapsulates the components of virtual machines and applications, and the operational policies and service levels associated with those components. A virtual application can be as simple as an individual virtual machine with a specific operating system (virtual appliance), or as complex as a complete corporate Web site. Each virtual machine in a virtual application contains a preinstalled, preconfigured operating system and might contain an application stack optimized to provide a specific set of services.

In the vSphere Web Services SDK, the `VirtualApp` managed object represents a virtual application. A `VirtualApp` object extends `ResourcePool` with the following capabilities:

- Store product information such as product name, vendor, properties, and licenses in `vAppConfigInfo`.
- Specify power-on and power-off sequence specification.
- Import and export of `VirtualApp` objects as OVF packages.
- Perform application-level customization using the OVF environment.

Management Overview

You can use the Web Services SDK to create and manage virtual applications by following these steps:

Procedure

- 1 Call the `CreateVApp` method to create a virtual application without children. See [Creating a VirtualApp](#).
- 2 Add child objects. See [Managing VirtualApp Children](#).
- 3 Export the `VirtualApp` to OVF (`ExportVApp` method) See [Exporting a Virtual Application](#).

You can then import the OVF to create and customize the virtual application.

Direct and Linked Children

A virtual application consists of one or more child virtual machines or virtual applications.

`VirtualApp` children have the following characteristics:

- Each child has exactly one parent `VirtualApp`.
- Each child can participate in power-on and power-off sequences.
- The lifetime of each child is determined by the parent `VirtualApp` object.

`VirtualApp` children are either direct or linked, based on where a child derives its resources.

- **Direct Children.** A direct child of a virtual application is a virtual machine or virtual application object that you add explicitly. See [Managing VirtualApp Children](#) for a list of methods. Direct children share resources with the parent `VirtualApp` object. Both virtual machines and virtual application can be direct children.
- **Linked Children.** A linked child of a virtual application is a virtual machine or virtual application that you add by calling the `UpdateLinkedChildren` method. Linked children increase the flexibility of the `VirtualApp` by allowing child entities to use different resources from the parent `VirtualApp` object. Linked children can be part of a different clusters, but a virtual application and its children must be in the same `Datacenter`. Both virtual machines and virtual applications can be linked children.

Linked children gives better flexibility. In particular, you can create virtual applications that span clusters. The vSphere Client does not support adding or removing links, though it does show links.

When you add a linked child to a virtual application, the following rules apply:

- An `InvalidArgument` fault is thrown if the `UpdateLinkedChildren` method is called on a link target that is a direct child of another virtual application.
- When you add a virtual machine or virtual application that is already a linked child of another virtual application, the existing link is removed and replaced with the new link.

- The life-time of a linked child is determined by the `destroyWithParent` property on the `VAppEntityConfigInfo` data object. If set to `true`, the child is destroyed when the parent `VirtualApp` is destroyed. Otherwise, the link is removed when the `VirtualApp` is destroyed.

If you add a virtual application that consists of multiple entities, for example multiple virtual machines, the entities are moved sequentially and committed one at a time, as specified in the list. If a failure is detected, the method terminates with an exception.

OVF Packages

Open Virtualization Format (OVF) is a distribution format for virtual applications. vSphere uses the OVF package as a unit of distribution and storage for virtual applications. Because these entities are uploaded, downloaded, and stored in OVF package format, vSphere supports access to and deployment of a wide variety of virtual applications.

A virtual application typically consists of one or more virtual disk files and a configuration file.

- The virtual disk files contain the operating systems and applications that run on the virtual machines in the virtual application.
- The configuration file contains metadata that describes how the virtual application is configured and deployed.

An OVF package might also include certificate and manifest files.

The OVF package contains metadata that describes the capabilities and infrastructure requirements of the virtual application, and contains references to the virtual disks and other files that store the virtual machine state. Most of this information is stored in an XML document called the OVF envelope. When an OVF package is instantiated into either a `VirtualApp` or a `VirtualMachine` object (which depends on metadata in the envelope), then the configuration stored in the OVF envelope is applied to the `VirtualVApp` and the `VirtualMachine` objects.

Some of the information in the OVF file is used unaltered, with entire `ovf:Section_Type` elements included in the `VirtualApp` object body. Other sections are transformed or extended by instantiation. You do not need detailed knowledge of all OVF package elements, but a basic understanding of key parts of the package and how they relate to virtual applications is useful.

See the OVF specification at the DMTF Web site for additional information.

Creating a VirtualApp

You always create a `VirtualApp` without children. The `CreateVApp` method includes the following parameters:

- `resSpec` – Properties you would specify for a `ResourcePool`.
- `configSpec` – `VAppConfigSpec` data object for specifying virtual-application specific information.

- `vmFolder` – Depends on the `VirtualApp` structure:
 - When creating top-level virtual applications, that is, virtual applications with no ancestor virtual applications, you must specify a folder.
 - If the `VirtualApp` has another virtual application in the ancestry chain, the `folder` parameter must be `NULL` when you create the `VirtualApp`.

Managing VirtualApp Children

You can add virtual machines and virtual applications to your virtual application as direct or linked children.

You use different methods for adding or removing direct or linked children, as follows:

- Direct children. Use one of the following methods:
 - `CreateChildVMTask` adds a new virtual machine.
 - `CreateVApp` adds a new virtual application.
 - `MoveIntoResourcePool` adds or removes an existing virtual machine or virtual application
- Linked children. Use `UpdateLinkedChildren` to add or remove virtual machines or virtual applications.

You can call the `UpdateVappConfig` method to specify how each virtual machine fits into the virtual application.

Property	Enumeration
<code>destroyWithParent</code>	True if the entity should be removed when the <code>VirtualApp</code> is removed.
<code>key</code>	Key for the virtual machine or virtual application, a managed object reference to the child.
<code>startAction</code>	One of the strings in the <code>VAppAutoStartAction</code> enumeration.
<code>startDelay</code>	Delay, in seconds, before continuing with the next entity.
<code>startOrder</code>	Specifies the start order for this entity. Entities are started from lower numbers to higher-numbers and reverse on shutdown. Multiple entities with the same start order are started in parallel and the order is unspecified. This value must be 0 or higher.
<code>stopAction</code>	Defines the stop action for the entity. Can be set to <code>none</code> , <code>powerOff</code> , <code>guestShutdown</code> , or <code>suspend</code> . If set to <code>none</code> , then the entity does not participate in auto-stop.
<code>stopDelay</code>	Delay, in seconds, before continuing with the next entity.
<code>tag</code>	Tag for the entity.
<code>waitingForGuest</code>	Determines if the virtual machine should start after receiving a heartbeat, from the guest.

For more information about direct children and linked children, see [Direct and Linked Children](#).

Exporting a Virtual Application

To export a virtual application, you must generate an OVF package. The Web Services API supports the generation of OVF packages. It does not support the generation of OVA files. An OVA file is a tar file that contains an OVF package. The OVF package consists of one or more images and an OVF file descriptor. You can create an OVA file by creating a tar file out of the OVF package for your exported virtual application.

The following steps describe how to use the vSphere `VirtualApp` and `OvfManager` API to generate an OVF package for a virtual application. The steps assume the simplest scenario: downloading one image from one device URL. You use the same steps to download many images from many device URLs. You can also export a `VirtualMachine` with the same steps, but use `VirtualMachine.ExportVm` rather than `VirtualApp.ExportVApp`.

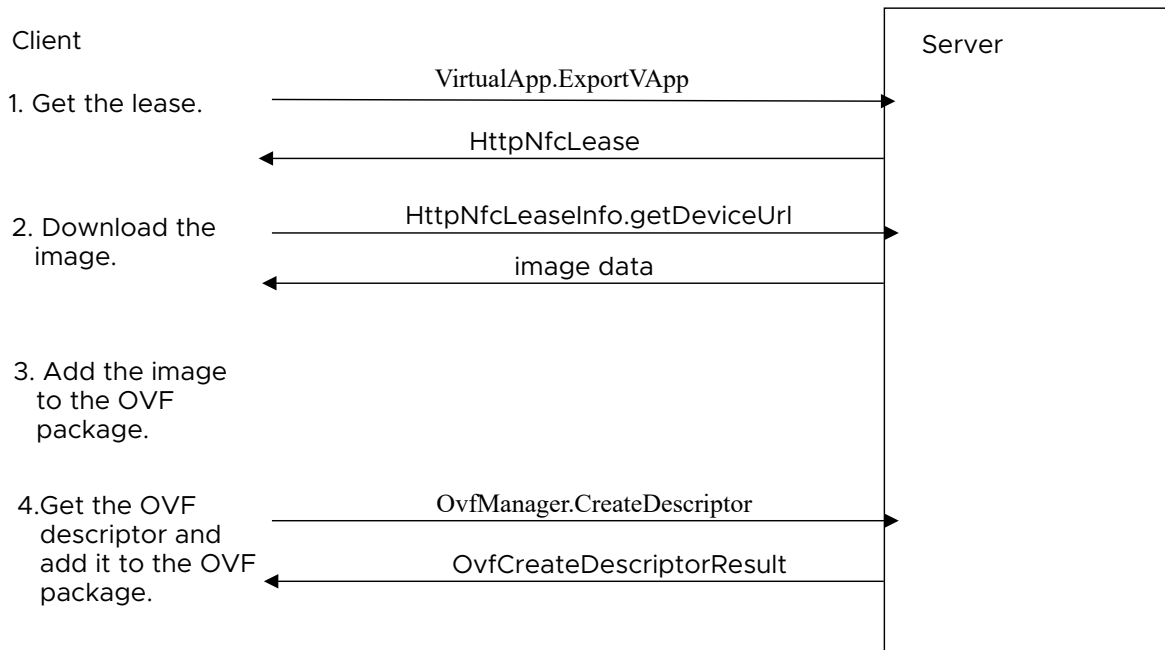
Procedure

- 1 Call the `VirtualApp.ExportVApp` method, which returns `HttpNfcLease`. The `deviceURL` is stored in the `info` property of `HttpNfcLease`.
- 2 Call the `HttpNfcLeaseInfo.getDeviceUrl` method to access the device URL and download the image data from the device URL.
- 3 Add the image to the OVF package.
- 4 Call the `OvfManager.CreateDescriptor` method, which returns `OvfCreateDescriptorResult`. Write the file descriptor to a file with the file extension `.ovf`. Add the `.ovf` file to the OVF package.

Example

[Figure 15-1. Generating an OVF Package](#) shows the major steps.

Figure 15-1. Generating an OVF Package



VirtualApp and OvfManager Methods

The following table describes the methods used by the `VirtualApp` and the `OvfManager` API:

Method	Description
<code>CreateDescriptor</code>	Creates an OVF descriptor for the specified <code>ManagedEntity</code> , which may be a <code>VirtualMachine</code> or a <code>VirtualApp</code> . <code>CreateDescriptor</code> is a method in the <code>OvfManager</code> managed object.
<code>ExportVApp</code>	Obtains an export lease on the virtual application. The export lease contains a list of URLs for the disks of the virtual machines in this virtual application. <code>ExportVApp</code> is a method in the <code>VirtualApp</code> managed object.
<code>getDeviceUrl</code>	Retrieves the device IDs and URLs from the server. <code>getDeviceUrl</code> is an accessor method provided in the generated JAX-WS bindings. It does not appear in the class diagram in VirtualApp Data Structures .

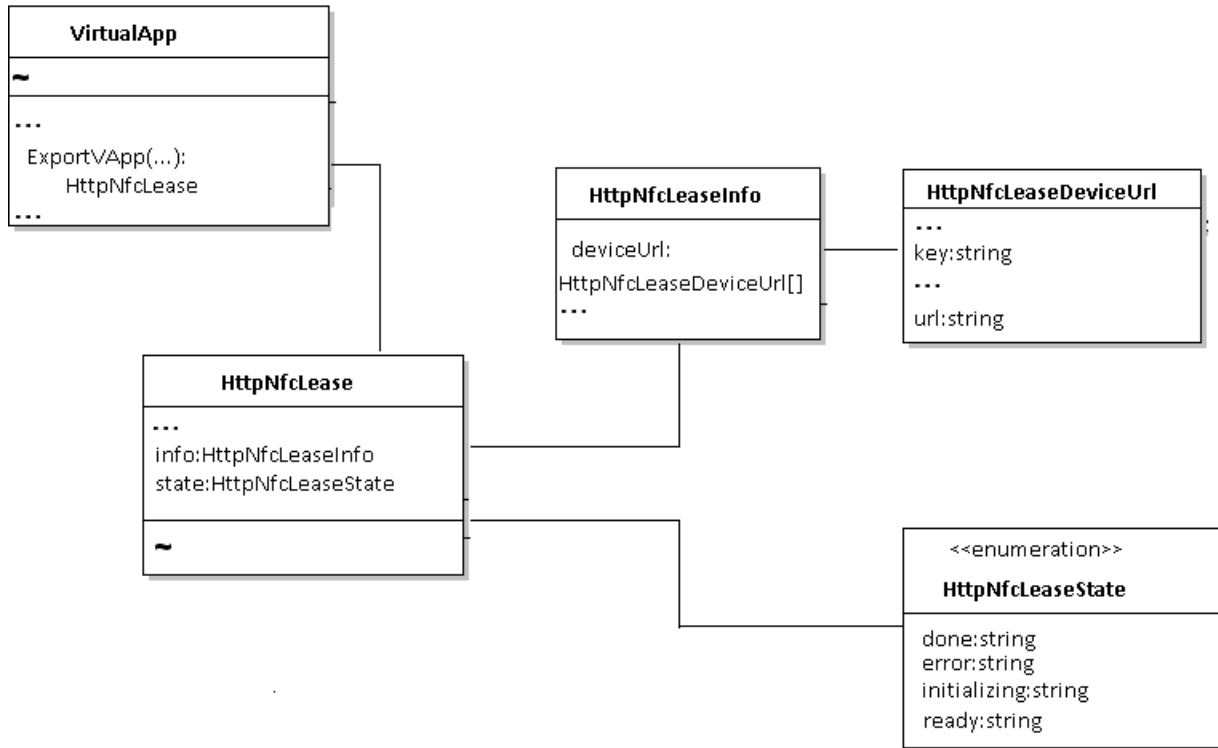
The next two sections deal with the `VirtualApp` and `OvfManager` data structures.

VirtualApp Data Structures

The `VirtualApp` managed object contains the `ExportVApp` method, which returns an `HttpNfcLease`. The `HttpNfcLease` contains the info and state properties, where info is of type `HttpNfcLeaseInfo` and state is of type `HttpNfcLeaseState`. The `HttpNfcLeaseInfo` data object has several properties, one of which is the `deviceUrl` of type `HttpNfcLeaseDeviceUrl[]`. The `HttpNfcLeaseState` has four different states—done, error, initializing and ready.

The following class diagram shows the UML representation of the data structures used in the `VirtualApp` API.

Figure 15-2. VirtualApp Class Diagram



The VirtualApp API data structures are the following:

- **VirtualApp**—A managed object that is a collection of virtual machines (and potentially other VirtualApp containers) that are operated and monitored as a unit.
- **HttpNfcLease**—A managed object returned when you call `VirtualApp.ExportVApp`. It represents a lease on the virtual application. While you hold the lease, you block the operations that alter the state of the virtual application.
- **HttpNfcLeaseInfo**—A data object that holds information about the lease, such as the virtual application covered by the lease, and the device URLs for up/downloading images.
- **HttpNfcLeaseState**—An enumeration that is a list of possible states of a lease.

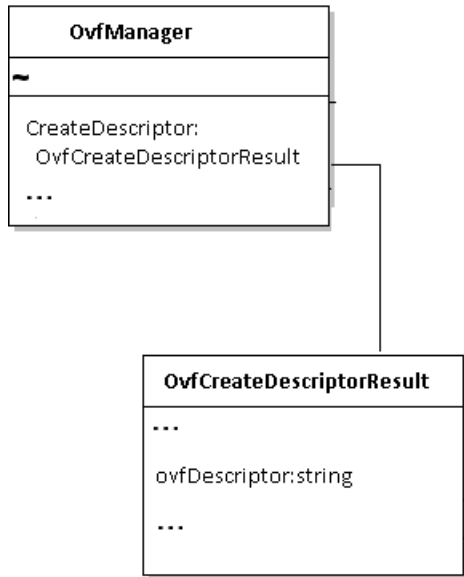
HttpNfcLeaseDeviceUrl—A data object that provides a mapping from logical device IDs to upload/download URLs.

OvfManager Data Structures

The OvfManager managed object has a `CreateDescriptor` method that returns an `OvfCreateDescriptorResult`. The `OvfCreateDescriptorResult` has the `ovfDescriptor` string.

Figure 15-3. [OvfManager Class Diagram](#) shows the UML representation of the data structures used in the OvfManager API.

Figure 15-3. OvfManager Class Diagram



The `OvfManager` data structures are the following:

- **OvfManager**—A managed object that provides a service interface to parse and generate OVF descriptors.
- **OvfCreateDescriptorResult**—A data object that contains the result of creating the OVF descriptor for the virtual application.

Example of Generating an OVF Package

In summary, the steps in generating an OVF package are the following:

Procedure

- 1 Get the managed object reference to the `VirtualApp` object. Call the `ExportVApp` method, which returns an `HttpNfcLease` data object. Wait for the state of the lease to turn to `READY`. Get the list of device URLs from the lease and store them in an array.
- 2 For each of the URLs in the list of device URLs, download the images from that URL to the client.
- 3 Save the image to the OVF package (directory/folder). Create an `OvfFile` object using the `deviceId`, absolute path of the downloaded image, and the size of the image on the local disk.

- 4 Call the `OvfManager.CreateDescriptor` method by passing the managed object reference to the `VirtualApp` and the `OvfFile` object wrapped in an `OvfCreateDescriptorParams` object. This method returns `OvfCreateDescriptorResult`, which contains the file descriptor. Write the file descriptor to a file with the file extension `.ovf`. Add the `.ovf` file to the OVF package.

The following is an example of how to generate an OVF package. The example assumes a more complex scenario: downloading more than one image from more than one device URL. The example is based on the `OVFManagerExportVAAP.java` sample, which is located in the `SDK/vsphere-ws/java/JAXWS/samples/com/vmware/vapp/` directory.

Example

You can use the `ExportVM` method instead of the `ExportVApp` method when exporting a `VirtualMachine`.

```
package com.vmware.vapp;

import java.io.*;
import java.net.URL;
import java.util.*;

...
/** 1. Get the MOR of the VirtualApp.
    ManagedObjectReference vAppMoRef = getVAPPByName(vApp);
...
/**      Call the ExportVApp method, which returns an HttpNfcLease data object. */
ManagedObjectReference httpNfcLease = vimPort.exportVApp(vAppMoRef);

...
/**      Wait for the state of the lease to turn to READY. */
Object[] result = waitForValues.wait(httpNfcLease,
                                     new String[]{"state"},
                                     new String[]{"state"},
                                     new Object[][]{new Object[]{
                                         HttpNfcLeaseState.READY,
                                         HttpNfcLeaseState.ERROR}});
if (result[0].equals(HttpNfcLeaseState.READY)) {

...
/**      Get the list of device URLs from the lease. */
List<HttpNfcLeaseDeviceUrl> deviceUrlArr = httpNfcLeaseInfo.getDeviceUrl();
if (deviceUrlArr != null) {
...
/** 2. For each of the URLs in the list of device URLs,
    *      download the images from that URL to the client. */
for (int i = 0; i < deviceUrlArr.size(); i++) {
    String deviceId = deviceUrlArr.get(i).getKey();
    String deviceUrlStr = deviceUrlArr.get(i).getUrl();
    String absoluteFile = deviceUrlStr.substring(deviceUrlStr.lastIndexOf("/") + 1);

...
/** 3. Save the image to the OVF package (directory/folder). Create an OvfFile object using
    * the deviceId, absolute path of the downloaded image, and the size of the image on the
    * local disk.
```

```

*/
    long writtenSize = writeVMDKFile(absoluteFile, deviceUrlStr.replace("*/", host));
    OvfFile ovfFile = new OvfFile();
    ovfFile.setPath(absoluteFile);
    ovfFile.setDeviceId(deviceId);
    ovfFile.setSize(writtenSize);
    ovfFiles.add(ovfFile);
}

/** 4. Call the OvfManager.CreateDescriptor method by passing the managed object reference
 * to the VirtualApp and the OvfFile object wrapped in an OvfCreateDescriptorParams object.
 * This method returns OvfCreateDescriptorResult, which contains the file descriptor.
 * Write the file descriptor to a file with the file extension .ovf. Add the .ovf file to
 * the OVF package.
 */
    ovfCreateDescriptorParams.getOvfFiles().addAll(ovfFiles);
    OvfCreateDescriptorResult ovfCreateDescriptorResult =
        vimPort.createDescriptor( serviceContent.getOvfManager(),
                                vAppMoRef,
                                ovfCreateDescriptorParams);

    String outOVF = localpath + "/" + vApp + ".ovf";
    File outFile = new File(outOVF);
    FileWriter out = new FileWriter(outFile);

    out.write(ovfCreateDescriptorResult.getOvfDescriptor());
    out.close();

```

Importing an OVF Package

To import the virtual application OVF template, you follow a few basic steps. The steps are the same for an OVF package that contains a single virtual machines or an OVF package that contains a more complex virtual application.

Procedure

- 1 Parse the OVF descriptor by calling `OvfManager.parseDescriptor`.
- 2 Validate the target ESX/ESXi host by calling `OvfManager.validateHost`.
- 3 Create the `VirtualAppImportSpec` by calling `OvfManager.createImportSpec`.

This structure contains all the information needed to create the entities on the vCenter Server, including children. Clients do not have to read or modify `VirtualAppImportSpec` to perform basic OVF operations.

- 4 Create the vCenter Server entities by calling `ResourcePool.importVApp`.

The method uses a parsed OVF descriptor to create `VirtualApp` and `VirtualMachine` objects in the vSphere environment.

Results

The import process itself consists of two steps:

- The server creates the virtual machines and virtual applications.

You must wait for the server to create all inventory objects. During object creation, the server monitors the `state` property on the `HttpNfcLease` object returned from the `ImportVApp` call. When the server completes object creation, the server changes the lease to `ready` state and you can begin uploading virtual disk contents. If an error occurs while the server is creating inventory objects, the lease changes to the error state, and the import process is terminated.

- The client application uploads virtual disk contents do an HTTP POST request with the content of the disk to the provided URLs. The disk is in the stream-optimized VMDK format (<http://www.vmware.com/technical-resources/interfaces/vmdk.html>). As an alternative, you can use the OVF tool, available at <http://communities.vmware.com/community/developer/forums/ovf> at VMware Communities.

When all inventory objects have been created and the `HttpNfcLease` has changed to `ready` state, you can upload disk contents by using the URLs provided in the `info` property of the `HttpNfcLease` object. You must call the `HttpNfcLeaseProgress` method on the lease periodically to keep the lease alive and report progress to the server. Failure to do so causes the lease to time out, terminating the import process.

When you are done uploading disks, complete the lease by calling the `HttpNfcLeaseComplete` method. You can terminate the import process by calling the `HttpNfcLeaseAbort` method.

If the import process fails, is terminated, or times out, all created inventory objects are removed, including all virtual disks.

Virtual Application Life Cycle

You can power a virtual application on or off and perform other lifecycle operations.

Powering a Virtual Application On or Off

You can use the `PowerOnVApp_Task` method to power on a `VirtualApp` object. This method starts the virtual machines or child virtual applications in the order specified in the virtual application configuration.

While a virtual application is starting, all power operations performed on subentities are disabled.

If a virtual machine in a virtual application fails to start, an exception is returned and the power-on sequence terminates. In case of a failure, virtual machines that are already started remain powered on.

You can use the `PowerOffVApp_Task` method to power off a virtual application. This method stops the virtual machines or child virtual applications in the order specified in the `VirtualApp` object configuration if `force` is `false`. If `force` is set to `true`, this method stops all virtual machines (in no specific order and possibly in parallel) regardless of the `VirtualApp` object auto-start configuration.

While a virtual application is stopping, all power operations performed on subentities are disabled.

Unregistering a Virtual Application

You can call the `UnregisterVApp_Task` method to remove a `VirtualApp` object from the inventory without removing any of the component virtual machine files on disk. All high-level information stored with the management server (ESXi or vCenter Server system) is removed, including information about `VirtualApp` object configuration, statistics, permissions, and alarms.

Suspending a Virtual Application

You can call the `SuspendVApp_Task` method to suspend all running virtual machines in a virtual application, including virtual machines running in child virtual application. The virtual machines are suspended in the order that is used for a power off operation, which is the reverse of a power on sequence.

While a virtual application is being suspended, all power operations performed on subentities are disabled. If you attempt to perform a power operation, a `TaskInProgress` error results.

Destroying a Virtual Application

When a `VirtualApp` object is destroyed, all of its virtual machines and any child virtual applications are destroyed.

The `VirtualAppVAppState` type defines the set of states a `VirtualApp` object can be in. The transitory state between started and stopped is modeled explicitly, since the starting or stopping of a virtual application might take minutes to complete.

The life-time of a linked child is determined by the `destroyWithParent` property on the `VAppEntityConfigInfo` data object. If set to `true`, the child is destroyed when the parent virtual application is destroyed. Otherwise, only the link is removed when the virtual application is destroyed.

Resource Management

16

Underlying all virtual components are the actual physical resources of the host system, such as CPU, RAM, storage, network infrastructure, and so on. vSphere supports sharing of resources on an individual host or across hosts using resource pools. vSphere also supports clusters for failover or load balancing.

This chapter includes the following topics:

- [Resource Management Objects](#)
- [Introduction to Resource Management](#)
- [Resource Allocation](#)
- [Creating and Configuring Resource Pools](#)
- [Introduction to vSphere Clusters](#)
- [Creating and Configuring Clusters](#)
- [Managing DRS Clusters](#)
- [Managing HA Clusters](#)

Resource Management Objects

Central to resource management for all environments is either a `ComputeResource` or a `ClusterComputeResource` managed object.

- The `ComputeResource` managed object represents the set of resources for a set of virtual machines. A `ComputeResource` is always associated with a root `ResourcePool` object, representing the resources of a single host.
- The `ClusterComputeResource` data object aggregates the compute resources of multiple associated `HostSystem` objects into a single compute resource for use by virtual machines.

A `ClusterComputeResource` is associated with a root `ResourcePool` representing the combined resources of the cluster. If you plan on using VMware cluster services such as HA (High Availability), DRS (Distributed Resource Scheduling), or on using EVC (Enhanced vMotion Compatibility), use `ClusterComputeResource`.

Important HA, DRS, and EVC may require feature licenses. If any clustering functionality does not work properly, check whether you have licenses for it.

- The `ResourcePool` managed object represents a set of physical resources of a single host, a subset of a host's resources, or resources spanning multiple hosts. Resource pools can be subdivided by creating child resource pools. Only virtual machines associated with a resource pool can be powered on.

Introduction to Resource Management

An ESXi host allocates to each virtual machine a portion of the underlying hardware resources based on several factors:

- Total available resources for the ESXi host, resource pool, or cluster to which the virtual machine belongs.
- Number of virtual machines powered on and resource usage by those virtual machines.
- Overhead required to manage the virtualization.
- Constraints defined by the user.

Resource management allows you to dynamically allocate resources to virtual machines so that you can more efficiently use available capacity. You can change resource allocation in the following ways.

- Specify resource allocation for individual virtual machines. See [Configuring Resource Allocation Constraints for Virtual Machines](#).
- Create a hierarchy of resource pools and add the virtual machine to a resource pool with characteristics appropriate for its use. See [Resource Pool Hierarchies](#).
- Add hosts and virtual machines to a cluster so you can take advantage of VMware DRS for recommendations or automatic resource redistribution. See [Creating and Configuring Clusters](#).

Resource Allocation

When you create a virtual machine, you always specify the resource pool that the virtual machine can draw resources from and optionally a host on which the virtual machine should run. You can access the resource pool as follows:

- Standalone host – When you call `Folder.AddStandaloneHost_Task`, the call returns a `Task` object that contains the `ComputeResource`. The `ComputeResource.resourcePool` property is the root resource pool associated with the compute resource (and with the host).

- **Cluster** – When you call `Folder.CreateClusterEx`, the method returns a managed object reference to a `ClusterComputeResource` instance. Because `ClusterComputeResource` inherits all properties of `ComputeResource`, you can access the root resource pool through the `ClusterComputeResource.resourcePool` property.

Resource Pool Hierarchies

Resource pool hierarchies allow detailed control over which virtual machines are allowed how many resources, by dividing resources between children at each level of the hierarchy. Child resource pools can be configured with reservations and limits, similar to virtual machines. Child resource pools can also be configured with shares settings, which take effect in resource contention situations.

For more information about how reservations and limits apply to resource pool hierarchies, see [Configuring Reservation and Limit for Resource Pools](#). For more information about how shares apply to resource pool hierarchies, see [Configuring Priority Shares for Resource Pools](#).

Cluster Overview

vSphere supports grouping ESXi hosts that are managed by the same vCenter Server system into clusters. Clusters take advantage of features such as VMware DRS and VMware HA.

- **VMware HA** (VMware High Availability) fails over virtual machines from one host in a cluster to another host, in the event of host failure.
- **VMware DRS** (VMware Distributed Resource Scheduler) provides dynamic redistribution of resources. DRS also includes support for Distributed Power Management (DPM), which makes recommendations or decisions to power off hosts and power them on again as needed, to save energy.

You can set up VMware DRS to automatically migrate virtual machines, or to display recommendations if resources are not used efficiently across the cluster.

See [Creating and Configuring Clusters](#) and [Managing DRS Clusters](#).

Creating and Configuring Resource Pools

A root resource pool is associated with each `ComputeResource` and with each `ClusterComputeResource`.

You can create a hierarchy of resource pools by calling the `ResourcePool.CreateResourcePool` method and passing in a `ResourceConfigSpec` argument. The `ResourceConfigSpec.cpuAllocation` and `ResourceConfigSpec.memoryAllocation` properties point to `ResourceAllocationInfo` objects that allow you to specify the following information.

- `reservation` – Amount of CPU or memory that is guaranteed available to virtual machines within the resource pool. Reserved resources are not wasted if they are not used. If the utilization is less than the reservation, the resources can be borrowed by virtual machines running within other resource pools.
- `expandableReservation` – In a resource pool with an expandable reservation, the reservation on a resource pool can expand beyond the specified value, if the parent resource pool has unreserved resources. A non-expandable reservation is called a fixed reservation. See [Understanding Expandable Reservations](#).
- `limit` – Upper limit for CPU or memory resources assigned to this resource pool. The resource pool does not allocate more resources to its children, even if resources are available through its parent. This property is typically used to ensure consistent performance by isolating other resource pools from the effects of the running virtual machines within this pool. Set this property to -1 to indicate no fixed upper limit on resource usage.
- `shares` – Relative metric for allocating memory or processing capacity among multiple resource pools in resource contention situations. The shares value indicates resource priority relative to the shares values of sibling resource pools or virtual machines. The `SharesInfo` data object has two properties, `level` and `shares`, that allow you to specify resource allocation.
 - `level` – Choose `high`, `low`, or `normal` to map to a predetermined set of numeric values for shares. See the *API Reference Guide* for the numbers for CPU, memory, and disk shares. Set this property to `custom` to specify an explicit number of shares instead.
 - `shares` – Allows you to specify a custom value for the number of shares you want to allocate to the resource pool. This property is ignored unless the `level` is set to `custom`.

To change the configuration, call the `ResourcePool.UpdateConfig` or `ResourcePool.UpdateChildResourceConfiguration` method and pass in a `ResourceConfigSpec` that contains values for all fields you want to update.

Configuring Reservation and Limit for Resource Pools

Resource pools are a tool to aggregate physical host resources and channel the aggregated resources to individual virtual machines. You use resource pools to group virtual machines, either to isolate portions of aggregated resources or to prioritize resource allocations between groups of virtual machines.

Resource pools, like individual virtual machines, can be configured with boundaries for resource allocation. Resource pools can also be configured with priority shares, although shares settings act somewhat differently for resource pools than for virtual machines. See [Understanding Fixed Shares](#) and [Understanding Scalable Shares](#).

You configure a `reservation` value for a resource pool to help protect its virtual machines from the negative impact of virtual machines in other pools. If you configure a fixed reservation for a resource pool, the host will always make at least that much of the resource available for the resource pool to distribute among its children. See [Understanding Fixed Reservations](#).

Note If a resource pool's virtual machines are not using the resource pool's entire reserved amount of a resource, the host may allocate the unused amount to virtual machines in other pools, on a temporary basis. The host will return the borrowed resource to the resource pool that reserved it, whenever needed to start virtual machines within that resource pool.

You configure a `limit` value for a resource pool to help prevent its virtual machines from negatively impacting performance of virtual machines in other resource pools. The host will never allocate more than that quantity of the resource to any or all of the virtual machines within the pool, whether immediate children or children of a nested pool. After all of the resource is allocated, a resource shortage might cause virtual machines within the pool to fail to start or fail to progress, but it has no effect on virtual machines outside the pool.

If you configure expandable reservations for a resource pool and its siblings, and you configure no reserved amounts for any of the siblings, then they share the parent's `reservation` amount on a 'first come first serve' basis. If any siblings are configured for fixed reservations, those siblings are guaranteed their fixed reservations but no more, while the remainder of the parent's reservation is available to share among the pools configured for expandable reservations. See [Understanding Exxpandable Reservations](#).

Understanding Fixed Reservations

A fixed reservation for a resource pool provides a way to guarantee that all virtual machines running simultaneously within the pool have access to a minimum quantity of the resource. The pool's reservation should be large enough to allocate among the virtual machines such that each one has its configured minimum required to start up.

To configure fixed reservations, set the `expandableReservation` property to `false` in `ResourceConfigSpec.ResourceAllocationInfo`. Set the pool's `reservation` amount to an aggregate value that is sufficient to support the maximum number of virtual machines that need to run concurrently within the resource pool. If the pool has nested resource pools that are also configured for fixed reservations, you should determine their `reservation` amounts in the same way, recursively.

When you configure a resource pool hierarchy for fixed reservations, consider these guidelines:

- A resource pool that contains only virtual machines should be configured with a reservation amount at least as great as the sum of the reservations of its children, or as many of them as need to run concurrently. Otherwise, one or more virtual machines will fail to get its minimum resource allocation and the host will not start it.
- A resource pool that contains only nested resource pools configured for fixed reservations must be configured with a reservation amount at least as great as the sum of the reservations of its children. This ensures that its child resource pools can fulfill their obligations to their own children, according to their configured `reservation` amounts. If you cannot configure the parent pool with an adequate amount of the resource, consider configuring the nested resource pools for expandable reservations instead.
- A resource pool that contains both virtual machines and nested resource pools is not a best practice, because it can complicate configuration management. If you configure a resource pool in this way, you should first make sure the virtual machines can receive their own reservation amount, and then configure reservations for the nested pools from the remainder of the parent's reservation after subtracting the virtual machine reservations.

Before you create new child resource pools, check available resources in the parent pool. The `ResourcePool.runtimeInfo` property is a `ResourcePoolRuntimeInfo` data object. The `ResourcePoolRuntimeInfo.cpu` and `ResourcePoolRuntimeInfo.memory` properties are `ResourcePoolResourceUsage` objects with resource usage information, including an `unreservedForPool` property. If the parent pool does not have enough available resources, reconfigure the reservation values of child pools before adding the new pool.

Understanding Expandable Reservations

Expandable reservations enable dynamic allocation of resources to meet the minimum requirements of virtual machines within a number of resource pools. When you set the `expandableReservation` property on sibling resource pools, you enable them to act as a single resource pool for the purpose of providing the minimum of resource that virtual machines need to power on. When a virtual machine starts in any one of the sibling resource pools, it draws its `reservation` amount from the parent's `reservation` amount, rather than from its own resource pool.

If the parent resource pool is also expandable, it can draw resources in turn from its parent, and the sequence can continue in this way until it ends with a parent resource pool that has a fixed reservation.

To configure expandable reservations, set the `expandableReservation` property to `true` in the `ResourceConfigSpec.cpuAllocation` or `ResourceConfigSpec.memoryAllocation` of sibling resource pools to `true`. For maximum flexibility, set the `reservation` amount of each sibling pool 0. When sibling pools do not reserve any resources, any virtual machine that starts in the sibling pools will take its reservation amount from the parent pool's reservation.

If a sibling pool contains critical virtual machines whose reservations need to be prioritized, set the pool's reservation to the sum of the reservations of the prioritized virtual machines. That reservation amount is saved for the resource pool's virtual machines, and is unavailable for siblings to use.

To illustrate how expandable reservations work, consider the following examples.

Expandable Reservation Example 1

Assume an administrator manages a parent pool P1, and defines two child resource pools, C1 and C2, for two different groups of users. C1 and C2 are self-serve resource pools, allowing users to configure their own virtual machines. The administrator does not know in advance exactly what resources the users will need, so the administrator wants the resource pool configuration to be flexible.

Without expandable reservations, the administrator needs to explicitly allocate fixed amounts of resources to C1 and C2. Such specific allocations can be inflexible, especially in deep resource pool hierarchies, and can complicate setting reservations for C1 and C2. By making the reservations for C1 and C2 expandable, the administrator allows users to more flexibly share and inherit the common reservation for pool P1.

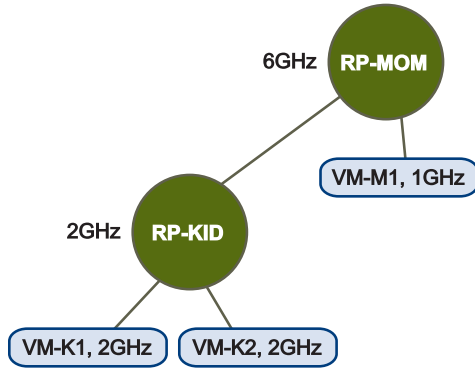
Expandable reservations cause a loss of resource pool isolation in the context of admission control. For example, if C1 and C2 have their reservation amounts set to 0, then virtual machines in C1 might use all of P1's memory reservation, so that no memory is available to start virtual machines in C2.

Expandable Reservation Example 2

Assume a parent resource pool RP-MOM has a reservation of 6GHz and one running virtual machine VM-M1 that reserves 1GHz. RP-MOM also has a child resource pool RP-KID with a reservation of 2GHz and with **Expandable Reservation** selected. RP-KID contains two virtual machines, VM-K1 and VM-K2, with reservations of 2GHz each.

When a user powers on VM-K1, it can reserve the resources it needs directly from RP-KID (which has 2GHz). When the user tries to power on VM-K2, RP-KID has already allocated its 2GHz reservation to VM-K1, but it has **Expandable Reservation** configured, so it tries to borrow resources from RP-MOM's reservation. RP-MOM has 6GHz minus 1GHz (reserved by VM-M1) minus 2GHz (reserved by RP-KID), which leaves 3GHz of RP-MOM's reservation that is not reserved by other resource pools or virtual machines in RP-MOM. With 3GHz available, VM-K2 is able to power on.

Figure 16-1. Admission Control with Expandable Resource Pools, Scenario 1

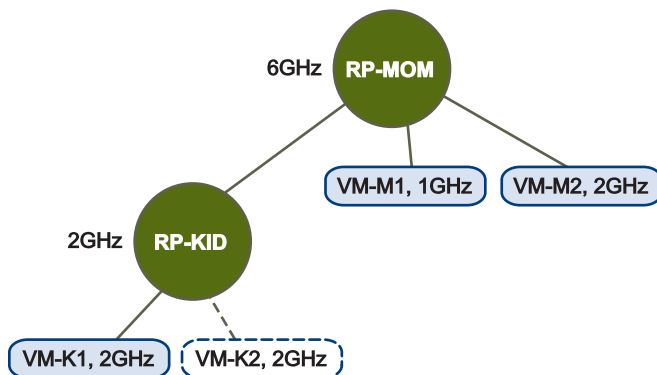


Expandable Reservation Example 3

Assume a parent resource pool RP-MOM has a reservation of 6GHz and two running virtual machines, VM-M1 that reserves 1GHz, and VM-M2 that reserves 2GHz. RP-MOM also has a child resource pool RP-KID with a reservation of 2GHz and with **Expandable Reservation** selected. RP-KID contains two virtual machines, VM-K1 and VM-K2, with reservations of 2GHz each.

When a user powers on VM-K1, it can reserve the resources it needs directly from RP-KID (which has 2GHz). When the user tries to power on VM-K2, RP-KID has already allocated its 2GHz reservation to VM-K1, but it has **Expandable Reservation** configured, so it tries to borrow resources from RP-MOM's reservation. RP-MOM has 6GHz minus 3GHz (reserved by VM-M1 and VM-M2) minus 2GHz (reserved by RP-KID), which leaves only 1GHz of RP-MOM's reservation that is not reserved by other resource pools or virtual machines in RP-MOM. Since VM-K2 requires 2GHz to pass admission control, VM-K2 is not able to power on.

Figure 16-2. Admission Control with Expandable Resource Pools, Scenario 2



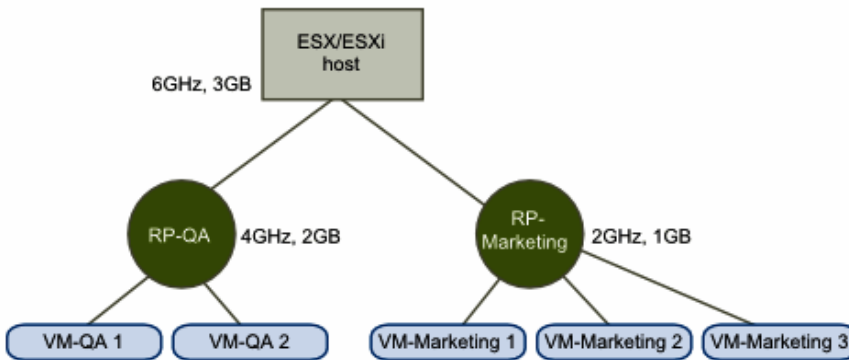
Configuring Priority Shares for Resource Pools

In cases of resource contention, data center hosts need a way to arbitrate between competing virtual machines. You can configure the shares settings of a virtual machine to set its priority relative to its siblings. You can configure the shares settings of a resource pool to affect the priorities of all virtual machines within the resource pool, as a group.

The following illustration shows a standalone host that has several virtual machines. The marketing department uses three of the virtual machines and the QA department uses two virtual machines. Because the QA department needs larger amounts of CPU and memory, the administrator creates one resource pool for each group. The administrator sets CPU Shares to High for the QA department pool and to Normal for the Marketing department pool so that the QA department users can run automated tests. The second resource pool with fewer CPU and memory resources is sufficient for the lighter load of the marketing staff.

Whenever the QA department is not fully using its allocation, the marketing department can use the available resources. When there is resource contention because running virtual machines demand more resources than are available from the host, the resource shares configuration guides arbitration between the competing virtual machines.

Figure 16-3. Allocating Resources to Resource Pools



You have two options for priority shares configuration that cause the priorities to be applied in different ways. The fixed shares option limits the adverse performance impact of resource contention to a single resource pool and all its descendants. The scalable shares option distributes the performance impact across all resource pools, in proportion to their priority levels.

The chief benefit of configuring fixed shares for a parent resource pool is the predictability of performance for virtual machines within its child resource pools. You can know in advance that its virtual machines can demand a fixed fraction of a virtual resource available from a host or cluster. The drawback of configuring fixed shares is that adding virtual machines to a resource pool impacts the performance of all virtual machines in the same pool and its descendants because the resource pool is not entitled to allocate additional resources.

The chief benefit of configuring scalable shares for a parent resource pool is that resource allocation during contention is adjusted at run time to achieve a fair allocation to virtual machines beyond the boundary of a single resource pool. In effect, the child pool's resource entitlement expands to accommodate more virtual machines as they are added to the pool. The drawback of configuring scalable shares is that a child resource pool cannot isolate its virtual machines from the demands of virtual machines in other pools that draw from the same scalable parent resource pool.

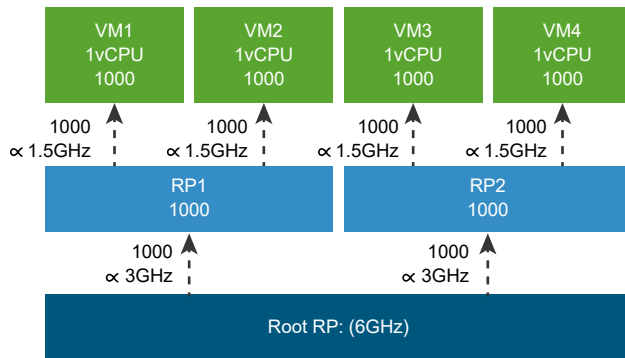
Understanding Fixed Shares

When a parent resource pool is not configured to be scalable, the shares configured for its child resource pools provide a way to prioritize allocation of fixed fractions of the parent resource pool to its children. This serves to isolate users of sibling resource pools so that virtual machines in one child resource pool cannot impact the performance of virtual machines in other child resource pools.

Suppose for example a cluster has 3 physical cores running at 2GHz, for a total of 6GHz. The root resource pool has a total of 6GHz of virtual CPU capacity to divide between its children. If there is no resource contention, all running virtual machines can be allocated their configured amounts of CPU. If there is contention for CPU resources, the 6GHz is divided between child resource pools according to the CPU shares configured for them.

This fictional data center has two child resource pools, each supporting the users in a different division of a business. Division 1 and Division 2 start out about the same size, so the IT department configures RP1 and RP2 with equal values for custom shares, and an equal number of virtual machines.

Figure 16-4. Example Resource Pool Configuration with Custom Shares



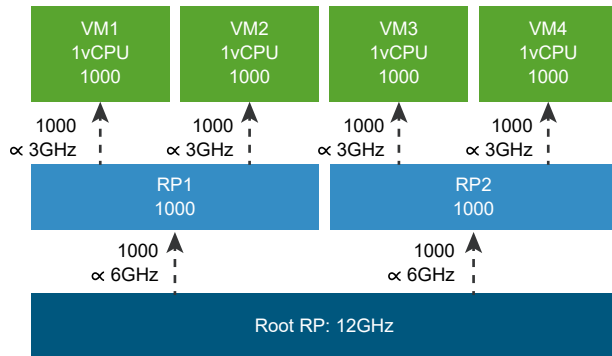
All child resource pools and virtual machines are configured with custom settings of 1000 shares each. Because the two resource pools are configured with the same number of shares, they can allocate the same amount of compute resource to their virtual machines in cases of resource contention. Likewise, all virtual machines are entitled to draw the same fraction of compute resource as their siblings.

Note For simplicity of illustration, these examples deal only with CPU shares. A real configuration would include both CPU and memory resources, which act independently in regard to calculating the absolute resources available to a running virtual machine.

In effect, a non-scalable resource pool guarantees that each of its children gets a fixed fraction of its resources. However, the amount allocated to each child is fixed only in relation to the quantity of resources controlled by the resource pool itself. If a root resource pool gains additional resources due to a hardware upgrade, all its children gain resource entitlements in proportion to their configured shares.

For example, suppose the cluster has been upgraded from 3 to 6 physical cores running at 2GHz, for a total of 12GHz of pooled physical capacity. The root resource pool now has a total of 12GHz of virtual CPU capacity to divide between its children. If there is contention for CPU resources, the 12GHz is divided between child resource pools according to the CPU shares configured for them.

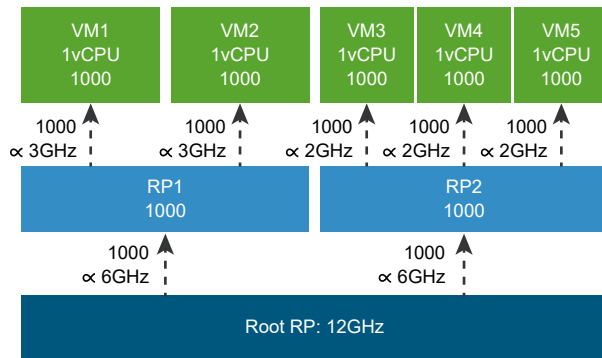
Figure 16-5. Increased Physical Capacity Provides Increased Virtual Capacity to All Virtual Machines



After the upgrade, each virtual machine is entitled to twice its previous resource allocation, in absolute terms. However, if another child, whether resource pool or virtual machine, is added to a parent pool, previously existing children find their resource allocation reduced, as the pool's resources are shared between more children.

Suppose Division 2 hires some new users, and IT adds an extra virtual machine to RP2 for their use.

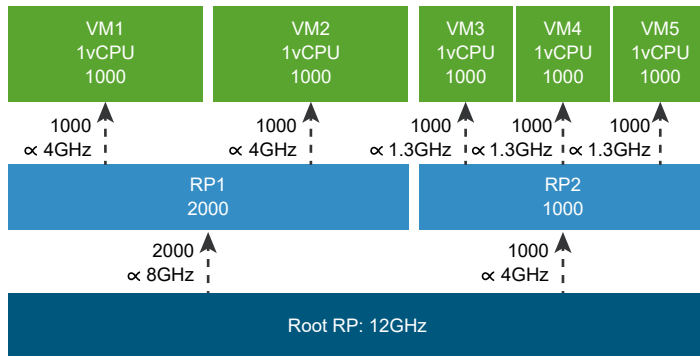
Figure 16-6. Dilution of Fixed Shares From Adding a Virtual Machine to a Resource Pool



Where previously all virtual machines in the business had the same priority for allocation of scarce compute resource, increasing the virtual machine load in RP2 means that more virtual machines contend for the same quantity of its resource. Consequently, each virtual machine in RP2 might see its performance reduced when all virtual machines are running at full capacity.

To continue from the previous example, some users in Division 1 might be concerned about performance because they plan to hire more employees in the future. Suppose they persuade the IT department that their virtual machines should be in a higher priority resource pool than others, so they will have access to more compute resource if needed. The IT department configures different custom shares values for RP1 and RP2, so that RP1 has twice the priority of RP2.

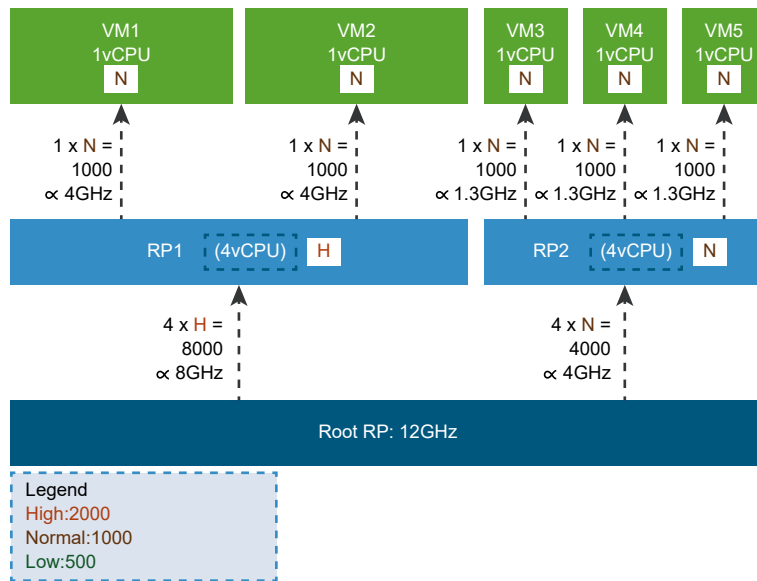
Figure 16-7. Effects of Reprioritizing a Resource Pool in a Fixed Shares Configuration



The child resource pool RP1 has 2000 shares configured, and RP2 has 1000 shares configured, so RP1 gets 2/3 of the physical CPU resources to share among its children, and RP2 gets 1/3 of the physical CPU resources to share among its children. You can calculate the amount of compute resource available to RP1 for its children as $12 \times 2000 / (2000+1000) = 8\text{GHz}$, and the amount of compute resource available to RP2 for its children as $12 \times 1000 / (2000+1000) = 4$ virtual GHz. The children within each resource pool have identical shares settings, so they share their own pool's resources equally.

Now suppose a new CIO joins the business and decides that from now on priority shares should be specified as levels rather than custom values. The IT department reconfigures all the shares settings to use the enums for shares levels.

Figure 16-8. Priority Shares Expressed as Levels Instead of Custom Values



When you specify shares levels, you have to make two calculations. First you convert the specified level to a numeric value that corresponds to custom shares settings. This conversion takes into account the number of virtual cores.

Note For memory resources, the conversion takes into account the configured memory size of the virtual machine.

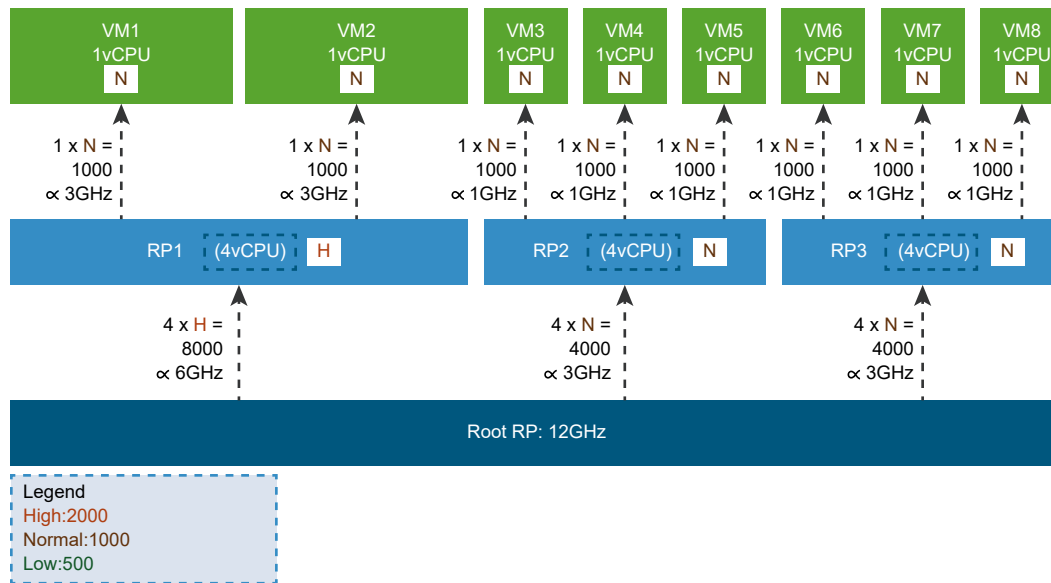
Then you use that numeric value to calculate the proportion of resources to allocate to the child pool, in the same way as with the custom shares settings.

To convert a shares level to a numeric value, multiply the number of vCPUs by a constant that corresponds to the level. For the purpose of this calculation, all resource pools have an implicit size of 4 vCPUs. Use the `VirtualMachine.config.hardware.numCPU` value as the number of vCPUs for virtual machines. The constant for the `high` priority level is 2000, and the constant for the `normal` level is 1000. You can calculate that the numeric shares belonging to RP1 are $4 \times 2000 = 8000$. RP2 has a priority level of `normal`, so the numeric shares belonging to RP2 are $4 \times 1000 = 4000$. The compute resource that RP1 can allocate is $12 \times 8000 / (8000 + 4000) = 8\text{ GHz}$, and the compute resource that RP2 can allocate to its children is $12 \times 4000 / (8000 + 4000) = 4\text{ GHz}$. The new CIO is satisfied that RP1 and RP2 achieve the same result with priority levels as with the previous custom shares values.

Note For memory shares, the constants are different. See the vSphere Web Services API Reference.

But suppose the IT department is asked to support a 3rd division, newly acquired by the business, which is about the same size as the division assigned to RP2. The IT department creates a new child resource pool, RP3, and configures the same number of virtual machines and the same shares levels as RP2. Now the division of resources looks like this:

Figure 16-9. Resource Dilution from Adding a Resource Pool



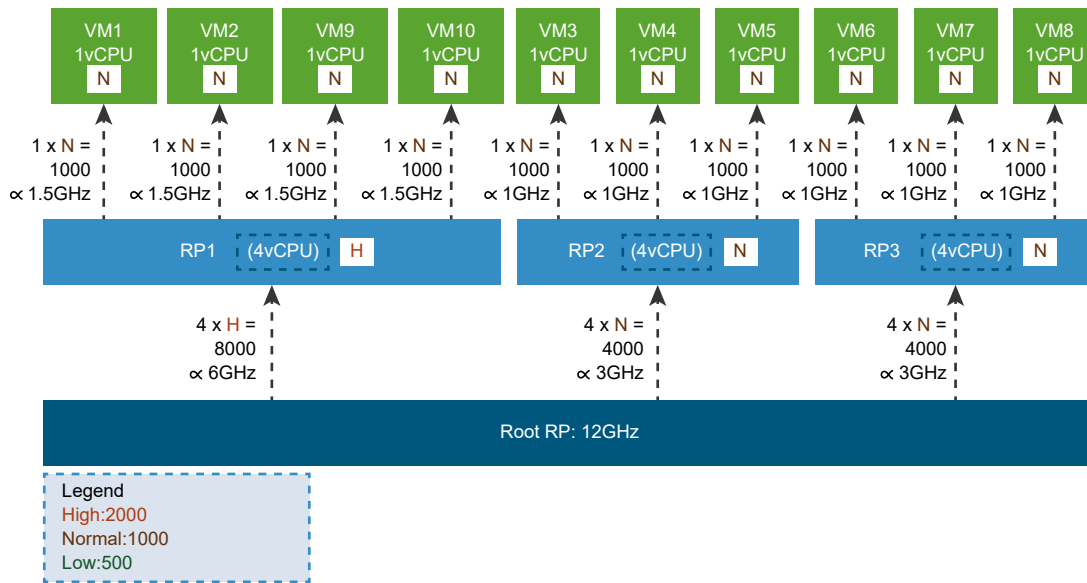
Assume the hardware resources available to the root resource pool have not changed at the time of the acquisition, so the 12GHz of virtual compute resource must be divided into fractions according to the configured shares of the child resource pools. After adding RP3, both RP1 and RP2 get less of the virtual compute resource to divide among their own children.

Virtual machines available to Division 1 now compete for $12 \times 8000 / (8000 + 4000 + 4000) = 6\text{GHz}$ in RP1. Virtual machines available to Division 2 compete for $12 \times 4000 / (8000 + 4000 + 4000) = 3\text{GHz}$ in RP2. Virtual machines available to Division 3 similarly compete for $12 \times 4000 / (8000 + 4000 + 4000) = 3\text{GHz}$ in RP3.

In this example, Division 1 and Division 2 employees might both perceive reduced performance until the IT department's capital budget increases to support the new configuration.

When Division 1 hires a number of new employees, IT adds two more virtual machines to RP1 to accommodate the increased size of Division 1. All four virtual machines in RP1 still benefit from RP1's high priority level, but now that RP1's compute resources are spread among a larger number of virtual machines, each of its virtual machines gets a smaller allocation when they are competing for CPU cycles.

Figure 16-10. Resource Dilution from Adding Virtual Machines to a High Priority Resource Pool

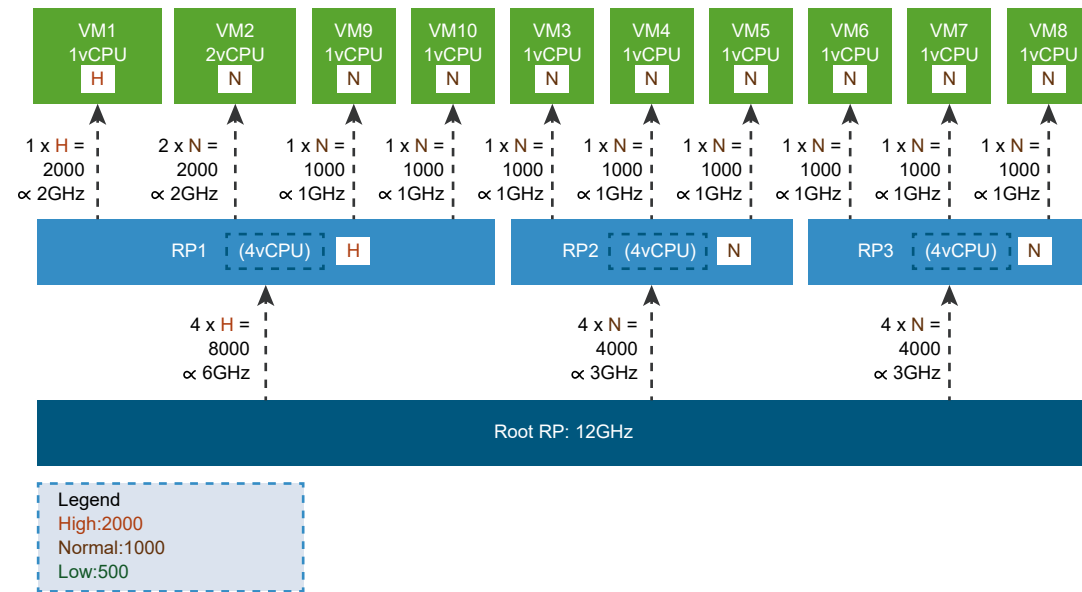


Some virtual machine users who were in Division 1 before the new employees joined are noticing greatly reduced performance. They are only getting 1.5GHz where they used to get 3GHz. When they learn that Division 2 employees are getting 1GHz from their virtual machines in RP2, which has `normal` priority, they might not be happy that they are getting only 1.5GHz from virtual machines in RP1, which has `high` priority.

The disappointed users speak to the IT department, where they learn that this is the way fixed shares for resource pools are intended to work. They isolate Division 1 virtual machine performance from the other divisions, but not from other users of the same resource pool. These Division 1 users learn that there are other ways to protect the performance of high priority virtual machines.

Some Division 1 users try other ways to get more compute resource from RP1's fixed allocation. They request changes to virtual machine configurations, which result in increased priority for some of their virtual machines at the expense of others.

Figure 16-11. Prioritizing Individual Virtual Machines Within a Resource Pool



After VM1 is reconfigured, you calculate its numeric shares by multiplying the configured `numCPU` (1) by its shares level (`high`), getting $1 \times 2000 = 2000$. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: $6\text{GHz} \times 2000 / (2 \times 2000 + 2 \times 1000) = 2\text{GHz}$.

You calculate VM2's numeric shares by multiplying the configured `numCPU` (2) by its shares level (`normal`), getting $2 \times 1000 = 2000$. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: $6\text{GHz} \times 2000 / (2 \times 2000 + 2 \times 1000) = 2\text{GHz}$.

In a similar way, calculate VM9's numeric shares by multiplying its configured `numCPU` (1) by its shares level (`normal`), getting $1 \times 1000 = 1000$. You calculate its resource allocation by using its numeric shares to prorate the allocation available to its parent resource pool: $6\text{GHz} \times 1000 / (2 \times 2000 + 2 \times 1000) = 1\text{GHz}$. VM10's resource configuration and associated calculations are identical to VM9.

When other Division 1 users realize that VM9 and VM10 only get 1GHz of virtual compute resource, they might not be satisfied with this result. These virtual machines are in a `high` priority resource pool, but they get the same resource allocation as Division 2's virtual machines, which are in a `normal` priority resource pool. That doesn't seem fair.

Division 1 users request a meeting with the CIO, who explains that the scalable shares feature is the best solution to their problem.

Understanding Scalable Shares

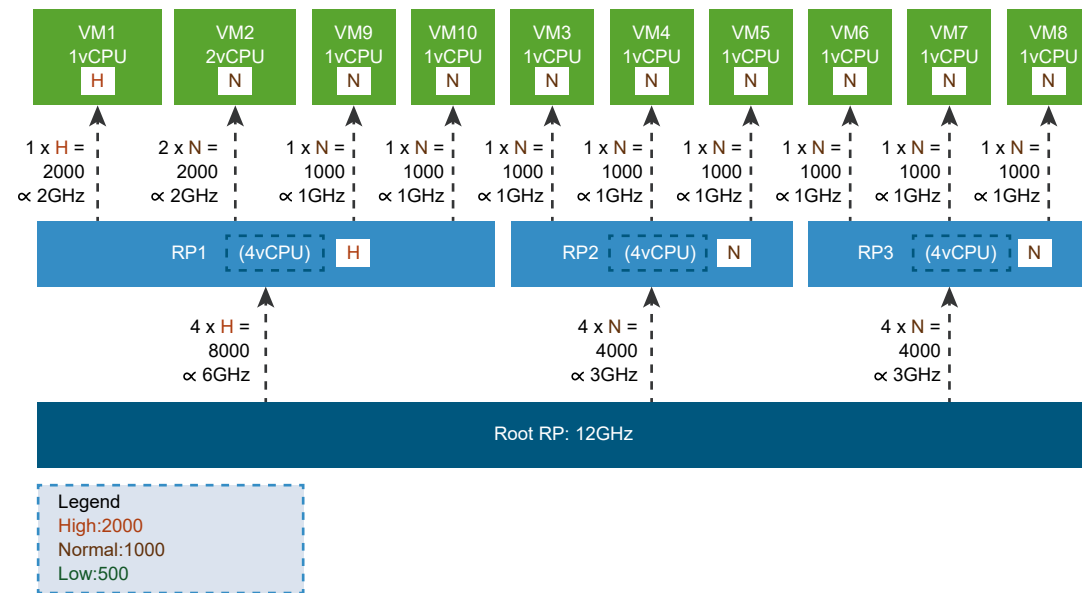
Configuring scalable shares for a parent resource pool means that all its child resource pools are not isolated from each other. This avoids the resource dilution problem within child resource pools, by expanding their resource entitlements when they have more virtual machines

running. The effect is to negate resource pool isolation while retaining the proportions of resource pool priority levels. A `high` priority resource pool is entitled to give twice as much of the available resource to its virtual machines, in comparison with a `normal` priority sibling resource pool, but all virtual machines suffer proportionately when many virtual machines run in the normal priority pool.

To illustrate the use of the scalable shares option, consider how it might apply to a fictional situation in which a data center is configured, with non-scalable shares, to support three divisions of a business. Division 1 is supported by resource pool RP1, which is configured as `high` priority, in an attempt to compensate for the dilution of RP1's resources that resulted from adding new virtual machines to RP1. Some virtual machines in RP1 are individually configured to get higher priority than their siblings.

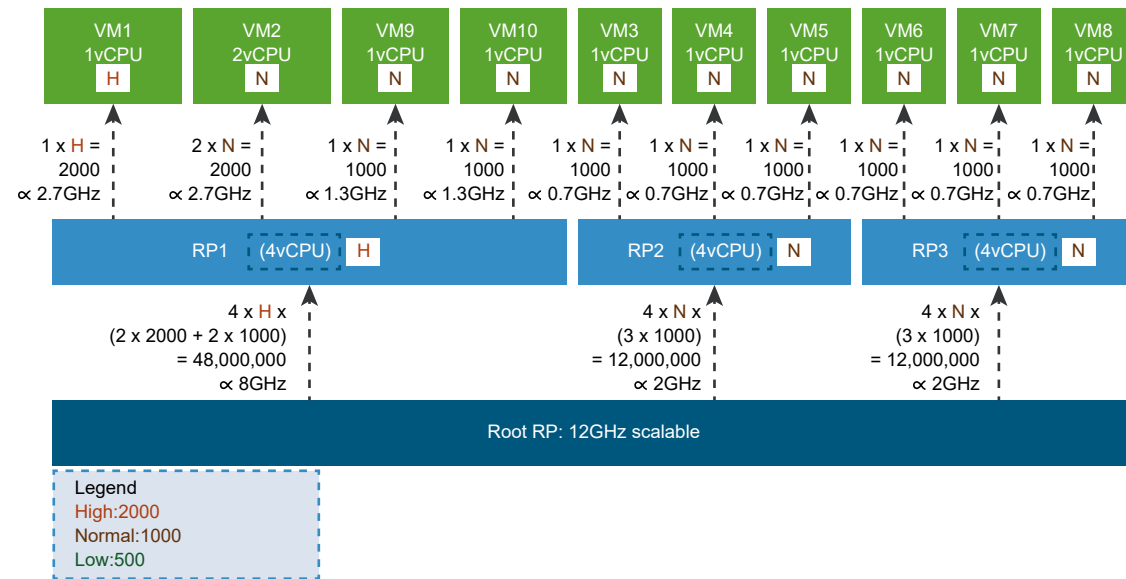
Note For simplicity of illustration, these examples deal only with CPU shares. A real configuration would include both CPU and memory resources, which act independently in regard to calculating the absolute resources available to a running virtual machine.

Figure 16-12. Example Data Center Configured with Fixed Shares



As a result of resource dilution within RP1 in a fixed shares configuration, some of RP1's virtual machines are entitled to exactly the same resource allocation as RP2's virtual machines, even though RP2 is configured as `normal` priority but RP1 is configured as `high` priority. Some users in Division 1 are disappointed because they expected their virtual machines to receive twice the allotment of RP2's virtual machines. The company's CIO agrees to reconfigure the data center to take advantage of the scalable shares option.

Figure 16-13. Example Scalable Shares Configuration



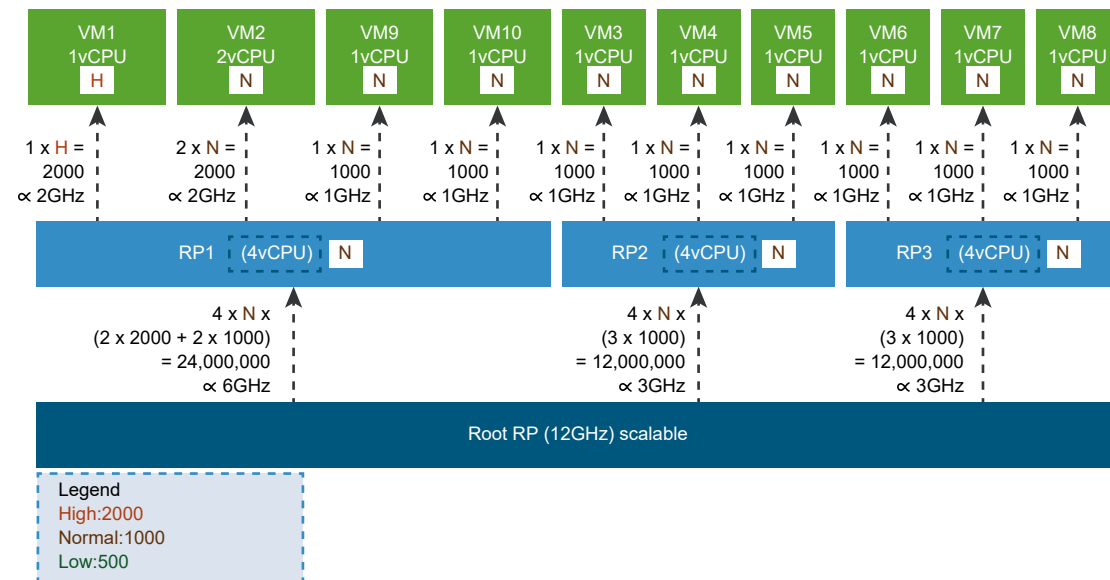
With scalable shares enabled on a parent resource pool, resource entitlements change. Because RP1 has more virtual machines than RP2 or RP3, it is entitled to a larger fraction of the root resource pool's resources during a resource contention situation. This is represented mathematically by multiplying the resource pool's numeric shares by the total of the shares of its children.

Calculating resource pool allocation in a scalable shares configuration is conceptually a six-step recursive process, working from leaf to root of the resource pool tree:

- 1 Convert the priority levels of its child virtual machines into numeric shares equivalents, in the same way as in a fixed shares configuration: Multiply `VirtualMachine.config.hardware.numCPU` by a constant that depends on the priority level. The constants are 2000, 1000, and 500 for high, normal, and low respectively.
- 2 If the resource pool has child resource pools, convert their priority levels into numeric shares equivalents, recursively.
- 3 Do the same conversion for the resource pool itself. For the purpose of this conversion, all resource pools are assigned an implicit size of 4 vCPUs.
- 4 Sum the children's numeric shares and multiply the sum by the parent's numeric shares.
- 5 Repeat steps 1-4 for all the parent's siblings.
- 6 Allocate the available resources among all siblings, in proportion to the products from steps 4 and 5.

As a result of the scalable shares configuration, Division 1 users find their virtual machine performance improved, but Divisions 2 and 3 suffer by comparison. To rectify this, the IT department resets the priority levels of RP1, RP2, and RP3 to `normal`. This adjustment leaves virtual machine performance comparable across divisions: `high` priority virtual machines get twice the resource allocation of `normal` priority virtual machines, even across resource pool boundaries.

Figure 16-14. Scalable Shares Configuration with All Resource Pools Configured at `normal` Level



A best practice when configuring scalable shares is to configure resource pools to the `normal` priority level and apply `high` priority levels or `low` priority levels to specific virtual machines.

Deleting Child Resource Pools

The `ResourcePool.DestroyChildren` method recursively deletes all the child resource pools of a resource pool. The operation takes a single parameter, a reference to the parent `ResourcePool` managed object. Any virtual machines associated with the child resource pool are reassigned to the parent resource pool.

Moving Resource Pools or Virtual Machines Into a Resource Pool

You can move a resource pool and its children within a resource pool hierarchy.

The `ResourcePool.MoveIntoResourcePool` method lets you move virtual machines, virtual applications, or resource pool hierarchies into a new resource pool. You call the method with an array of `ResourcePool` or `VirtualMachine` managed object references that you want to move. The whole resource pool hierarchy, including child resource pools and virtual machines, is moved when you move a resource pool.

Minimum available resources of the immediate children must always be less than or equal to the resources of the immediate parent. The root resource pool cannot be moved.

Introduction to vSphere Clusters

Clusters are useful with VMware DRS (distributed resource scheduler) and VMware HA (high availability). Clusters can be created quickly with VMware HCI. This guide briefly introduces the vSphere interfaces for cluster configuration. Clusters are useful to reduce power use, downtime, and maintenance.

- VMware DRS. *vSphere Resource Management* guide
- VMware HA. *vSphere Availability* guide
- VMware HCI, Hyper-converged infrastructure. In vSphere Client, click **New Cluster**.

VMware DRS

A VMware DRS cluster is a collection of ESXi hosts and associated virtual machines with shared resources and a shared management interface. To obtain the benefits of cluster-level resource scheduling you create a DRS cluster.

When you add a host to a DRS cluster, the host's resources become part of the cluster's resources. In addition to aggregation of resources, a DRS cluster supports cluster-wide resource pools and enforces cluster-level resource allocation policies. The following cluster-level resource management capabilities are available.

- **Load Balancing.** The vCenter Server monitors distribution and use of CPU and memory resources for all hosts and virtual machines in the cluster. DRS compares these metrics to an ideal resource utilization given the attributes of the cluster's resource pools and virtual machines, the current demand, and the DRS Score, which measures the execution efficiency of the virtual machine. DRS then performs (or recommends) virtual machine migrations. When you first power on a virtual machine in the cluster, DRS attempts to maintain proper load balancing either by placing the virtual machine on an appropriate host or by recommending one.
- **Power Management.** When the VMware DPM (Distributed Power Management) feature is enabled, DRS compares cluster- and host-level capacity to the demands of the cluster's virtual machines, including recent historical demand. DPM places (or recommends placing) hosts in standby power mode if sufficient excess capacity is found. DPM powers on (or recommends powering on) hosts if capacity is needed. Depending on the resulting host power state recommendations, virtual machines might need to be migrated to and from the hosts.
- **Virtual Machine Placement.** You can control the placement of virtual machines on hosts within a cluster, by assigning DRS affinity or antiaffinity rules.

See [Managing DRS Clusters](#).

VMware HA

VMware HA supports high availability for virtual machines by pooling the hosts they reside on into a cluster. If one host fails, its virtual machines move to a different host in the cluster.

VMware HA monitors ESXi hosts and in the event of host failure, migrates virtual machines to hosts with capacity. When you add new virtual machines to a VMware HA cluster, VMware HA checks whether enough capacity to power on that virtual machine is available on that host or a different host.

See [Managing HA Clusters](#).

VMware HCI

VMware HCI manages the configuration of hyper-converged compute, storage, and network resources for a cluster, as a collection of identically configured ESXi hosts. This unified approach simplifies cluster configuration, offering a way to configure all resource types as a group, using a single interface.

VMware HCI clusters are ideal for newly established datacenters. Only conformant hosts are allowed to be a part of the HCI cluster. They must have the same CPU design and contain the minimum number of physical NICs specified in `DvsProfile`. Network settings are wiped to a clean state before configuration. Non-management port groups in standard switches are migrated to a virtual distributed switch.

The `ConfigureHCI_Task` method configures a VMware HCI (hyper converged infrastructure) cluster. The `ExtendHCI_Task` method expands and reconfigures a VMware HCI cluster. Before and after HCI cluster configuration, you can run the `ValidateHCIConfiguration` method to check that the `ClusterComputeResourceHCIConfigSpec` is correct.

The `BatchAddHostsToCluster_Task` method adds more hosts as needed to a VMware HCI cluster, but cannot change cluster configuration.

These configure, extend, and host add methods are composite tasks. That is, they perform multiple operations in a single task. For example, `ConfigureHCI_Task` can:

- Set up cluster services such as DRS, HA, and vMotion
- For an HCI cluster, verify that all hosts have same CPU vendor
- Add new standalone hosts and set state (lockdown)
- Move all hosts into the HCI cluster
- Configure vSAN datastore
- Configure virtual distributed switch and port groups

Automatic remediation of failed hosts is not attempted. The administrator must intervene to correct issues before attempting to place hosts back in the cluster. Remediated hosts must be ready to participate in the vSAN datastore.

Creating and Configuring Clusters

The vSphere Web Services SDK includes objects and methods for all cluster management tasks. Some documentation is available in the *vSphere API Reference*. For additional background and

details about the failover and load balancing behavior, see *vSphere Resource Management* and *vSphere Availability*.

Creating a Cluster

If your environment includes a vCenter Server and multiple ESXi hosts, you can create a cluster by calling the `Folder.CreateClusterEx` method. You pass in a name for the new cluster and a `ClusterConfigSpecEx` data object. In the data object, you can specify the following properties:

- VMware DRS

- `drsConfig` property in the `ClusterDrsConfigInfo` data object contains configuration information for the VMware DRS service. Properties in this object specify the cluster-wide (default) behavior for virtual machine and the threshold for generating cluster recommendations. You can enable and disable VMware DRS with the `ClusterDrsConfigInfo.enabled` property.
- `drsVmConfigSpec` property in the `ClusterDrsVmConfigSpec` data object points to a `ClusterDrsVmConfigInfo` data object that specifies the DRS configuration for a single virtual machine. `ClusterDrsVmConfigInfo` overrides the default DRS configuration for an individual virtual machine and allows you to specify the DRS behavior and whether DRS can perform migration or recommend initial placement for a virtual machine.

To update a DRS configuration, you call

`ComputeResource.ReconfigureComputeResource_Task` and pass in a `ClusterConfigSpecEx` object. In the `ClusterConfigSpecEx.drsVmConfigSpec` property, you can specify an array of `ClusterDrsVmConfigSpec` objects that define the configuration for individual virtual machines.

- `rulesSpec` property in the `ClusterRuleSpec` data object points a `ClusterRuleInfo` data object that specifies the affinity and antiaffinity rules DRS should use. See the *vSphere API Reference* entry for `ClusterRuleInfo`.
- VMware HA – distributed availability service (DAS)
- `dasConfig` property in the `ClusterDasConfigInfo` data object specifies HA service on the cluster. Properties in this object determine whether strict admission control is enabled, what the default virtual machine settings in this cluster are, whether VMware HA restarts virtual machines after host failure, and so on. See the *vSphere API Reference*.
 - `dasVmConfigSpec` property in `ClusterDasVmConfigSpec` object, where the `info` property is a `ClusterDasVmConfigInfo` data object that specifies the HA configuration for a single virtual machine. You can apply different settings to different virtual machines, or use the default specified in the `dasConfig` property for all virtual machines in the cluster.

- VMware HCI
 - The `ClusterComputeResourceHCIConfigInfo` data object contains properties to capture the configured hosts, virtual distributed switch settings, host configurations, and HCI workflow state. The HCI cluster includes vSphere facilities for DRS, HA, vMotion, and vSAN.
 - The `ClusterComputeResourceHCIConfigSpec` data object specifies the virtual distributed switch settings, host configurations, enhanced vMotion capability, and vSAN configuration.
 - Once configurations are specified, you create an HCI cluster with the `ConfigureHCI_Task` method, and reconfigure an HCI cluster with the `ExtendHCI_Task` method.

Adding a Host to a Cluster

The methods available for adding hosts to a cluster are useful under different circumstances. Each method returns a managed object reference to a task.

- `ClusterComputeResource.AddHost_Task` adds a host to the cluster. The host may be specified by a numeric IP address or a DNS resolvable name. If the cluster supports nested resource pools and you pass an optional `resourcePool` argument, the host's resource pool hierarchy is imported into the new nested resource pool. If a cluster does not support nested resource pools, the host resource pool hierarchy is discarded and all virtual machines on the host are added to the cluster's root resource pool.
- `ClusterComputeResource.MoveInto_Task` moves a host in the datacenter into a cluster, or from one cluster into another. The `MoveHostInto_Task` method is similar, with extra `resourcePool` parameter.
- `Folder.BatchAddStandaloneHosts_Task` adds a list of `newHosts` to the inventory as standalone hosts. This operation works though the list of hosts and returns, in the task result, a list of hosts that were successfully added.

Reconfiguring a Cluster

You can alter a vSphere cluster by calling the `ComputeResource.ReconfigureComputeResource_Task` method. You can alter an HCI cluster by calling the `ClusterComputeResource.ExtendHCI_Task` method.

To reconfigure a vSphere cluster, for instance to enable or disable VMware DRS or VMware HA, you can change properties in the `ComputeResourceConfigSpec` data object, and pass it in to `ReconfigureComputeResource_Task`. For relevant properties, see [Creating a Cluster](#). To enlarge a vSphere cluster, you can call `AddHost_Task`.

To enlarge an HCI cluster, possibly after adding hosts with `BatchAddStandaloneHosts_Task`, you can call `ClusterComputeResource.ExtendHCI_Task`, passing in the new hosts as parameter `hostInputs`.

To move a datacenter host into a cluster, or from one cluster to another if the host is in maintenance mode, you can call the `ClusterComputeResource.MoveInto_Task` method. The `MoveHostInto_Task` method is similar but allows moving the host into a nested resource pool.

To move hosts out of a cluster, and make them standalone hosts in the datacenter, you can call the `Folder.MoveIntoFolder_Task` method.

To remove a host from a cluster, and from the datacenter, you can call the `Folder.Destroy_Task` method, if you are granted the `Host.Inventory.RemoveHostFromCluster` privilege.

Managing DRS Clusters

The vSphere Client UI allows you to explore DRS cluster behavior, which is also described in the *vSphere Resource Management* guide. When DRS is running, it generates recommendations and associated information that result in a well balanced cluster, including:

- Initial placement of virtual machines
- Virtual machine migration for load balancing. Each migration recommendation has a rating, which you can find in the `ClusterRecommendation.rating` property. Client applications can choose to consider only high-priority migrations or migrations with multiple priority levels.
- Whether or not DRS clusters are valid, and have enough resources to start additional virtual machines.

DRS recommendations are stored in the `ClusterComputeResource.recommendation` property, which is an array of `ClusterRecommendation` data objects. Each `ClusterRecommendation` includes information about the action to perform and information you can use to display information to end users or for logging.

- Client applications can call `ClusterComputeResource.ApplyRecommendation` to apply one or more recommendations.
- For more fine-grained control, client applications can perform individual actions only. The `ClusterRecommendation.action` property is an array of `ClusterAction` objects. Each `ClusterAction` includes a target for the action and the type, which is a string that is one of the values of the `ActionType` enum (`HostPowerV1`, `MigrationV1`, `VmPowerV1`). Client applications can use the `ActionType` information to act on DRS recommendations by powering on hosts, migrating virtual machines, or powering on virtual machines by calling `Datacenter.PowerOnMultiVM_Task`.

Managing HA Clusters

You can add a host to an HA cluster by calling the `AddHost_Task` method, or move a host into a cluster by calling the `MoveHostInto_Task` method or similar.

You might have to call `HostSystem.ReconfigureHostForDAS_Task` to reconfigure the host for HA if the automatic HA configuration fails. See [Adding a Host to a Cluster](#).

Primary and Secondary Hosts

You can add a secondary host to a cluster by calling the `ClusterComputeResource.AddHost_Task` method, which requires that you specify the host name, port, and password for the host to be added as a `HostConnectSpec`.

When you add a host to a VMware HA cluster, an agent is uploaded to the host and configured to communicate with other agents in the cluster. The first five hosts added to the cluster are designated as primary hosts, and all subsequent hosts are designated as secondary hosts. The primary hosts maintain and replicate all cluster state and are used to initiate failover actions. If a primary host is removed from the cluster, VMware HA promotes another host to primary status.

Any host that joins the cluster must communicate with an existing primary host to complete its configuration (except when you are adding the first host to the cluster). At least one primary host must be functional for VMware HA to operate correctly. If all primary hosts are unavailable (not responding), no hosts can be successfully configured for VMware HA.

One of the primary hosts is also designated as the active primary host and its responsibilities include:

- Deciding where to restart virtual machines
- Keeping track of failed restart attempts
- Determining when it is appropriate to keep trying to restart a virtual machine.

Failure Detection and Host Network Isolation

Agents on the different hosts contact and monitor each other through the exchange of heartbeats, by default every second. If a 15-second period elapses without the receipt of heartbeats from a host, and the host cannot be pinged, the host is declared as failed. The virtual machines running on the failed host are restarted on the alternate hosts with the most available unreserved capacity (CPU and memory).

Host network isolation occurs when a host is still running, but it can no longer communicate with other hosts in the cluster. With default settings, if a host stops receiving heartbeats from all other hosts in the cluster for more than 12 seconds, it attempts to ping its isolation addresses. If this also fails, the host declares itself isolated from the network.

When the isolated host's network connection is not restored for 15 seconds or longer, the other hosts in the cluster treat that host as failed and try to fail over its virtual machines. However, when an isolated host retains access to the shared storage it also retains the disk lock on virtual machine files. To avoid potential data corruption, VMFS disk locking prevents simultaneous write operations to the virtual machine disk files. Therefore attempts to fail over the isolated host's virtual machines do not succeed. By default, the isolated host leaves its virtual machines powered on, but you can change HA host isolation response as described in the *vSphere Availability* guide.

Using VMware HA and DRS Together

Using VMware HA with VMware DRS combines automatic failover with load balancing. This combination can result in faster rebalancing of virtual machines after VMware HA has moved virtual machines to different hosts.

When VMware HA performs failover and restarts virtual machines on different hosts, its first priority is the immediate availability of all virtual machines. After the virtual machines have been restarted, those hosts on which they were powered on might be heavily loaded, while other hosts might be comparatively lightly loaded. VMware HA uses the CPU and memory reservation to determine failover, but the actual usage might be higher.

In a cluster using DRS and VMware HA with admission control turned on, virtual machines might not be evacuated from hosts entering maintenance mode because of resources reserved to maintain the failover level. You must manually migrate the virtual machines off of the hosts using VMotion.

When VMware HA admission control is disabled, failover resource constraints are not passed on to DRS and VMware Distributed Power Management (DPM). The constraints are not enforced.

- DRS evacuates virtual machines from hosts and place the hosts in maintenance mode or standby mode regardless of the effect this might have on failover requirements.
- VMware DPM powers off hosts (place them in standby mode) even if doing so violates failover requirements.

The VMware HCI cluster includes VMware DRS and VMware HA features, so all the above remarks apply.

Tasks and Scheduled Tasks

17

VMware vSphere uses an asynchronous client-server communication model. Methods that end with `_Task` are non-blocking, returning a reference to a `Task` managed object. You can use `Task` and `ViewManager` managed objects to monitor tasks, cancel certain tasks, and create custom tasks.

If you are using a vCenter Server system, the `ScheduledTaskManager` allows you to schedule your own tasks for a one-time run or for repeated runs.

This chapter includes the following topics:

- [Creating Tasks](#)
- [Accessing and Manipulating Multiple Tasks](#)
- [Understanding the ScheduledTaskManager Interface](#)
- [Using a TaskHistoryCollector](#)
- [Managing the HistoryCollector](#)
- [Sample Code Reference](#)

Creating Tasks

Each time a vSphere server runs a method, it creates a `Task` and a corresponding `TaskInfo` data object. Some methods run synchronously and return data as the `Task` completes. But methods that end with `_Task` run asynchronously, and return a reference to a `Task` that will be created and completed as a processor becomes available. They are created to perform the functions in a non-blocking manner. Therefore, you must use the reference to the `Task` to monitor the status and results of the `Task`. vSphere operations that include the suffix `_Task` in their names are asynchronous and return `Task` references.

The `Task` object provides information about the status of the invoked operation through its `TaskInfo` data object. An instance of `TaskInfo` populates the `info` property of the `Task` managed object at runtime. By monitoring properties of the `TaskInfo` object, a client application can take appropriate action when the `Task` completes, or can handle errors if the `Task` does not complete successfully.

When a vSphere server creates a `Task`, it also creates a `TaskEvent` object. The `TaskEvent` object contains a copy of the `TaskInfo` object (`TaskEvent.info`). The `TaskEvent` copy of the `TaskInfo` object is a snapshot of the `Task` state at the time of its creation. It does not change after it is created. To find the current status of the task, use the `Task.info.eventChainId` property.

Session Persistence

A `Task` and its associated objects are session specific, so they will not persist after the session is closed. When your client opens a session, you can only obtain information about the `Task` objects that your client is authorized to view.

Cancelling a Task

To cancel a `Task` that is still running, call the `Task.CancelTask` method, passing in the managed object reference to the `Task` you want to cancel, as shown in this example:

```
my_conn.cancelTask(taskMoRef);
```

You can only cancel a `Task` that has its `cancelable` property set to `true` and its `state` property set to `running`. The operation that initiates the `Task` sets the value of `cancelable` when it creates the `Task`. For example, a `CreateVM_Task` cannot be cancelled. Before attempting to cancel a running `Task`, you can check the values of the `cancelable` property and the `state` property of the `TaskInfo` data object associated with the `Task`.

Using TaskInfo to Determine Task Status

A `Task` object provides information about the status of the invoked operation through its `TaskInfo` data object. An instance of `TaskInfo` populates the `info` property of the `Task` managed object at runtime. By monitoring properties of the `TaskInfo` object, a client application can take appropriate action when the `Task` completes, or can handle errors if the `Task` does not complete successfully.

The `Task.info` property contains a `TaskInfo` data object that contains information about the `Task` the server returns to your client application.

When a `Task` is instantiated by the server, the `TaskInfo.result` property is initialized to `Unset`. Upon successful completion of an operation, the `result` property is populated with the return type specific to the operation. The `result` might be a data object, a reference to a managed object, or any other data structure as defined by the operation.

For example:

- 1 The `ClusterComputeResource.AddHost_Task` method returns a `Task` object whose `info` property contains a `TaskInfo` data object.
- 2 At the start of the operation, the `result` property is `Unset`.

- 3 Upon successful completion of the operation, the `result` property of `TaskInfo` contains the managed object reference of the newly added `HostSystem`.

TaskInfo Values

The following table lists some of the values obtained from a `TaskInfo` data object at the beginning and the end of the `Task` instantiated by the `CreateVM_Task` method.

Property	Datatype	Start of Task Sample Values	End of Task Sample Values
<code>cancelable</code>	<code>boolean</code>	<code>false</code>	<code>false</code>
...			
<code>completeTime</code>	<code>dateTime</code>	<code>Unset</code>	<code>"2009-02-19T22:53:35.015338Z"</code>
<code>progress</code>	<code>int</code>	<code>36</code>	<code>100</code>
<code>queueTime</code>	<code>dateTime</code>	<code>"2009-02-19T22:50:39.111604Z"</code>	<code>"2009-02-19T22:50:39.111604Z"</code>
<code>reason</code>	<code>TaskReason</code>	<code>reason</code>	<code>reason</code>
<code>result</code>	<code>anyType</code>	<code>Unset</code>	<code>64</code>
....			
<code>state</code>	<code>TaskInfoState</code>	<code>"running"</code>	<code>"success"</code>

Monitoring TaskInfo Properties

To monitor the state of a `Task`, use the `PropertyCollector.WaitForUpdatesEx` method.

You can monitor the values of `TaskInfo` properties, which change as the `Task` runs to completion. For example, you can check the values of `startTime`, `queueTime`, `completeTime`, `progress`, `result`, and `state` as the operation progresses. Monitor these properties in your code in a separate thread until the `Task` completes, while the main line of your code continues with other activities.

For more information about monitoring properties, see [Client Data Synchronization \(WaitForUpdatesEx\)](#).

Your code must handle the datatype returned when the `Task` completes (managed object reference, data object, and so on). In addition to `success`, `queued`, and `running`, an operation can enter an `error` state, which your code must handle.

A `Task` object has a lifecycle that is independent of the `TaskManager` that creates it and independent of the entity with which it is associated. It exists to convey status about an operation. You can discard the reference to it when your application no longer needs the information.

The following example shows a code fragment that obtains values for the `info` property from each `Task` object in the array.

Example: Displaying TaskInfoState Values for Tasks in recentTask Array

```
...
private void displayTasks(ObjectContent[] oContents) {
    for(int oci=0; oci<oContents.length; ++oci) {
        System.out.println("Task");
        DynamicProperty[] dps = oContents[oci].getPropSet();
        if(dps!=null) {
            String op="", name="", type="", state="", error="";
            for(int dpi=0; dpi<dps.length; ++dpi) {
                DynamicProperty dp = dps[dpi];
                if("info.entity".equals(dp.getName())) {
                    type = ((ManagedObjectReference)dp.getVal()).getType();
                } else if ("info.entityName".equals(dp.getName())) {
                    name = ((String)dp.getVal());
                } else if ("info.name".equals(dp.getName())) {
                    op = ((String)dp.getVal());
                } else if ("info.state".equals(dp.getName())) {
                    TaskInfoState tis = (TaskInfoState)dp.getVal();
                    if(TaskInfoState.error.equals(tis)) {
                        state = "-Error";
                    } else if(TaskInfoState.queued.equals(tis)) {
                        state = "-Queued";
                    } else if(TaskInfoState.running.equals(tis)) {
                        state = "-Running";
                    } else if(TaskInfoState.success.equals(tis)) {
                        state = "-Success";
                    }
                } else if ("info.cancelled".equals(dp.getName())) {
                    Boolean b = (Boolean)dp.getVal();
                    if(b != null && b.booleanValue()) {
                        state += "-Cancelled";
                    }
                }
            }
        }
    }
}
```

[Sample Run of the TaskList Java Application](#) shows output from a run of the program. See the source code listing for `TaskList.java` or for `TaskList.cs` in the vSphere Web Services SDK package for details.

Example: Sample Run of the TaskList Java Application

```
java com.vmware.samples.general.TaskList --url https://srv/sdk --username root --password
*****

Started
Task
Operation AcquireCimServicesTicket
Name srv
Type HostSystem
State -Success
```

```
Error
=====
Ended TaskList
```

Accessing and Manipulating Multiple Tasks

Use the `ViewManager`'s `ListView` method to identify the set of `Tasks` you want to monitor.

You can specify a smaller and more efficient data set using one of the `ViewManager` views with the `PropertyCollector`. Each view represents objects you have selected on the server. Views are more efficient because you only need a single instance of a `PropertyCollector` object, instead of multiple instances with multiple filter specifications.

Task Monitoring Example Using the Listview Object

You can use the `ViewManager`'s `CreateListView` method to specify a set of tasks that you want to monitor with the `PropertyCollector`. This example shows how to initiate virtual machine cloning operations and monitor completion by tracking the `Task` objects.

For general task monitoring, a best practice is to use a `ViewManager` to monitor specific tasks. See the *API Reference* for more information about using views.

Note For simplicity, the calls to `RetrievePropertiesEx()` and `WaitForUpdatesEx()` in this example ignore the possibility of additional chunks of data returned by the property collector. For more information about chunking in property collector results, see [Server Data Transmission](#).

Procedure

- 1 [Create an Authenticated Session with the Server](#)

Use command-line arguments to connect to a vSphere server and log in.

- 2 [Find Virtual Machine MObj](#)

Use the `PropertyCollector` to get a managed object reference for a virtual machine named in the command line.

- 3 [Get_Parent_Property](#)

Given the managed object reference for a virtual machine object, use the `PropertyCollector` to get the managed object reference for its parent object.

- 4 [Create Clones of the Virtual Machine](#)

Create two clones of the virtual machine and build a `PropertyFilterSpec` that uses a `ListView` managed object to track the `Tasks` to completion.

- 5 [Monitor Tasks with WaitForUpdatesEx](#)

Use the `PropertyCollector.WaitForUpdatesEx` method to monitor and report `Task` state changes.

Create an Authenticated Session with the Server

Use command-line arguments to connect to a vSphere server and log in.

Given the arguments on the command line, this task makes an SSL connection to the vSphere server, retrieves a reference to the SessionManager, and authenticates a session with the credentials passed on the command line.

Prerequisites

For this task you need:

- The Web Services API WSDL bindings file.
- The hostname of the vSphere server.
- A username and password to authenticate with the vSphere server.
- The name of a virtual machine available to be cloned.

Procedure

- 1 Import the vSphere Web Services API libraries and the necessary Java (and JAX-WS connection, bindings, and SOAP) libraries:

```
import com.vmware.vim25.*;

import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;
```

- 2 Create a class to hold data and methods.

```
public class cloneVMTask {
    static ManagedObjectReference pCollector;
    static ManagedObjectReference viewMgr;
    static ServiceContent serviceContent;
    static VimPortType methods;
```

- 3 Define a TrustManager implementation.

Authentication is handled by using a TrustManager and supplying a host name verifier method.

Note For the purposes of this example, this TrustManager implementation will accept all certificates. This is only appropriate for a development environment. Production code should implement certificate validation.

```
private static class TrustAllTrustManager
    implements javax.net.ssl.TrustManager,
               javax.net.ssl.X509TrustManager {
```

```

public java.security.cert.X509Certificate[] getAcceptedIssuers() {
    return null;
}
public boolean isServerTrusted(java.security.cert.X509Certificate[] certs) {
    return true;
}
public boolean isClientTrusted(java.security.cert.X509Certificate[] certs) {
    return true;
}
public void checkServerTrusted(java.security.cert.X509Certificate[] certs,
                               String authType)
    throws java.security.cert.CertificateException {
    return;
}
public void checkClientTrusted(java.security.cert.X509Certificate[] certs,
                               String authType)
    throws java.security.cert.CertificateException {
    return;
}
} // end TrustAllTrustManager

```

4 Read command line arguments.

```

public static void main(String [] args) throws Exception {
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    String vmName = args[3];
    String url = "https://" + serverName + "/sdk/vimService";
}

```

5 Create SSL context and session context for the connection.

```

javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());

```

6 Set a host name verifier that will enable any connection.

The host name verifier is invoked during the SSL handshake.

```

HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {

```

```

        return true;
    }
};
HttpsURLConnection.setDefaultHostnameVerifier(hv);

```

7 Create a VimService object to obtain a VimPort binding provider.

The `BindingProvider` provides access to the protocol fields in request/response messages. Retrieve the request context which will be used for processing message requests.

```

VimService vimService = new VimService();
methods = vimService.getVimPort();
Map<String, Object> ctxt = ((BindingProvider) methods).getRequestContext();

```

8 Store the Server URL in the request context.

Specify `true` to maintain the connection between the client and server. The client API will include the Server's HTTP cookie in its requests to maintain the session. If you do not set this to `true`, the Server will start a new session with each request.

```

ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

```

9 Create a MOref for the ServiceInstance and retrieve the ServiceContent.

```

// -- ManagedObjectReference for the ServiceInstance on the Server
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
serviceContent = methods.retrieveServiceContent(SVC_INST_REF);

```

10 Use the SessionManager to log in.

```

methods.login(serviceContent.getSessionManager(),
             userName,
             password,
             null);

```

11 Get references to the PropertyCollector and the ViewManager.

```

pCollector = serviceContent.getPropertyCollector();
viewMgr = serviceContent.getViewManager();

```

What to do next

With an authenticated session, you can use the `PropertyCollector` to retrieve the `MOref` of the virtual machine, then create clones and monitor the `Task` objects:

```

ManagedObjectReference vmRef = getVmRef( vmName );
cloneVM( vmRef );

```

A best practice is to close the session when done:

```
methods.logout(serviceContent.getSessionManager());
} // end main
} // end cloneVMTask
```

Find Virtual Machine MObjRef

Use the PropertyCollector to get a managed object reference for a virtual machine named in the command line.

Given the name of a virtual machine, this function retrieves references to all of the virtual machines in the datacenter and looks for a match to the specified name.

Prerequisites

For this task you need:

- A virtual machine name is referenced by the String variable `vmName`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A VimPort binding provider is referenced by the variable `methods`, which is attached to the vSphere server connection context.
- The ServiceInstance content is referenced by the variable `serviceContent`.
- The ViewManager object is referenced by the variable `viewMgr`.
- The PropertyCollector object is referenced by the variable `pCollector`.

Procedure

- 1 Declare a function that accepts a virtual machine name and returns a managed object reference.

```
private static ManagedObjectReference getVmRef( String vmName )
throws Exception
{
    ManagedObjectReference vmRef = null;
```

- 2 Use a container view to collect references to all virtual machines in the datacenter.

Start the collection from the root folder at the top of the managed object hierarchy.

```
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");
ManagedObjectReference cViewRef =
    methods.createContainerView(viewMgr,
                                serviceContent.getRootFolder(),
                                vmList,
                                true);
```

3 Create an `ObjectSpec` to define the beginning of the traversal.

Use the `setObj` method to specify that the container view is the starting object for this traversal. Set the `skip` property to `true` to indicate that you don't want the `PropertyCollector` to collect properties from the container.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cViewRef);
oSpec.setSkip(true);
```

4 Create a traversal spec to traverse from the starting object to the objects in the view.

Set the `skip` property to `false` to indicate that you don't want the `PropertyCollector` to skip the objects reached by this traversal.

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseEntities");
tSpec.setType("ContainerView");
tSpec.setPath("view");
tSpec.setSkip(false);
```

5 Add the traversal spec to the object spec.

```
oSpec.getSelectSet().add(tSpec);
```

6 Specify the property for retrieval (virtual machine name).

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("name");
```

7 Create a `PropertyFilterSpec` and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

8 Create a list for the filters and add the spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

9 Get the data from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector, fSpecList, ro);
```

10 Go through the returned list and look for a match to the specified `vmName`.

```
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        String vmname = null;
```



```

List<DynamicProperty> dps = oc.getPropSet();
if (dps != null) {
    for (DynamicProperty dp : dps) {
        vmname = (String) dp.getVal();
        // If the name of this virtual machine matches
        // the specified name, save the managed object reference.
        if (vmname.equals(vmName)) {
            vmRef = oc.getObj();
            break;
        }
    }
    if (vmRef != null) { break; }
}
}
if (vmRef == null) {
    System.out.println("Specified Virtual Machine not found.");
    throw new Exception();
}
return vmRef;
} // end getVmRef

```

What to do next

Given the MOref of the named virtual machine, you can use the PropertyCollector to retrieve its MOref to its parent folder, which you will need to clone the virtual machine.

Get_Parent_Property

Given the managed object reference for a virtual machine object, use the PropertyCollector to get the managed object reference for its parent object.

Given the MOref of a virtual machine, this function uses the PropertyCollector to retrieve the MOref of the parent of the virtual machine object. The MOref of the parent folder can be used to clone the virtual machine.

A `VirtualMachine` can be a child of either a `Folder` object or a `VirtualApp` object. If it is a child of a `Folder` object, the `parent` property contains the MOref of the `Folder` object and the `parentVApp` property is null. If the `VirtualMachine` is a child of a `VirtualApp` object, the `parent` property is null and the `parentVApp` property contains the MOref of the `VirtualApp` object. The method in this example returns a key-value pair from which the caller can determine which kind of parent the `VirtualMachine` has.

Prerequisites

For this task you need:

- A virtual machine managed object reference in a variable named `vmRef`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.

- A VimPort binding provider referenced by the variable `methods`, which is attached to the vSphere server connection context.
- A PropertyCollector instance referenced by the variable `pCollector`.

Procedure

- 1 Declare a function that accepts a virtual machine MOref and returns a MOref to its parent object.

```
private static DynamicProperty getVMParentProperty(ManagedObjectReference vmRef)
throws Exception {
```

- 2 Create an `ObjectSpec` to define the property collection. Use the `setObj` method to specify that the `vmRef` is the starting object for this property collection. Set the `skip` property to `false` to indicate that you want to collect properties from the starting object. Omit the `selectSet` property because the property collection does not need to traverse away from the starting object.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(false);
```

- 3 Specify the properties for retrieval (`VirtualMachine.parent` and `VirtualMachine.parentVApp`).

Because the `VirtualMachine` has two mutually exclusive parent properties, depending on the parent type, this code collects both properties.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.getPathSet().add("parent");
pSpec.getPathSet().add("parentVApp");
```

- 4 Create a `PropertyFilterSpec` and add the object and property specs to it.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

- 5 Create a list for the filters and add the property filter spec to it.

```
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);
```

- 6 Use the filter spec to retrieve the property collection from the server.

```
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(pCollector, fSpecList, ro);
```

7 Unwrap the parent folder property.

```

    if (props != null) {
        for (ObjectContent oc : props.getObjects()) {
            List<DynamicProperty> dps = oc.getPropSet();
            if (dps != null) {
                for (DynamicProperty dp : dps) {
                    if (dp.getName().equals("parent") || dp.getName().equals("parentVApp")) {
                        return dp;
                    }
                }
            }
        }
    }
    System.out.println("Parent not found.");
    throw new Exception();
}

```

What to do next

Now that you have the reference information for the virtual machine that you specified on the command line (`vmRef`) and a reference for the parent (in the `dp` property), you are ready to extract the type and MOref of the parent.

Create Clones of the Virtual Machine

Create two clones of the virtual machine and build a `PropertyFilterSpec` that uses a `ListView` managed object to track the Tasks to completion.

Given the MOref of a virtual machine and the MOref of its parent folder, this function initiates two clone operations, resulting in two `Task` managed objects that you can use to monitor progress of the clone operations. The function uses the `Task` objects to build a `PropertyFilterSpec` that you can use to report status updates for the two `Task` objects.

Prerequisites

For this task you need:

- A virtual machine managed object reference in a variable named `vmRef`.
- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- A `VimPort` binding provider referenced by the variable `methods`, which is attached to the vSphere server connection context.
- The `ViewManager` object is referenced by the variable `viewMgr`.
- A `PropertyCollector` instance referenced by the variable `pCollector`.

Procedure

- 1 Declare a method that accepts a virtual machine MOref to clone it and monitor the `Task` objects.

```
private static void cloneVM(ManagedObjectReference vmRef) throws Exception {
```

- 2 Use the `getVMParent` method to get a MOref for the parent folder of the virtual machine.

If the parent object is a `Folder`, the `CloneVMTask` method will create the clones in the same folder that contains the specified virtual machine. If the parent object is a `VirtualApp`, the `VirtualMachine` cannot be cloned.

```
DynamicProperty parent = getVMParentProperty( vmRef );
if (parent.getName().equals("parentVApp")) {
    System.out.println("Will not clone VM in a VApp.");
    throw new Exception();
} else {
    ManagedObjectReference folder = (ManagedObjectReference) parent.getVal();
}
```

- 3 Create a clone specification. Use default values whenever possible.

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineRelocateSpec vmrs = new VirtualMachineRelocateSpec();
cloneSpec.setLocation(vmrs);
cloneSpec.setPowerOn(true);
cloneSpec.setTemplate(false);
```

- 4 Create two clone virtual machines.

```
ManagedObjectReference cloneTask = methods.cloneVMTask( vmRef,
                                                         folder,
                                                         "clone__1",
                                                         cloneSpec);

ManagedObjectReference cloneTask2 = methods.cloneVMTask( vmRef,
                                                         folder,
                                                         "clone__2",
                                                         cloneSpec);
```

The `CloneVMTask` method returns a `Task` object from each invocation. You can use `PropertyCollector` to track task completion.

- 5 Create a list view to reference the clone tasks.

```
List<ManagedObjectReference> taskList = new ArrayList<ManagedObjectReference>();
taskList.add(cloneTask);
taskList.add(cloneTask2);
ManagedObjectReference cloneTaskList = methods.createListView(viewMgr,
                                                                taskList);
```

6 Create an object spec to describe the property collection.

Add the list of `Task` objects to the object spec. Set the `skip` property to `true` so the `PropertyCollector` will ignore the properties of the `ListView` object itself.

```
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(cloneTaskList);
oSpec.setSkip(true);
```

7 Create a traversal spec to tell the `PropertyCollector` how to move from the `ListView` object to the `Task` objects.

Set the `skip` property to `false` so the `PropertyCollector` will try to collect properties from the managed objects at the end of this traversal step.

```
TraversalSpec tSpec = new TraversalSpec();
tSpec.setName("traverseTasks");
tSpec.setPath("view");
tSpec.setSkip(false);
tSpec.setType("ListView");
```

8 Add the traversal spec to the object spec.

```
oSpec.getSelectSet().add(tSpec);
```

9 Create a property spec to collect the `Task.info.state` and `Task.info.result` properties.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("Task");
pSpec.setAll(false);
pSpec.getPathSet().add("info.state");
pSpec.getPathSet().add("info.result");
```

10 Create a filter spec from the object spec and the property spec, and use the filter spec to create a `PropertyFilter` object to guide the property collection.

The filter managed object needs to persist on the server until the `PropertyCollector` finishes reporting task status.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
ManagedObjectReference pFilter = methods.createFilter(pCollector,
                                                    fSpec,
                                                    true);
```

Results

Two cloning operations are underway, with status of the operations reflected in `Task` objects. A `PropertyFilter` managed object is created and configured to monitor the task status.

What to do next

Use the `PropertyCollector.WaitForUpdatesEx` method to monitor the status of the `Task` objects.

Monitor Tasks with `WaitForUpdatesEx`

Use the `PropertyCollector.WaitForUpdatesEx` method to monitor and report `Task` state changes.

This example uses the `waitForUpdatesEx` method to look for a change in the `info.state` property and the `info.result.property` of `cloneTask` and `cloneTask2`. When the state is "success", `info.result` is the managed object reference of the clone virtual machine.

Note The order of property retrieval is not guaranteed, and it may take more than one call to `waitForUpdatesEx` to retrieve both properties for a task

Prerequisites

For this task you need:

- An authenticated Web Services session with the vSphere server that manages the virtual machine.
- Managed object references for two `Task` objects in the variables `cloneTask` and `cloneTask2`.
- The `ViewManager` object is referenced by the variable `viewMgr`.
- The `PropertyCollector` object is referenced by the variable `pCollector`.

Procedure

- 1 Initialize variables for the wait loop.

```
String version = "";
Boolean wait = true;
WaitOptions waitOptions = new WaitOptions();

while ( wait ) {
```

- 2 Call `WaitForUpdatesEx`.

If `waitForUpdatesEx` returns `null`, that indicates that the call has timed out.

```
UpdateSet uSet = methods.waitForUpdatesEx(pCollector,
                                          version,
                                          waitOptions);

if (uSet == null) {
    wait = false;
} else {
```

3 Save the version for subsequent calls to `WaitForUpdatesEx`.

```
version = uSet.getVersion();
```

4 Get the list of property updates.

```
List<PropertyFilterUpdate> pfUpdates = uSet.getFilterSet();
for (PropertyFilterUpdate pfu : pfUpdates) {
```

5 Get the list of object updates produced by the filter.

```
List<ObjectUpdate> oUpdates = pfu.getObjectSet();
for (ObjectUpdate ou : oUpdates) {
```

6 Look for `ObjectUpdate.kind=="MODIFY"` (property modified).

```
if (ou.getKind() == ObjectUpdateKind.MODIFY) {
    String name = "";
    TaskInfoState state;
    ManagedObjectReference cloneRef = new ManagedObjectReference();
```

7 Get the changed data.

```
List<PropertyChange> pChanges = ou.getChangeSet();
```

8 Output the property changes.

If the task completes successfully, the `result` property contains the `MOref` of the new clone.

```
for (PropertyChange pc : pChanges) {
    name = pc.getName();
    if (name.equals("info.state")) {
        state = (TaskInfoState)pc.getVal();
        System.out.println("State is "+state.value());
    } else if (name.equals("info.result")) {
        cloneRef = (ManagedObjectReference)pc.getVal();
        System.out.println("Clone reference is "+cloneRef.getValue());
    } // end if name.equals
} // end for pc
} // end if ou.getKind
} // end for ou
} // end for pfu
} // end if user
} // end while wait
} // end cloneVM
```

Results

This code does not set a time-out (`WaitOptions.maxWaitSeconds` is unset), so after it has retrieved all of the property changes, `waitForUpdatesEx` will block the thread, waiting for the TCP connection with the vSphere Server to time out.

How a client application handles the session depends on the particular context. For example, the client can call `WaitForUpdatesEx` from its own thread, look for specific updates and then stop calling the method. For the preceding example, you might choose to watch for the `Task.info.state` to transition to a value of `success` or `error`.

For more information about `WaitOptions` and the `waitForUpdatesEx` method, see [Client Data Synchronization \(WaitForUpdatesEx\)](#).

Gathering Data with a TaskManager Interface

`TaskManager` is a service interface that you can also use for accessing and manipulating `Task` objects. This approach uses a `PropertyCollector` that includes the `recentTask` property of the `TaskManager` managed object that corresponds to the Recent Tasks pane at the bottom of the vSphere client User Interface.

You can use the following `TaskManager` properties in your client application.

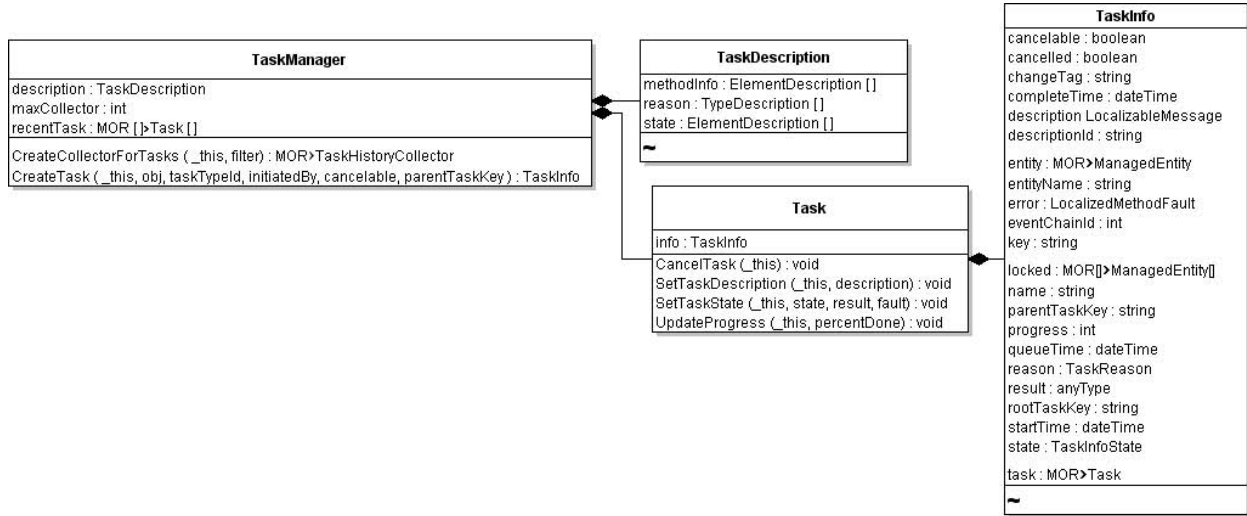
- `description` – `TaskDescription` object that includes a `MethodInfo` property. `MethodInfo` contains a key-based array that `TaskManager` uses to populate the value of a `TaskInfo` data object's `descriptionId` property with the name of the operation. Examples of two elements from this key-based array are `MethodInfo["Folder.createVm"]` and `MethodInfo["Folder.createClusterEx"]`.
- `recentTask` – Array of `Task` managed object references that are queued to run, running, or completed within the past 10 minutes. On ESX/ESXi hosts that are managed by a vCenter Server, a completed task must also be one of the 200 most recent tasks to be included in the array. A vSphere Client connected to a vSphere Server displays queued, running, and completed tasks in the Recent Tasks pane.

In addition to these properties, `TaskManager` has the following methods:

- `CreateTask` – Used by other methods to create a custom `Task` object. Developers creating extensions can use this method to create custom `Task` objects.
- `CreateCollectorForTasks` – Creates an object that contains all tasks from the vCenter Server database that meet specific criteria. You cannot run this method against an ESX/ESXi system. See [Using a TaskHistoryCollector](#).

Figure 17-1. TaskManager and Task Managed Objects shows a UML class diagram for `TaskManager` and associated objects.

Figure 17-1. TaskManager and Task Managed Objects



Examining Recent Tasks with TaskManager

To obtain the list of recent tasks, use a `PropertyCollector` to obtain references to the `TaskManager` and to all `Task` objects from the `recentTask` property of the `TaskManager`.

The following example shows an excerpt from the `TaskList.java` sample that creates the `ObjectSpec`, `PropertySpec`, and a `TraversalSpec` to obtain references to all `Task` objects on the server from the `TaskList`. See also [Chapter 6 Property Collector](#).

Example: PropertyFilterSpec Definition to Obtain recentTask Property Values

```

private PropertyFilterSpec[] createPFSForRecentTasks(ManagedObjectReference taskManagerRef) {
    PropertySpec pSpec = new PropertySpec();
    pSpec.setAll(Boolean.FALSE);
    pSpec.setType("Task");
    pSpec.setPathSet(new String[] {"info.entity", "info.entityName", "info.name",
        "info.state", "info.cancelled", "info.error"});
    ObjectSpec oSpec = new ObjectSpec();
    oSpec.setObj(taskManagerRef);
    oSpec.setSkip(Boolean.FALSE);
    TraversalSpec tSpec = new TraversalSpec();
    tSpec.setType("TaskManager");
    tSpec.setPath("recentTask");
    tSpec.setSkip(Boolean.FALSE);
    oSpec.setSelectSet(new SelectionSpec[] {tSpec});
    PropertyFilterSpec pfSpec = new PropertyFilterSpec();
    pfSpec.setPropSet(new PropertySpec[] {pSpec});
    pfSpec.setObjectSet(new ObjectSpec[] {oSpec});
    return new PropertyFilterSpec[] {pfSpec};
}
  
```

For ESXi hosts managed by vCenter Server, use a `TaskHistoryCollector`. See [Using a TaskHistoryCollector](#).

Understanding the ScheduledTaskManager Interface

You can use the `ScheduledTaskManager` to schedule tasks. In the vSphere Client, scheduled tasks display in the Task & Events tab.

You can define actions to occur on vCenter Server at different times:

- When a vCenter Server system starts up operations, such as after a reboot
- At a specific time and day
- At hourly, daily, weekly, or monthly intervals

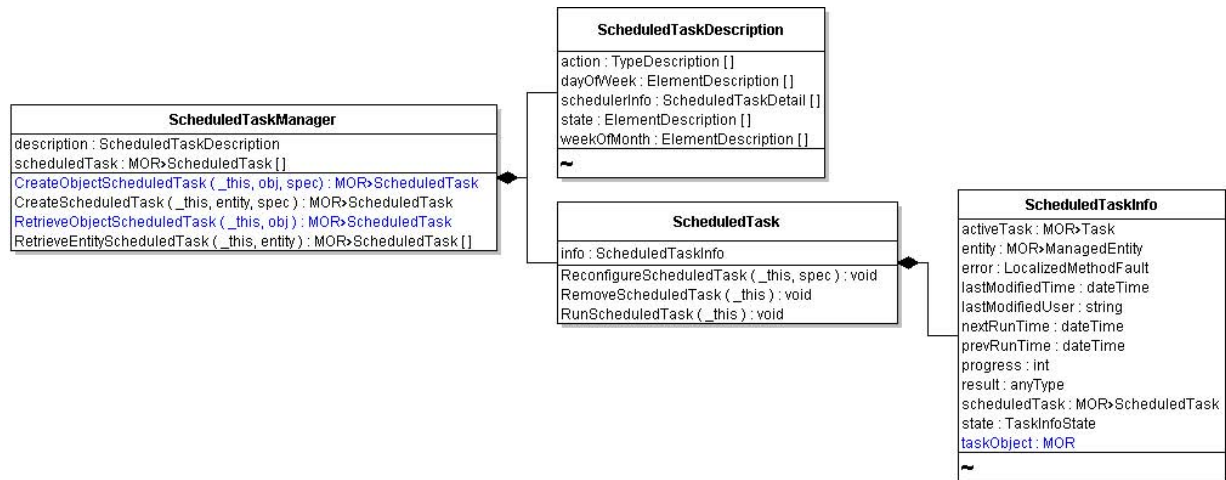
You can schedule scripts to be run or methods to be invoked on the server. You apply the action to an entity in the inventory, such as a virtual machine or a host.

You can perform the following actions with `ScheduledTaskManager`.

- Retrieve scheduled tasks for a specific managed entity by calling the `ScheduledTaskManager.RetrieveEntityScheduledTask` method.
- Create a scheduled task by calling the `ScheduledTaskManager.CreateScheduledTask` method. See [Scheduling Tasks](#).

Figure 17-2. `ScheduledTaskManager` and `ScheduledTask` Managed Objects shows the `ScheduledTaskManager` service interface and associated data objects.

Figure 17-2. `ScheduledTaskManager` and `ScheduledTask` Managed Objects



The `ScheduledTaskManager.scheduledTask` property contains an array of the `ScheduledTask` objects configured for the server. If you have no actions scheduled, this property is empty. For any `ScheduledTask` objects in this array, you can use the `info` property of the `ScheduledTask` object to obtain information about the status of the scheduled action. Information includes the task's `progress`, `state`, previous and next runtimes, and other details contained in the `ScheduledTaskInfo` data object.

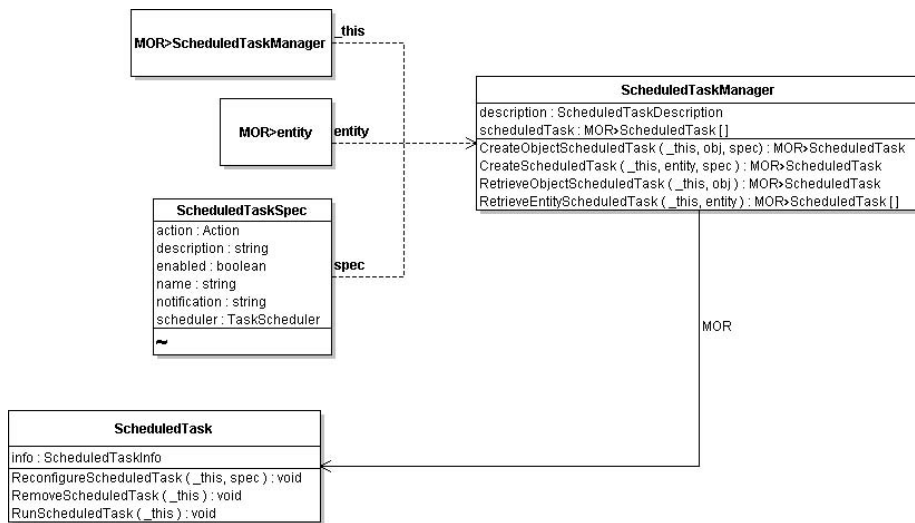
If the action specified for a `ScheduledTask` creates its own `Task` (such as with any of the asynchronous operations), the managed object reference to the `Task` populates the `activeTask` property of `ScheduledTaskInfo`.

Scheduling Tasks

You create a `ScheduledTask` by invoking the `ScheduledTaskManager.CreateScheduledTask` method. When you invoke the method, you include a `ScheduledTaskSpec` object that defines the schedule and specifies the action to take at the specified time. A scheduled action applies to an object based on these rules:

- If you specify a container object as the entity for the scheduled action, the schedule applies to all entities that are direct descendants of the container. You can set a `ScheduledTask` at the `Folder`, `Datacenter`, or `VirtualApp` level and have the scheduled action apply to all entities associated with the `Folder`, `Datacenter`, or `VirtualApp`.
- If you specify a node object in the inventory, such as a virtual machine, the action applies only to the virtual machine.

Figure 17-3. Using `ScheduledTaskManager` to Create a `ScheduledTask`



Defining the Schedule and Action

The `ScheduledTaskSpec` data object contains all the information to create a `ScheduledTask`.

- **action** – Action to take when the `ScheduledTask` runs. Specify an `Action` data object, which is an abstract type that is extended by several specific action types. The `Action` data objects are also used by the `Alarm` infrastructure. See [Specifying Alarm Actions](#).
- **notification** – Specifies the email address for sending notification messages about the `ScheduledTask`. To use notifications, the vCenter Server system must have an SMTP email gateway configured. By default, `notification` is set to an empty string.

- `scheduler` – Specifies the time, frequency, and other details of the schedule. The `TaskScheduler` data object is the base type for several specific schedule objects. See [Scheduling Recurring Operations](#).

Scheduling Recurring Operations

You can specify the times, days, or frequency of scheduled tasks by creating the appropriate instances of `TaskScheduler` subtypes and setting the `scheduler` property of the `ScheduledTaskSpec`.

The `TaskScheduler` base type has two properties:

- `activeTime` is the time at which the action should occur. If you leave this property unset, it defaults to the time when that specification for the scheduled task was submitted to the server.
- `expireTime` is the time after which the scheduled action should not occur. By default, this property is unset, so the scheduled task does not expire.

The following table provides some usage information about the `TaskScheduler` subtypes. The examples in the table are Java code fragments.

TaskScheduler Subtype	Usage
<code>AfterStartupTaskScheduler</code>	<p>Schedule a task to start as soon as the vCenter Server system is started, or at a defined time after startup. The value must be zero (task triggered at startup) or higher.</p> <p>Example: Schedule a task to run 10 minutes after vCenter Server startup.</p> <pre>AfterStartupTaskScheduler asts = new AfterStartupTaskScheduler(); asts.setMinute(10);</pre>
<code>OnceTaskScheduler</code>	<p>Schedule an action to run once only at the specified date and time.</p> <p>Example: Schedule a task to run 30 minutes after the schedule is submitted to the server.</p> <pre>Calendar runTime = Calendar.getInstance(); runtime.add(Calendar.MINUTE, 30); OnceTaskScheduler ots = new OnceTaskScheduler(); ots.setRunAt(runTime);</pre>
<code>RecurrentTaskScheduler</code>	<p>Base type for <code>HourlyTaskScheduler</code>, <code>DailyTaskScheduler</code>, <code>WeeklyTaskScheduler</code>, and <code>MonthlyTaskScheduler</code> objects. Set the <code>interval</code> property to define how frequently a task should run. For example, setting the <code>interval</code> property of an hourly task to 4 causes the task to run every 4 hours.</p>
<code>HourlyTaskScheduler</code>	<p>Schedule a task to run once every hour (or every specified number of hours) at a specified time. Set the <code>interval</code> property to run the task after a specified number of hours.</p> <p>Example: Schedule a task to run every 4 hours at half-past the hour.</p>

TaskScheduler Subtype	Usage
	<pre>HourlyTaskScheduler hts = new HourlyTaskScheduler(); hts.setMinute(30); hts.setInterval(4);</pre>
DailyTaskScheduler	<p>Schedule a task to run daily or a specified number of days at a specified time (hour and minutes). Use in conjunction with the interval property to run the task after a specified number of days.</p> <p>Example: Schedule a task to run daily at 9:30 am (EST).</p> <pre>DailyTaskScheduler dts = new DailyTaskScheduler(); dts.setMinute(30); dts.setHour(14);</pre>
WeeklyTaskScheduler	<p>Schedule a task to run every week (or every specified number of weeks) on a specified day (or days) at a specific time. The hours and minutes are set as UTC values. At least one of the boolean values must be set to <code>true</code>. You can also set the interval property to run the task after a specified number of weeks.</p> <p>Example: Schedule a task to run every Tuesday and Sunday at 30 minutes past midnight.</p> <pre>WeeklyTaskScheduler wts = new WeeklyTaskScheduler(); wts.setMonday(true); wts.setTuesday(true); ... wts.setSaturday(false); wts.setSunday(true); dts.setMinute(30); dts.setHour(4);</pre>
MonthlyByDayTaskScheduler	<p>Schedule a task to run every month (or every specified number of months) on a specified day at a specified time (hour and minutes). You can also set the interval property to run the task after a specified number of months.</p> <p>Example: Schedule a task to run every 3 months (on the last day of the month) at 12:30 p.m.</p> <pre>MonthlyByDayTaskScheduler mbdts = new MonthlyByDayTaskScheduler(); mbdts.setDay(31); mbdts.setInterval(3); mbdts.setMinute(30); mbdts.setHour(14);</pre>

TaskScheduler Subtype	Usage
MonthlyByWeekdayTaskScheduler	<p>Schedule a task to run every month (or every specified number of months) on a specified week, weekday, and time (hour: minutes). You can also set the interval property to run the task after a specified number of months.</p> <p>Example: Schedule a task to run on the last Wednesday of each month at 12:30 a.m.</p> <pre>MonthlyByWeekdayTaskScheduler mbwts = new MonthlyByWeekdayTaskScheduler(); mbwts.setOffset(WeekOfMonth.last); mbwts.setWeekday(DayOfWeek.wednesday); mbwts.setHour(4); mbwts.setMinute(30);</pre>

The `hour` and `minute` properties of all objects that extend the `RecurrentTaskSchedule` data object are specified in Coordinated Universal Time (UTC) values rather than the local time of the server. When you define the schedule, convert your local time to a UTC value.

The following code fragment defines a `ScheduledTask` that powers on virtual machines daily at 4:15 a.m., if the server local time is in the Pacific Standard Time (PST) time zone. For a server in the Eastern European Summer Time (EEST) zone, the setting is read by the system as 3:15 pm.

Example: Scheduled Task for Powering-on Virtual Machines

```
...
// Set the schedule using the DailyTaskScheduler subtype.
DailyTaskScheduler dTScheduler = new DailyTaskScheduler();
dTScheduler.setHour(12);
dTScheduler.setMinute(15);
ScheduledTaskSpec tSpec = new ScheduledTaskSpec();
tSpec.setDescription("Start virtual machine as per schedule.");
tSpec.setEnabled(Boolean.TRUE);
tSpec.setName("Power On Virtual Machine");
tSpec.setAction(ma);
tSpec.setScheduler(dTScheduler);
tSpec.setNotification("admin@vmware.com");
my_conn.createScheduledTask(_sic.getScheduledTaskManager(), vmRef, tSpec);
...
```

Cancelling a Scheduled Task

You can cancel a scheduled task in several ways.

- To cancel the current run of a scheduled task, call `ScheduledTask.RemoveScheduledTask`. This method does not cancel subsequent runs of the `ScheduledTask`.
- To cancel an upcoming run of a `ScheduledTask`, call `ScheduledTask.ReconfigureScheduledTask` with a new `ScheduledTaskSpec` data object containing the new specifications for the schedule.

- To cancel a `ScheduledTask` that spawns a second task, create a `PropertyCollector` to obtain the reference to the `Tasks` and call its `CancelTask` method. The task must be cancellable.

Using a TaskHistoryCollector

A `TaskHistoryCollector` lets you gather information about tasks. You create a `TaskHistoryCollector` using the `TaskManager.CreateCollectorForTasks` method.

To create a `TaskHistoryCollector`

- 1 Identify the type of `Task` objects that you want to collect, and create an instance of a `TaskFilterSpec` data object that specifies your filter criteria.

The `TaskFilterSpec` includes an `taskId` property, which you use to limit the set of collected task objects to specific types. You can also provide a time range in the `TaskFilterSpec` by defining an `TaskFilterSpecByTime` data object for its `time` property. See the *vSphere API Reference*.

- 2 Obtain the managed object reference to the `TaskManager` on your server instance.
- 3 Submit the `filter` and the reference to the server in the `CreateTaskHistoryCollector` method. The server returns a reference to a `TaskHistoryCollector` object.

After a `HistoryCollector` has been created, the server appends new objects that meet the filter criteria to the collection as they occur. The system appends the new object to the collection by placing it in the first position of the `latestPage` and removes the oldest object from the collection. The `latestPage` property of the `TaskHistoryCollector` object has a property that consists of the 1000 most recent objects in the collection. Use a `PropertyCollector` to obtain the items from the `latestPage` property.

A `HistoryCollector` exists only for the duration of the session that instantiated it. Call the `HistoryCollector.DestroyCollector` method to delete the collector before the session ends.

Creating a TaskHistoryCollector Filter

When you create a `TaskHistoryCollector`, you can define filters. For example, rather than returning all `Task` objects associated with virtual machines, you might create a filter to collect only `Task` objects associated with virtual machines that were started by the backup-administrator between 2:00 and 4:00 a.m. on a specific date.

The `TaskFilterSpec` object allows you to specify the collection criteria. Most of the properties are optional and can be submitted as `null` values. The `TaskFilterSpec` lets you collect tasks based on user name, entity type, time, and state of the `Task`.

Managing the HistoryCollector

The `HistoryCollector` managed object provides operations for managing the life-cycle and scrollable view of a collection.

- `DestroyCollector` – A `HistoryCollector` exists only for the current session. Invoke the `DestroyCollector` operation to explicitly destroy the collector before the session ends.
- `ResetCollector` – Adjusts the starting position for the subset of objects from the collector to the object immediately preceding the current `latestPage`.
- `RewindCollector` – Positions the `latestPage` to the oldest item in the array. When a `HistoryCollector` is created, this is the default location.
- `SetCollectorPageSize` – Accepts an integer parameter to set the size of the `latestPage` property of a `HistoryCollector`. The default size of a `HistoryCollector` is an array that consists of at most 1000 objects of the appropriate type (`Task`, `Event`). The array is sorted by creation date and time of the objects.

Sample Code Reference

The following table lists the sample applications included with the vSphere SDK that demonstrate how to use some of the managed objects discussed in this chapter.

Java
(SDK\vsphere- ws\java\JAXWS\samples\com\vmware\host)
AddVirtualNic.java
AddVirtualSwitch.java
AddVirtualSwitchPortGroup.java
RemoveVirtualNic.java
RemoveVirtualSwitch.java
RemoveVirtualSwitchPortGroup.java

Events are sent by vSphere to convey information about things that happen in the system. You can monitor events directly or use an `EventHistoryCollector` to retrieve events from a certain period.

Alarms are sent by vSphere to alert users to problems. You can also create your own alarm to monitor the system and set up follow-up actions. Alarm setup includes specifying the trigger condition and defining the action that should result.

This chapter includes the following topics:

- [Event and Alarm Management Objects](#)
- [Understanding Events](#)
- [Using an EventHistoryCollector](#)
- [Using Alarms](#)
- [Defining Alarms Using the AlarmSpec Data Object](#)
- [Sample Code Reference](#)

Event and Alarm Management Objects

`EventManager` is the service interface for working with the event infrastructure.

`Event` subtypes define the events that the system generates. See [Event Data Objects](#) and [Creating Custom Events](#).

`EventHistoryCollector` allows you to monitor events. You can create a filter to limit the number of events your code retrieves. You can monitor both system events and your own events. See [Using an EventHistoryCollector](#).

The `AlarmManager` is the service interface for creating, setting, and managing alarms. You create an alarm, specifying trigger conditions and the action to take. When the conditions defined for the `Alarm` occur on the system, the `Action` specified for the alarm starts. The alarm also generates an `Event` that you can retrieve with an `EventHistoryCollector`.

Understanding Events

An `Event` is a data object type that contains information about state changes of managed entities and other objects on the server. Events include user actions and system actions that occur on datacenters, datastores, clusters, hosts, resource pools, virtual machines, networks, and distributed virtual switches. For example, these common system activities generate one or more `Event` data objects:

- Powering a virtual machine on or off
- Creating a virtual machine
- Installing VMware Tools on the guest OS of a virtual machine
- Reconfiguring a compute resource
- Adding a newly configured ESXi system to a vCenter Server system

In the vSphere Client, information from `Event` objects generated on a standalone ESXi hosts displays in the Events tab. For managed hosts, information from `Event` objects displays in the Tasks & Events tab.

Persistence of `Event` objects depends on the system setup.

- Standalone ESXi hosts – `Event` objects are not persistent. Events are retained only for as long as the host system's local memory can contain them. Rebooting a standalone ESXi host or powering off a virtual machine removes `Event` objects from local memory.

A standalone ESXi host might keep about 15 minutes worth of `Event` data, but this can vary depending on the processing load of the host, the number of virtual machines, and other factors.

- Managed ESXi systems. `Event` objects are persistent. Managed ESXi systems send `Event` data to the vCenter Server system that manages them, and the vCenter Server system stores the information its database.

You can use the event sample applications included in the SDK package with either managed or standalone ESXi systems and with vCenter Server systems.

Using an `EventHistoryCollector`, you can obtain information about these objects as they are being collected on a specific ESXi system, or from a specific historical period from the database. See [Using an EventHistoryCollector](#).

Managing Events with EventManager

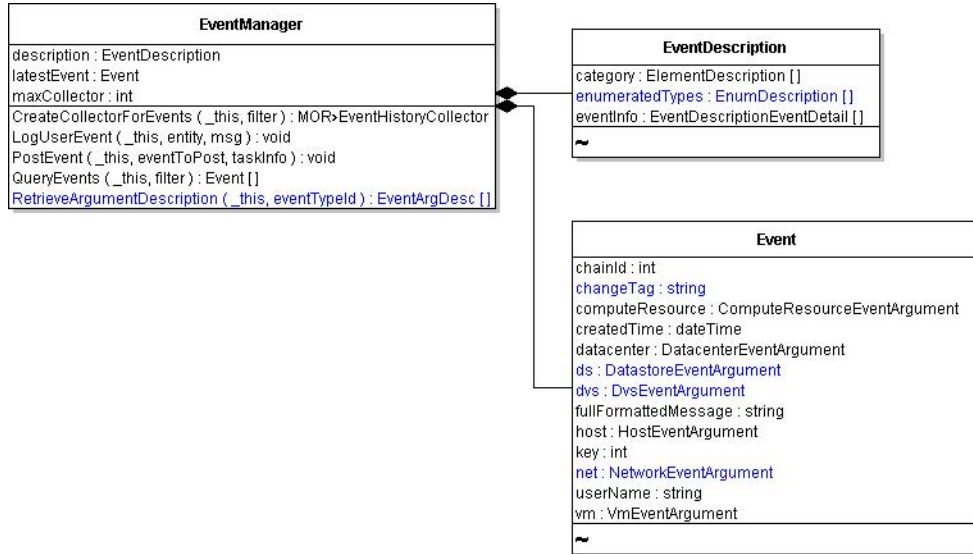
`EventManager` is the service interface for working with the event infrastructure.

[Figure 18-1. EventManager Managed Object and Associated Objects](#) shows `EventManager` and related objects. An `EventManager` has these properties:

- A `description` property, defined as an instance of an `EventDescription` data object, which contains an event category and other information.

- A `latestEvent` property that contains the most recent `Event` data object in memory.
- A `maxCollector` property that specifies the number of `EventHistoryCollector` objects per client session that can be created. This value is set by the vCenter Server system.

Figure 18-1. EventManager Managed Object and Associated Objects



Event Data Objects

`Event` subtypes define the events that the system generates.

Figure 18-2. [Event Data Object and Sample Subtypes](#) shows only a few of the subtypes that extend the `Event` data object. For example, `TaskEvent` inherits all `Event` properties and includes an `info` property that is an instance of a `TaskInfo` object (see [Monitoring TaskInfo Properties](#)).

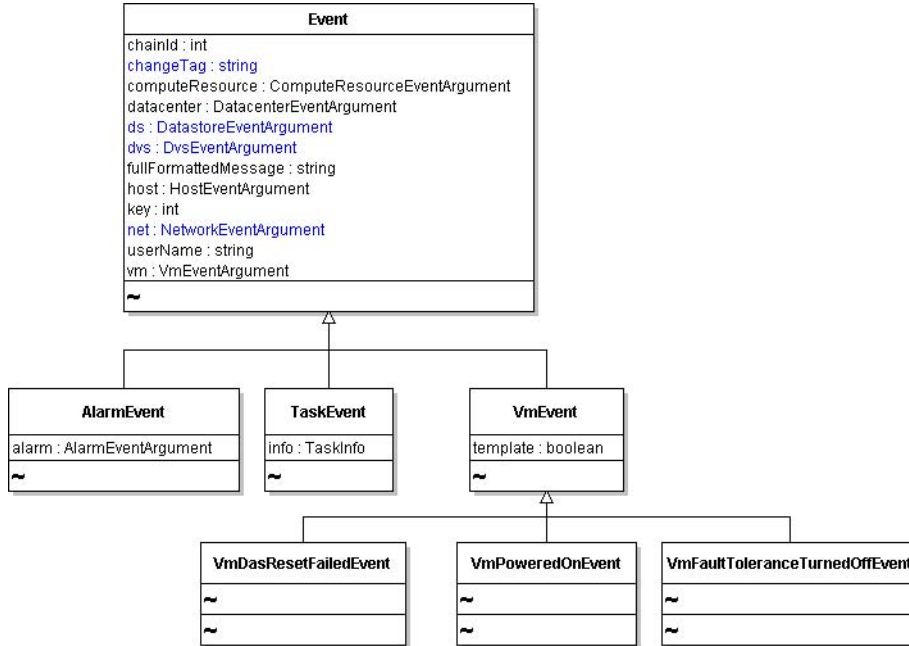
The following event objects are commonly generated by a console-style client application:

```

com.vmware.vim.VmPoweredOnEvent
com.vmware.vim.VmStartingEvent
com.vmware.vim.VmReconfiguredEvent
com.vmware.vim.VmCreatedEvent
com.vmware.vim.VmBeingCreatedEvent

```

Figure 18-2. Event Data Object and Sample Subtypes



Formatting Event Message Content

When displayed at the console, `Event` data objects are not formatted and do not provide context information. You can format an `Event` message using the predefined string in the `Event.fullFormattedMessage` property.

You can also format an `Event` message based on contextual information. At runtime, the `Event` data object is populated with values that contain information associated with the source of an event, for example, the `Event` data object's `computeResource`, `datacenter`, `ds`, `dvs`, `host`, `net`, and `vm` properties.

You can use the properties of an `Event` object with the information in the `EventDescriptionEventDetail` in `EventManager.description.eventInfo` to format event messages.

Creating Custom Events

The Web Services API allows you to create custom `Event` objects that convey information specific to your application. There are two types of custom events, the user log event and the extended event.

Creating User Log Events

The `EventManager.LogUserEvent` method allows you to create user log `Event` objects. You can associate your custom `Event` with any managed entity. User log events are useful for marking actions or status associated with the objects your application deals with.

The following steps show how to create a user log `Event`.

Procedure

- 1 Obtain the managed object reference to the `EventManager`.

```
..
ManagedObjectReference _svcRef = new ManagedObjectReference();
ServiceContent _sic = my_conn.retrieveServiceContent(_svcRef);
ManagedObjectReference eMgrRef = _sic.getEventManager();
...
```

- 2 Obtain the managed object reference to the entity with which you are associating the `Event`.

For example, suppose you have a reference to a virtual machine (`myVMRef`) and you want to log a message to record the fact that a virus check completed. You want to use `myVMRef` as a parameter to the `LogUserEvent` method in the next step.

- 3 Call the `LogUserEvent` method, passing in the `EventManager` and the `Event` reference and a string consisting of the Event message for the `msg` parameter of the operation.

```
LogUserEvent(eMgrRef, myVMRef, "Completed virus check at 1:05 AM on Sunday December 21.");
```

Results

User-defined Event objects display in the vSphere Client among the other events on the system, with the prefix `User logged event:` followed by the text submitted in your `msg` parameter. In other client applications, such as in the console-based Event sample applications, custom events display as `com.vmware.vim.GeneralUserEvent` objects.

Creating Extended Events

The `EventManager.EventEx` method allows you to create an event that contains an arbitrary dictionary of key-value pairs. This kind of custom event allows greater flexibility to store application data that is not associated with a managed entity, and is not limited to a single string value. The custom event structure also contains more sophisticated metadata than user log events.

The following steps show pseudocode examples of the operations you need to do in your client code.

Procedure

- 1 Obtain the managed object reference to the `EventManager`.

```
si = connection.retrieveServiceContent(svc_ref);
em = si.eventManager;
```

- 2 Choose a severity for the custom event.

```
severity = EventEventSeverity.warning;
```

3 Create a local copy of an extended event with metadata.

```
e = vim.event.EventEx(severity,
                      eventType="com.example.events.Total_System_Backup",
                      createTime = si.CurrentTime(),
                      chainId=0,
                      key=0,
                      userName=local_account_name);
```

4 Create a set of key-value pairs to store the information you want to associate with the event.

```
arg_list = {};
arg1 = vmopl.KeyAnyValue(key="reason", value="Upcoming governance audit");
arg2 = vmopl.KeyAnyValue(key="datacenter", value="Washington South");
arg3 = vmopl.KeyAnyValue(key="organization", value="IT Cloud Services");
arg_list.append(arg1, arg2, arg3);
```

5 Add the set of key-value pairs to the event object.

```
e.arguments = arg_list;
```

6 Use the Event Manager to post the event to the server.

```
em.PostEvent(e);
```

Using an EventHistoryCollector

An `EventHistoryCollector` lets you gather information about events that the server has generated. You create an `EventHistoryCollector` using the `EventManager.CreateCollectorForEvents` method.

The following steps show how to create an `EventHistoryCollector`.

Procedure

- 1 Identify the type of `Event` objects that you want to collect, and create an instance of an `EventFilterSpec` data object that specifies your filter criteria. See [Creating an EventHistoryCollector Filter](#).

The `EventFilterSpec` includes an `eventType` property, which you use to limit the set of collected event objects to specific types. You can also provide a time range in the `EventFilterSpec`, by defining an `EventFilterSpecByTime` data object for its `time` property. See the *vSphere API Reference* for details.

- 2 Obtain the managed object reference to the `EventManager` on your server instance.
- 3 Submit the `filter` and the reference to the server in the `CreateEventHistoryCollector` operation. The server returns a reference to an `EventHistoryCollector` object.

Results

After you have created the `HistoryCollector`, the server appends new objects that meet the filter criteria to the collection as they occur. The system appends the new object to the collection by placing it in the first position of the `latestPage` and it removes the oldest object from the collection. The `latestPage` property of the `EventHistoryCollector` object has a property that consists of the 1000 most recent objects in the collection. Use a `PropertyCollector` to obtain the items from the `latestPage` property.

A `HistoryCollector` exists only for the duration of the session that instantiated it. You invoke the `DestroyCollector` operation to explicitly eliminate the collector before the session ends.

Creating an EventHistoryCollector Filter

When you create an `EventHistoryCollector`, you can define filters. For example, rather than returning all `Event` objects associated with virtual machines, you might create a filter to collect only those `Event` objects associated with virtual machines that were started by the backup-administrator between 2:00 and 4:00 a.m. on a specific date.

The `EventFilterSpec` object allows you to specify the collection criteria. Most of the properties are optional and can be submitted as `null` values. The `EventFilterSpec` lets you collect events based on user name, entity type, time, and state of the `Event`.

Managing the HistoryCollector

The `HistoryCollector` managed object provides operations for managing the life-cycle and scrollable view of a collection.

- `DestroyCollector` – A `HistoryCollector` exists only for the current session. Invoke the `DestroyCollector` operation to explicitly destroy the collector before the session ends.
- `ResetCollector` – Adjusts the starting position for the subset of objects from the collector to the object immediately preceding the current `latestPage`.
- `RewindCollector` – Positions the `latestPage` to the oldest item in the array. When a `HistoryCollector` is created, this is the default location.
- `SetCollectorPageSize` – Accepts an integer parameter to set the size of the `latestPage` property of a `HistoryCollector`. The default size of a `HistoryCollector` is an array that consists of at most 1000 objects of the appropriate type (`Task`, `Event`). The array is sorted by creation date and time of the objects.

Using Alarms

The vSphere alarm infrastructure supports automating actions and sending different types of notification in response to certain server conditions. Many `Alarms` exist by default on vCenter Server systems. You can also create alarms yourself. For example, an `Alarm` can send an alert

email message when CPU usage on a specific virtual machine exceeds 99% for more than 30 minutes.

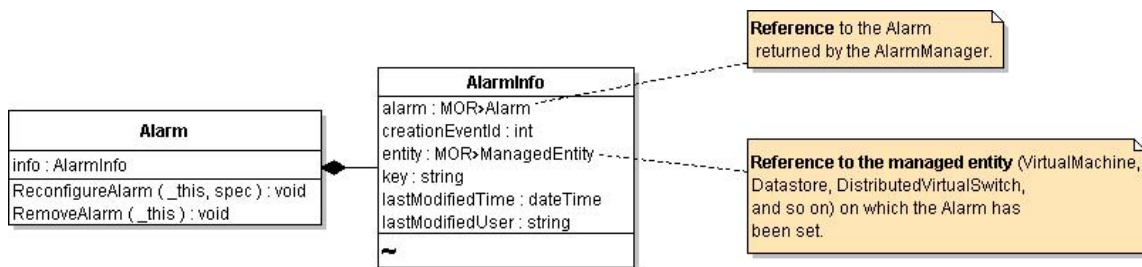
The alarm infrastructure integrates with other server components, such as events and performance counters.

The `AlarmManager` is the service interface for creating, setting, and managing alarms. You create an alarm, specifying trigger conditions and the action to take. When the conditions defined for the `Alarm` occur on the system, the `Action` specified for the alarm starts. The alarm also generates an `Event` that is posted to the `Event` history database. In addition, the action initiated by the `Alarm` might also post a second `Event` to the database, depending on the `Action` type.

Obtaining a List of Alarms

Use the `AlarmManager.GetAlarm` method to obtain an array of references to all `Alarm` managed objects defined for a specific managed entity. When you call the method, you can pass in an optional reference to a managed entity. Without a reference to a managed entity, the `GetAlarm` operation returns all `Alarm` objects for all entities that are visible to the principal associated with the session invoking the operation.

Figure 18-3. Alarm Managed Object



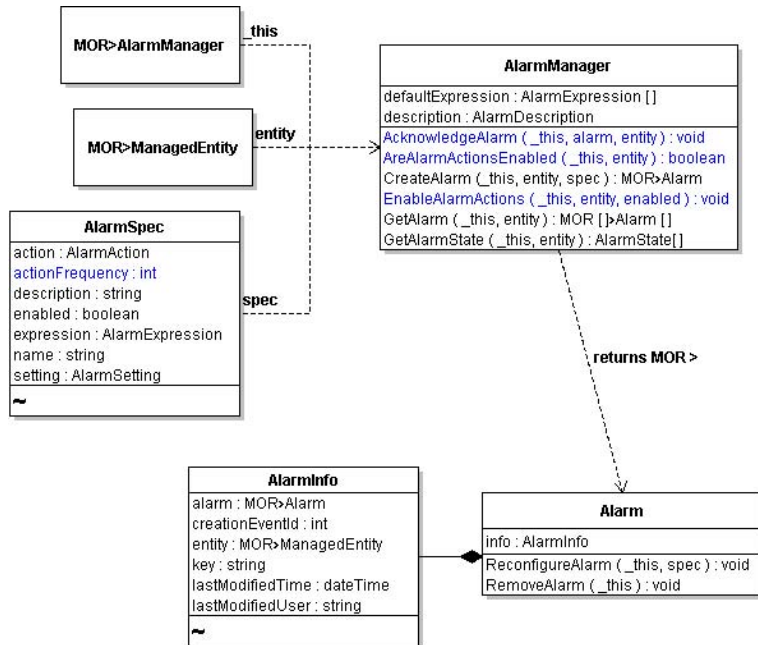
The `Alarm.info` property is an `AlarmInfo` data object. You can obtain information about active Alarms by collecting the properties of the `AlarmInfo` data object.

Creating an Alarm

You create an alarm with the `AlarmManager.CreateAlarm` method. In the simplest case, you specify the trigger condition in the `AlarmSpec.expression` property and the action to perform in the `AlarmSpec.action` property. When the expression evaluates to `true`, the alarm performs the action.

Figure 18-4. [CreateAlarm Method Inputs and Outputs](#) shows the `CreateAlarm` method.

Figure 18-4. CreateAlarm Method Inputs and Outputs



The following steps show how to create an alarm.

Procedure

- 1 Obtain a managed object reference to the `AlarmManager` associated with the vCenter Server.
- 2 Obtain a managed object reference of the entity on which you want to set the `Alarm`.
- 3 Create an `AlarmSpec` data object and specify the alarm details in its properties. See [Defining Alarms Using the AlarmSpec Data Object](#).
- 4 Call `AlarmManager.CreateAlarm`, passing in the references and the `AlarmSpec` data object. The system returns a managed object reference to the `Alarm` (see [Figure 18-4. CreateAlarm Method Inputs and Outputs](#)).

Results

The state of an alarm is contained in an `AlarmState` data object.

Defining Alarms Using the AlarmSpec Data Object

The `AlarmSpec` data object has properties for all aspects of an `Alarm`, including its expression and the action to take when the expression evaluates to true. The following properties define the alarm; see the *API Reference* for a complete list.

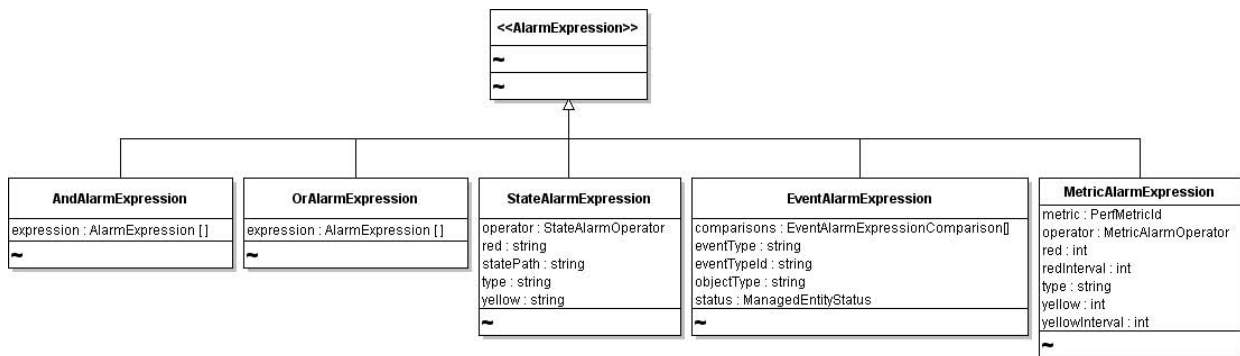
- `action` – Action to initiate when the `Alarm` becomes active. Specify one of the `Action` subtypes. See [Specifying Alarm Actions](#).

- `actionFrequency` – Number of seconds that the `Alarm` remains in the state required to initiate the specified action.
- `expression` – One or more `AlarmExpression` data objects combined in a way that evaluates to a true-false expression. See [Specifying Alarm Trigger Conditions with AlarmExpression](#).
- `setting` – Tolerance and frequency limits for the `Alarm` defined in the `AlarmSetting` data object. `AlarmSetting` contains two integer properties:
 - `reportingFrequency`, which specifies the number of seconds between activation of an alarm. Use 0 to specify that the alarm can activate as frequently as required.
 - `toleranceRange`, which specifies the acceptable range (measured in hundredth percentage) above and below the specified value defined in a `MetricAlarmExpression`.

Specifying Alarm Trigger Conditions with AlarmExpression

You use the `AlarmExpression` data object to specify the conditions under which you want the `Alarm` to become active. The `AlarmExpression` data object is an abstract type with several subtypes, which allow you to specify thresholds on objects, state of objects, or specify specific events to monitor.

Figure 18-5. AlarmExpression and Its Subtypes



AlarmExpression Types

By using the appropriate type of `AlarmExpression`, you can set alarms for different conditions, states, or events.

AlarmExpression	Description	Example
<code>StateAlarmExpression</code>	Specifies thresholds that trigger the alarm.	Triggered by a power state change of a virtual machine or state change of a distributed virtual switch.
<code>MetricAlarmExpression</code>	Specifies levels at which the alarm changes state. See Using MetricAlarmExpression .	Triggered when resource utilization metrics exceed a specified limit.

AlarmExpression	Description	Example
EventAlarmExpression	Specifies a type of event as the basis for the alarm.	Triggered by power on or power off events of primary or secondary virtual machines in a fault-tolerant cluster.
EventAlarmComparison	Specifies the property of the Event that should trigger the alarm and the operator to use as the basis for comparison.	
AndAlarmExpression OrAlarmExpression	Combines one or more instances of the AndAlarmExpression and the OrAlarmExpression data objects into an expression that evaluates to true or false.	

Using MetricAlarmExpression

The `MetricAlarmExpression` data object lets you set an alarm to monitor performance metrics. The vSphere Client uses the data object to indicate when hosts or clusters do not have sufficient resources in a DAS or DRS cluster environment. See the *Resource Management Guide*.

You set the `metric` property to the `PerfMetricId` of a performance metric that you want to monitor on the system. Set the `red` or `yellow` properties to identify the level at which the metric value moves from green, to yellow, to red. You must define `red`, `yellow`, or both properties. Use each of these properties with the `isAbove` or `isBelow` `MetricAlarmOperator` enumerations to complete the definition of the threshold.

In conjunction with `red` and `yellow` properties, you can use the `redInterval` or `yellowInterval` properties. These properties enable you to set the number of seconds that the performance metric must be in `red` or `yellow` state before the expression becomes `true` and triggers the defined action.

Specifying Alarm Actions

You specify the actions that the system should take by setting the `action` property of the `AlarmSpec` data object to the `AlarmAction` data object defined for the purpose.

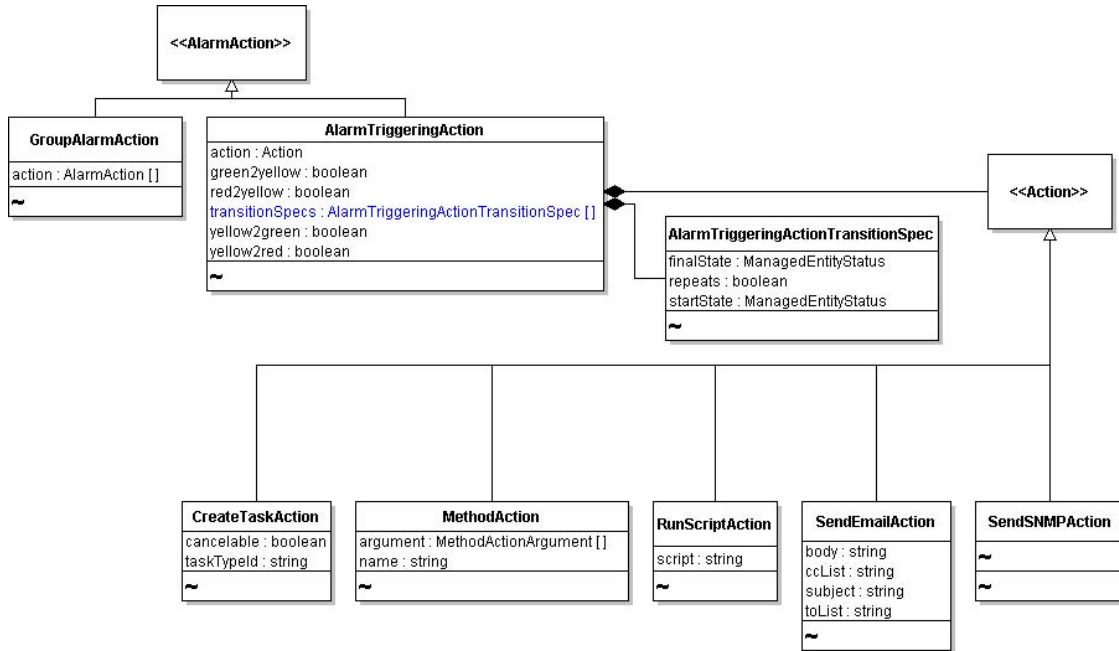
The `AlarmAction` data object is an abstract type that has two descendent objects.

- The `AlarmTriggeringAction` data object has an `action` property and a `transitionSpecs` property. `AlarmTriggeringActionTransitionSpec` allows you to define a starting state and a final state for the Alarm. You can limit the number of Alarm objects actually triggered to a single Alarm by specifying `false` for the `repeats` property of the `AlarmTriggeringActionTransitionSpec`.
- The `GroupAlarmAction` data object is an array version of the `AlarmAction` base type. You can create a single `AlarmAction` instance or an array of `AlarmAction` instances to take effect when the conditions specified for your alarm are met on the system.

The system can respond to an alarm in several ways:

- Invoking an operation. To invoke an operation, create a `MethodAction` data object.
- Running a Script. To run a script, create an instance of the `RunScriptAction` data object that specifies the fully qualified path to the shell script on the vCenter Server.
- Send an email message. To send an email message to a system administrator, use the `SendEmailAction` data object.

Figure 18-6. AlarmAction and Related Objects



For example, you can use the `MethodAction` data object type to invoke an operation on the server. The `MethodAction` data object contains the following properties:

- `name`—Name of the operation that you want to invoke at the scheduled time.
- `argument`—Specifies required parameters, if any, as an array of `MethodArgumentAction` data objects.

Depending on the entity associated with the alarm, the `MethodAction.argument` property might not be needed.

Deleting or Disabling an Alarm

An `Alarm` remains active until you delete it or disable it. To delete the alarm, obtain a managed object reference to the `Alarm` and invoke its `RemoveAlarm` operation.

To disable the `Alarm`, obtain managed object references to the `AlarmManager` and to the entity on which the `Alarm` is set. Call `AlarmManager.EnableAlarmActions` operation, passing the value `false` for the `enabled` parameter.

Sample Code Reference

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java
<code>\samples\alarms\MPowerStateAlarm.java</code>
<code>\samples\events\EventFormat.java</code>
<code>\samples\events\EventHistoryCollectorMonitor.java</code>
<code>\samples\events\VMEventHistoryCollectorMonitor.java</code>

VMware vSphere servers use performance counters to track resource use. At runtime, vSphere components generate performance data which the vSphere servers store in performance counters. You can use the `PerformanceManager` interface to retrieve the data.

This chapter includes the following topics:

- [vSphere Performance Data Collection](#)
- [PerformanceManager Objects and Methods](#)
- [Retrieving vSphere Performance Data](#)
- [Performance Counter Metadata](#)
- [Performance Intervals](#)
- [vSphere Performance and Data Storage](#)
- [Sample Code Reference](#)

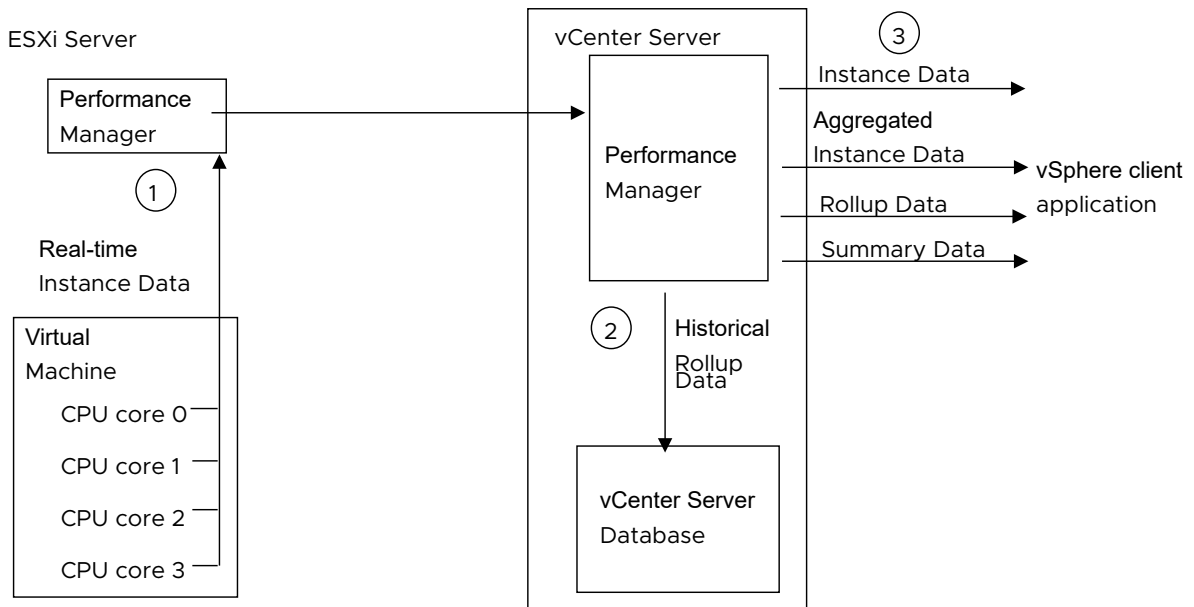
vSphere Performance Data Collection

In a vSphere environment, virtual and physical components generate performance data. To track the use of resources, ESXi Servers perform real-time data collection and vCenter Servers store the data in the vCenter database. vCenter Servers also store a historical rollup of the data according to defined performance intervals.

- Real-time data collection – An ESXi Server collects data for each performance counter every 20 seconds and maintains that data for one hour.
- Historical data rollup – A vCenter Server collects data from all of the hosts that the vCenter Server manages. The `PerformanceManager` defines performance intervals that specify time periods for performance data rollup, a methodology for combining data values. The server stores the rolled up performance counter data in the vCenter database.

The following figure represents vSphere performance data collection and retrieval.

Figure 19-1. vSphere Performance Data Collection and Retrieval



time instance data for one hour. For example, the figure shows collection of CPU statistics for four CPU cores.

- 2 The vCenter Server retrieves and stores data from the servers that it manages. The Server produces rollup data according to the settings of the historical intervals.
- 3 vSphere client applications can retrieve real-time instance data, aggregated instance data, historical rollup data, and summary data.

The following table defines terms that are used to describe vSphere performance management.

Term	Definition
performance providers	Performance providers include managed entities, such as hosts, virtual machines, compute resources, resource pools, datastores, and networks.
performance counter	Unit of statistical data collected on a vSphere server. For example, a vCenter server collects the average CPU utilization for hosts, virtual machines and clusters (the counter <code>cpu.usage.average</code>).
counter ID	System-generated identifier for a performance counter.
instance	An identifier derived from device configuration names. Examples of counter instances are the name of a virtual Ethernet adapter such as “ <code>vmnic0:</code> ”, or a number that identifies a CPU core, such as 0, 1, 2, or 3. Performance data is retrieved as specific instances of performance counters.
instance data	Performance data collected at 20-second intervals.
metric ID	<p>Combination of a counter ID and an instance. You use metric IDs – <code>PerfMetricId</code> objects – when you construct a performance query specification to identify the data to be collected.</p> <p>There are two system-defined instances that you can use to specify aggregate retrieval. See the description of aggregate performance data below.</p> <ul style="list-style-type: none"> ■ “*” – An asterisk directs the vSphere Server to return all instances plus rollup data. This is not supported for some disk-related counters. ■ “” – A string of length zero directs the vSphere Server to return only aggregated instance data or rollup type data. <p>The vSphere Server returns metric IDs embedded in the data objects that it returns as a response to performance queries.</p>
performance interval	<p>Data object (<code>PerfInterval</code>) which defines the time interval between collection events, the collection level, and the time period that the data will be stored on the Server.</p> <ul style="list-style-type: none"> ■ ESXi Servers define a built-in performance interval that specifies data collection every 20 seconds for each performance counter. ESXi Servers also define a single historical interval (<code>PerformanceManager.historicalInterval</code>) that defines aggregate performance data. This system-defined performance interval specifies aggregate data collection every 300 seconds for each counter. You cannot modify the performance intervals on an ESXi Server. ■ vCenter Servers define four performance intervals that determine how collected instance data is aggregated and stored. You can modify the system-defined intervals on a vCenter Server to a limited extent.

Term	Definition
collection level	Number between one and four that is assigned to a performance interval (<code>PerformanceManager.historicalInterval[].level</code>). The interval collection level corresponds to the level specified for individual performance counters (<code>PerfCounterInfo.level</code>). A vCenter Server uses a performance interval to perform performance data aggregation, using data for the counters with levels that match the performance interval collection level.
rollup type	Methodology for producing a single value from a set of statistical values (<code>PerformanceManager.perfCounter[].rollupType</code>). Examples of rollup types are average, latest, and summation.
aggregate performance data	A single value that represents a set of instance data values collected for a performance counter. The single value is derived using one of the rollup types.

PerformanceManager Objects and Methods

`PerformanceManager` provides methods for obtaining statistical data about various aspects of system performance, as generated and maintained by the performance providers. It also defines historical performance intervals and it identifies the set of performance counters that you can use to obtain performance data. The following table shows the `PerformanceManager` properties.

Property	Description
description	Composite object that includes information about the types of counters (<code>counterType</code>) and statistics (<code>statsType</code>) available on the system.
historicalInterval	<p>Array of system-defined performance intervals (<code>PerfInterval</code> data objects). Each object defines the interval between rollup events, the collection level, and the time period that the data is stored on the system.</p> <ul style="list-style-type: none"> ■ For an ESXi system, the array contains a single performance interval. You cannot modify the ESXi performance interval. ■ For vCenter Server systems, the <code>PerfInterval</code> objects control how ESXi performance data are rolled up and stored in the database. You can modify some of the <code>PerfInterval</code> properties on a vCenter Server.
perfCounter	Array of <code>PerfCounterInfo</code> data objects. The array identifies all of the performance counters known to the vCenter Server at the time a client accesses the array. The set of counters may change as ESXi hosts are added or removed from vCenter management. Each <code>PerfCounterInfo</code> object contains metadata associated with a performance counter.

The `PerformanceManager` methods allow you to retrieve performance statistics and to retrieve metadata that defines the statistics. The following table classifies the methods and describes their purposes.

Method Type	Method	Purpose
Performance data availability	QueryAvailablePerfMetric	Returns <code>PerfMetricId</code> objects which identify the counter data available on the specified entity. For example, a virtual machine provides the memory counter <code>granted</code> , which indicates the amount of physical memory that is mapped for the virtual machine. The <code>PerfMetricId</code> object for the <code>mem.granted.average</code> counter specifies the system-defined counter ID. Since this is a memory counter, the <code>PerfMetricId.instance</code> property is empty.
Performance data retrieval	QueryPerf	Returns statistics for a specific list of managed entities that provide performance data.
	QueryPerfComposite	Returns statistics for a host and its virtual machines. This method accepts the <code>refreshRate</code> for current statistics or the <code>intervalId</code> of one of the historical intervals as a parameter. Supported for the <code>HostSystem</code> managed entity only.
Performance counter metadata retrieval	QueryPerfCounter	Returns <code>PerfCounterInfo</code> data objects for the specified list of counter IDs.
	QueryPerfCounterByLevel	Returns <code>PerfCounterInfo</code> data objects for the specified collection level.
Performance provider information	QueryPerfProviderSummary	Returns the <code>PerfProviderSummary</code> data object for the specified managed object.
Collection parameters	ResetCounterLevelMapping	Restores a set of performance counters to their default collection levels.
	UpdateCounterLevelMapping	Changes the collection level for a set of performance counters.
	UpdatePerfInterval	Modifies the system-defined performance intervals.

Retrieving vSphere Performance Data

To retrieve collected data, your client application creates a query specification and passes the specification to a performance query method. The query specification is composed of one or more `PerfQuerySpec` objects. Each object identifies the following:

- Performance provider – managed entity for which the Server will return performance data (`PerfQuerySpec.entity`).
- Performance counters – `PerfMetricId` objects that identify performance counter instances (`PerfQuerySpec.metricId`).

- Performance interval – the sampling period that defines the data rollup (`PerfQuerySpec.intervalId`).
- Amount of data to be returned – start and end times (`PerfQuerySpec.startTime`, `PerfQuerySpec.endTime`) and maximum number of values (`PerfQuerySpec.maxSample`) to limit the amount of data to be returned.
- Output data format (`PerfQuerySpec.format`) – one of two kinds:
 - Normal output returned as values contained in data objects.
 - Formatted output returned as strings containing comma-separated values.

The combination of the `entity` and `metricID` properties determine the set of counters for which the server will return performance data. The combination of the `interval`, `startTime`, `endTime` properties produce instance, aggregated instance, rollup, or summarized data. The following table summarizes the different classifications of performance data that you can retrieve from a vCenter Server.

Performance Data	Description
Instance	ESXi Servers sample performance data every 20 seconds. 20-second interval data is called instance data or real-time data. To retrieve instance data, specify a value of 20 seconds for the <code>PerfQuerySpec.intervalId</code> property.
Aggregated Instance	<p>A vSphere client can retrieve aggregated instance data. To obtain aggregated instance data, specify the following <code>PerfQuerySpec</code> properties.</p> <ul style="list-style-type: none"> ■ <code>intervalId</code> – Specify 20 seconds to indicate instance data. ■ <code>metricId[].instance</code> – specify a zero-length string ("") for aggregated instance data.
Rollup	<p>The vCenter Server uses the historical intervals to rollup performance data from the servers that it manages. To retrieve historical performance data, specify the following <code>PerfQuerySpec</code> properties.</p> <ul style="list-style-type: none"> ■ <code>intervalId</code> – Specify a value that corresponds to one of the historical intervals (<code>PerformanceManager.historicalInterval[].samplingPeriod</code>). ■ <code>startTime/endTime</code> – If specified, use time values that are not within the last 30 minutes of the current time. If you do not specify a starting time, the Server will return values starting with the earliest data. If you do not specify an end time, the Server will return values that include the latest data.
Summary	When you call the <code>QueryPerf</code> method and specify a performance interval (<code>PerfQuerySpec.intervalId</code>) that does not match one of the historical intervals (<code>PerformanceManager.historicalInterval[].samplingPeriod</code>), the Server will attempt to summarize the stored data for the specified interval. In this case, the Server may return values that are different from the values that were stored for the historical intervals.

Performance Counter Example (QueryPerf)

The following code fragments are part of an example that uses the `PerformanceManager.QueryPerf` method to obtain performance statistics for a virtual machine.

The example code in this section does not include server connection code and it does not show the code for obtaining the managed object reference for the virtual machine. See [Chapter 3 Client Applications for the Web Services API](#) for an example of server connection code.

This example retrieves the following statistics:

- `disk.provisioned.LATEST` – virtual machine storage capacity.
- `mem.granted.AVERAGE` – amount of physical memory mapped for the virtual machine.
- `power.power.AVERAGE` – current power usage.

The example creates a query specification (`PerfQuerySpec`) to identify the data to be retrieved, calls the `QueryPerf` method, and prints out the retrieved performance data and corresponding performance counter metadata. The following sections describe the basic steps involved in retrieving performance statistics.

- Map the performance counters – [Mapping Performance Counters \(Counter Ids and Metadata\)](#).
- Create a performance query specification and call the `QueryPerf` method – [Retrieving Statistics](#).
- Process the returned data – [Handling Returned Performance Data](#).

Mapping Performance Counters (Counter Ids and Metadata)

Performance counters are represented by string names, for example `disk.provisioned.LATEST` or `mem.granted.AVERAGE`. A vSphere server tracks performance counters by using system-generated counter IDs. When you create a performance query, you use counter IDs to specify the statistics to be retrieved, so it is useful to map the names to IDs.

The example must specify counter IDs in the calls to `QueryPerf`, and it will use performance counter metadata when it prints information about the returned data. To obtain performance counter IDs and the corresponding performance counter metadata, the example creates two hash maps. This example maps the entire set of performance counters to support retrieval of any counter.

HashMap Declarations

The following code fragment declares two hash maps.

- `countersIdMap` – Uses full counter names to index performance counter IDs. A full counter name is the combination of counter group, name, and rollup type. The example uses this map to obtain counter IDs when it builds the performance query specification.
- `countersInfoMap` – Uses performance counter IDs to index `PerformanceCounterInfo` data objects. The example uses this map to obtain metadata when it prints the returned performance data.

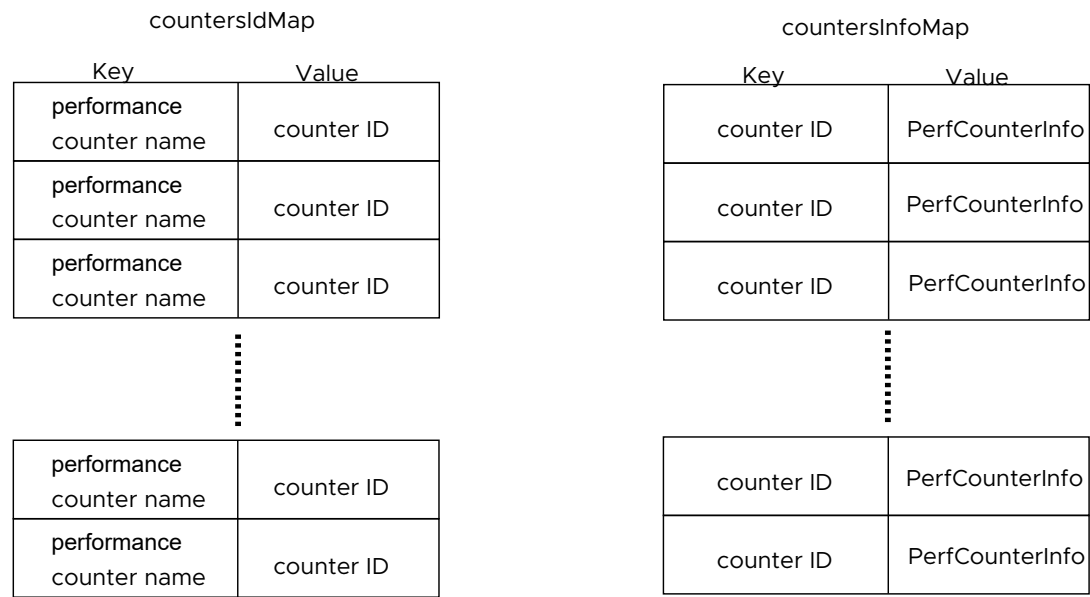
```
/*
 * Map of counter IDs indexed by counter name.
 * The full counter name is the hash key - group.name.ROLLUP-TYPE.
 */
```

```
private static HashMap<String, Integer> countersIdMap = new HashMap<String, Integer>();

/*
 * Map of performance counter data (PerfCounterInfo) indexed by counter ID
 * (PerfCounterInfo.key property).
 */
private static HashMap<Integer, PerfCounterInfo> countersInfoMap =
    new HashMap<Integer, PerfCounterInfo>();
```

The following figure shows a representation of the hash maps.

Figure 19-2. Performance Counter Hash Maps



Creating the Map

The example uses the Property Collector to retrieve the array of performance counters (PerfCounterInfo) known to the vCenter Server (PerformanceManager.perfCounter[]). It then uses the data to create the maps. The code fragment uses the variable `apiMethods`, which is a `VimPortType` object that provides access to the vSphere API methods. For information about the `VimPortType` object, see [Overview of a Java Sample Application for the Web Services SDK](#).

The following code fragment performs these steps:

- 1 Create an `ObjectSpec` to define the property collector context. This example specifies the Performance Manager.
- 2 Create a `PropertySpec` to identify the property to be retrieved. This example retrieves the `perfCounter` property, which is an array of `PerfCounterInfo` objects.
- 3 Create a `PropertyFilterSpec` for the call to the `PropertyCollector`. The `PropertyFilterSpec` creates the association between the `ObjectSpec` and `PropertySpec` for the operation.
- 4 Call the `PropertyCollector.RetrievePropertiesEx` method. This method blocks until the server returns the requested property data.
- 5 Cast the returned `xsd:anyType` value into the array of `PerfCounterInfo` objects.
- 6 Cycle through the returned array and load the maps. The counter-name to counter-ID map uses a fully qualified counter name. The qualified name is a path consisting of counter group, counter name, and rollup type – *group.counter.ROLLUP-TYPE*. The rollup type must be coded in uppercase letters. Examples of qualified names are `disk.provisioned.LATEST` and `mem.granted.AVERAGE`.

```
/*
 * Create an object spec to define the context to retrieve the PerformanceManager property.
 */
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(performanceMgrRef);

/*
 * Specify the property for retrieval
 * (PerformanceManager.perfCounter is the list of counters the vCenter Server is aware of.)
 */
PropertySpec pSpec = new PropertySpec();
pSpec.setType("PerformanceManager");
pSpec.getPathSet().add("perfCounter");

/*
 * Create a PropertyFilterSpec and add the object and property specs to it.
 */
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);
```

```

/*
 * Create a list for the filter and add the spec to it.
 */
List<PropertyFilterSpec> fSpecList = new ArrayList<PropertyFilterSpec>();
fSpecList.add(fSpec);

/*
 * Get the performance counters from the server.
 */
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = apiMethods.retrievePropertiesEx(pCollectorRef, fSpecList, ro);

/*
 * Turn the retrieved results into an array of PerfCounterInfo.
 */
List<PerfCounterInfo> perfCounters = new ArrayList<PerfCounterInfo>();
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                /*

/*
 * Cycle through the PerfCounterInfo objects and load the maps.
 */
                * DynamicProperty.val is an xsd:anyType value to be cast
                * to an ArrayOfPerfCounterInfo and assigned to a List<PerfCounterInfo>.
                */
                perfCounters = ((ArrayOfPerfCounterInfo) dp.getVal()).getPerfCounterInfo();
            }
        }
    }
}
for (PerfCounterInfo perfCounter : perfCounters) {
    Integer counterId = new Integer(perfCounter.getKey());

    /*
     * This map uses the counter ID to index performance counter metadata.
     */
    countersInfoMap.put(counterId, perfCounter);

    /*
     * Obtain the name components and construct the full counter name,
     * for example - power.power.AVERAGE.
     * This map uses the full counter name to index counter IDs.
     */
    String counterGroup = perfCounter.getGroupInfo().getKey();
    String counterName = perfCounter.getNameInfo().getKey();
    String counterRollupType = perfCounter.getRollupType().toString();
    String fullCounterName = counterGroup + "." + counterName + "." + counterRollupType;

    /*
     * Store the counter ID in a map indexed by the full counter name.
     */

```



```

        countersIdMap.put(fullCounterName, counterId);
    }

```

Retrieving Statistics

The following code fragment calls the `QueryPerf` method to retrieve statistics. It performs these tasks:

Procedure

- 1 Create a list of qualified performance counter names for retrieval. The name is a path consisting of *group-name.counter-name.ROLLUP-TYPE*, for example `mem.granted.AVERAGE`. The rollup type must be coded in uppercase letters to match the character case of the rollup type in the performance counter metadata (`PerfCounterInfo.rollupType`). See the *vSphere API Reference* for tables of available counters. The *vSphere API Reference* page for the `PerformanceManager` managed object contains links to the tables.
- 2 Create a list of `PerfMetricId` objects, one for each counter to be retrieved. The metric ID is a combination of the counter ID and the instance. To fill in the `PerfMetricId` properties, the example does the following:
 - Use the `countersIdMap` to translate a full counter name into a counter ID.
 - Specify an asterisk (*) for the `PerfMetricId.instance` property. The asterisk is the system-defined instance specification for combined instance and rollup retrieval.
- 3 Build a query specification for the method call. This query specifies the following:
 - Virtual machine for which performance data is being retrieved (`entityMor`);
 - Interval ID of 300 to collect 5-minute rollup data.
 - Comma-separated value (CSV) format for the retrieved data.
- 4 Call the `QueryPerf` method.

Example

```

/*
 * Use <group>.<name>.<ROLLUP-TYPE> path specification to identify counters.
 */
String[] counterNames = new String[] { "disk.provisioned.LATEST",
                                         "mem.granted.AVERAGE",
                                         "power.power.AVERAGE" };

/*
 * Create the list of PerfMetricIds, one for each counter.
 */
List<PerfMetricId> perfMetricIds = new ArrayList<PerfMetricId>();
for(int i = 0; i < counterNames.length; i++) {
    /*
     * Create the PerfMetricId object for the counterName.
     * Use an asterisk to select all metrics associated with counterId (instances and rollup).
    */
}

```

```

    */
    PerfMetricId metricId = new PerfMetricId();
    /* Get the ID for this counter. */
    metricId.setCounterId(countersIdMap.get(counterNames[i]));
    metricId.setInstance("");
    perfMetricIds.add(metricId);
}

/*
 * Create the query specification for queryPerf().
 * Specify 5 minute rollup interval and CSV output format.
 */
int intervalId = 300;
PerfQuerySpec querySpecification = new PerfQuerySpec();
querySpecification.setEntity(
querySpecification.setIntervalId(intervalId);
querySpecification.setFormat("csv");
querySpecification.getMetricId().addAll(perfMetricIds);

List<PerfQuerySpec> pqsList = new ArrayList<PerfQuerySpec>();
pqsList.add(querySpecification);

/*
 * Call queryPerf()
 *
 * QueryPerf() returns the statistics specified by the provided
 * PerfQuerySpec objects. When specified statistics are unavailable -
 * for example, when the counter doesn't exist on the target
 * ManagedEntity - QueryPerf() returns null for that counter.
 */
List<PerfEntityMetricBase> retrievedStats = apiMethods.queryPerf(performanceMgrRef, pqsList);

```

Performance Data Returned by a vSphere Server

The query methods return sampling information and performance data. The sampling information indicates the collection interval in seconds and the time that the data was collected. When you call performance query methods, you pass in query specifications (`PerfQuerySpec`) to identify the performance data to be retrieved. To indicate the format of the output data, specify either "normal" or "csv" for the `PerfQuerySpec.format` property.

The query methods return `PerfEntityMetricBase` objects which you must cast into the appropriate type that corresponds to the `PerfQuerySpec.format` value specified in the call to the method.

- The `QueryPerf` method returns a list of `PerfEntityMetricBase` objects.
- The `QueryPerfComposite` method returns a `PerfCompositeMetric` object, which contains `PerfEntityMetricBase` objects.

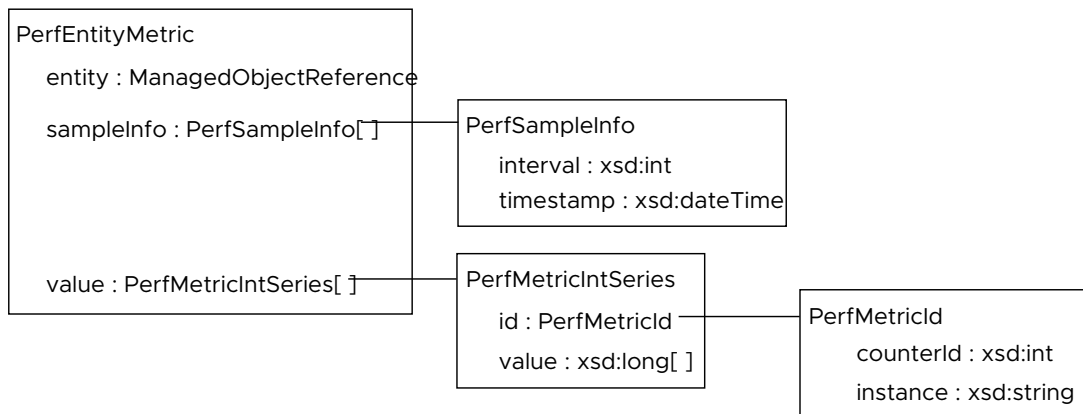
Normal Output Format

When you specify "normal" format, you must cast the returned `PerfEntityMetricBase` objects into `PerfEntityMetric` objects. Each `PerfEntityMetric` object contains the following properties:

- `entity` – Reference to the performance provider.
- `sampleInfo` – Array of sample information (`PerfSampleInfo` data objects), encoded as `xsd:int` and `xsd:dateTime` values.
- `value` – Array of data values (`PerfMetricIntSeries` data objects). Each object in the array contains the following properties:
 - `id` – Performance metric ID that identifies the counter instance.
 - `value` – Array of integers that corresponds to the array of sample information (`PerfEntityMetric.sampleInfo`).

The following figure shows a representation of the data object hierarchy returned by the query methods for normal format.

Figure 19-3. PerfEntityMetric Object Hierarchy



CSV Output Format

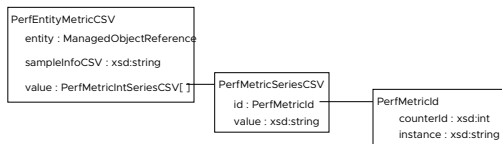
When you specify "csv" format, you must cast the returned `PerfEntityMetricBase` objects into `PerfEntityMetricCSV` objects. Both the sampling information and the collected data are encoded as comma-separated values suitable for display in tabular format.

The `PerfEntityMetricCSV` object contains the following properties:

- `entity` – Reference to the performance provider.
- `sampleInfoCSV` – String containing a set of interval and date-time values. The property contains string representations of `PerfSampleInfo xsd:int` and `xsd:dateTime` values. The string values are encoded in the following CSV format:
interval1, date1, interval2, date2
- `value` – Array of data values (`PerfMetricSeriesCSV` data objects). Each object in the array contains the following properties:
 - `id` – Performance metric ID that identifies the counter instance.
 - `value` – Set of sample values in CSV format, corresponding to the list of sample information (`PerfEntityMetricCSV.sampleInfoCSV`).

The following figure shows a representation of the data object hierarchy returned by the query methods for CSV format.

Figure 19-4. `PerfEntityMetricCSV` Object Hierarchy



Handling Returned Performance Data

The following code fragment prints out the returned performance data. This example uses CSV formatted data. The code fragment performs these tasks:

- Loop through the list of `PerfEntityMetricBase` objects returned by the `QueryPerf` method (`retrievedStats`).
 - Cast the `PerfEntityMetricBase` object to a `PerfEntityMetricCSV` object to handle the CSV output specified in the `PerfQuerySpec`.
 - Retrieve the sampled values.
 - Retrieve the interval information (`csvTimeInfoAboutStats`). The `sampleInfoCSV` string (`PerfEntityMetricCSV.sampleInfoCSV`) is `PerfSampleInfo` data formatted as `interval,time` pairs separated by commas – `interval-1,time-1,interval-2,time-2`. The list of pairs embedded in the string corresponds to the list of sampled values (`PerfEntityMetricCSV.value[]`).

- Print the time and interval information.
- Loop through the sampled values (`metricsValues`).
 - Use the counter metadata to print out identifying information about the counter along with the returned sampled value for the counter.
 - Use the `countersInfoMap` to translate the counter ID returned in the `PerfMetricSeriesCSV` object into the corresponding `PerfCounterInfo` object.

```

/*
 * Cycle through the PerfEntityMetricBase objects. Each object contains
 * a set of statistics for a single ManagedEntity.
 */
for(PerfEntityMetricBase singleEntityPerfStats : retrievedStats) {

    /*
     * Cast the base type (PerfEntityMetricBase) to the csv-specific sub-class.
     */
    PerfEntityMetricCSV entityStatsCsv = (PerfEntityMetricCSV)singleEntityPerfStats;

    /* Retrieve the list of sampled values. */
    List<PerfMetricSeriesCSV> metricsValues = entityStatsCsv.getValue();

    if(metricsValues.isEmpty()) {
        System.out.println("No stats retrieved. " +
            "Check whether the virtual machine is powered on.");
        throw new Exception();
    }

    /*
     * Retrieve time interval information (PerfEntityMetricCSV.sampleInfoCSV).
     */
    String csvTimeInfoAboutStats = entityStatsCsv.getSampleInfoCSV();

    /* Print the time and interval information. */
    System.out.println("Collection: interval (seconds),time (yyyy-mm-ddThh:mm:ssZ)");
    System.out.println(csvTimeInfoAboutStats);

    /*
     * Cycle through the PerfMetricSeriesCSV objects. Each object contains
     * statistics for a single counter on the ManagedEntity.
     */
    for(PerfMetricSeriesCSV csv : metricsValues) {
        /*
         * Use the counterId to obtain the associated PerfCounterInfo object
         */
        PerfCounterInfo pci = countersInfoMap.get(csv.getId().getCounterId());

        /* Print out the metadata for the counter. */
        System.out.println("-----");
        System.out.println(pci.getGroupInfo().getKey() + "."
            + pci.getNameInfo().getKey() + "."
            + pci.getRollupType() + " - "
            + pci.getUnitInfo().getKey());
    }
}

```

```

        System.out.println("Instance: "+csv.getId().getInstance());
        System.out.println("Values: " + csv.getValue());
    }
}

```

Large-Scale Performance Data Retrieval

The example described in the previous sections shows how to retrieve performance data for a single entity. When you design your application to retrieve performance data on a large scale, take the following information into consideration for more efficient processing.

- Use CSV formatted output. CSV format provides a more compact representation of the output data which can save on meta-data overhead.
- Create query specifications to reference a set of vSphere entities.
 - Using one `QueryPerf` method call per entity is not efficient.
 - Using a single call to `QueryPerf` to retrieve all of the performance data is not efficient.
 - As a general rule, specify between 10 and 50 entities in a single call to the `QueryPerf` method. This is a general recommendation because your system configuration may impose different constraints.
- Do not retrieve statistics more frequently than they are refreshed. For example, when you retrieve 20-second interval data, the data will not change until the next 20-second data collection event.
- Use `QueryAvailablePerfMetric` only when you intend to send a query for a specific counter using a specific performance interval. The method will return `PerfMetricId` objects that you can use for the query.

In all other cases, create the `PerfMetricId` objects for the query.

- For the `counterId` property, use the counter IDs from the `PerformanceManager` counter list (`PerformanceManager.perfCounter[].key`).
- For the `instance` property, specify an asterisk ("*") to retrieve instance and aggregate data or a zero-length string ("") to retrieve aggregate data only.

Using the QueryPerf Method as a Raw Data Feed

The `QueryPerf` method can operate as a raw data feed that bypasses the vCenter database and instead retrieves performance data from an ESXi host. You can use a raw data feed to obtain real-time instance data associated with 20-second interval collection and aggregate data associated with the 5-minute intervals.

You can use a raw data feed on vCenter Server 2.5 and later.

Performance Interval	Description
20-second	ESXi servers collect data for each performance counter every 20 seconds and maintain that data for an hour. When you specify a 20-second interval in the query specification for the <code>QueryPerf</code> method (<code>PerfQuerySpec.intervalId</code>), the method operates as a raw data feed. The Server ignores the historical interval collection levels and retrieves data for all of the requested counters from the ESXi servers. When you send a query for 20-second instance data, the server returns the most recent data collected for the 20-second interval. The server does not perform additional, unscheduled data collection to satisfy the query.
5-minute	<p>ESXi servers aggregate performance data according to the system-defined performance interval which specifies data collection every 300 seconds. To use a raw data feed for this data, specify the following <code>PerfQuerySpec</code> properties in the call to the <code>QueryPerf</code> method.</p> <ul style="list-style-type: none"> ■ <code>intervalId</code> – Specify 300 seconds to match the system-defined performance interval. ■ <code>startTime/endTime</code> – Specify time values within the last 30 minutes of the current time. The <code>QueryPerf</code> method checks the performance interval collection level on the vCenter Server. The method returns aggregated statistics for performance counters that specify a collection level (<code>PerfCounterInfo.level</code>) at or below the vCenter Server performance interval for the 300 second sampling period (<code>PerfInterval.level</code>). For example, if the vCenter Server performance interval is set to level one, and your query specification requests only performance counters that specify level four, the <code>QueryPerf</code> method will not return any data.

Comparison of Query Methods

The following table presents a comparison of performance query methods.

Method	Notes
QueryPerf	<ul style="list-style-type: none"> ■ Specify an array of <code>PerfQuerySpec</code> objects. ■ An unset <code>PerfQuerySpec.metricId</code> property produces results for all counters defined for <code>PerfQuerySpec.entity</code>. ■ <code>PerfQuerySpec.maxSample</code> is ignored for historical statistics. <p>You can use this method to retrieve historical statistics; you can also use it as a raw data feed. For information about retrieving the raw data collected on ESXi servers, see Using the QueryPerf Method as a Raw Data Feed.</p>
QueryPerfComposite	<ul style="list-style-type: none"> ■ Method works only at the host level. You can use a single call to the <code>QueryPerfComposite</code> method to retrieve performance data for a host and its virtual machines. ■ Specify a single <code>PerfQuerySpec</code> object. ■ You must specify a list of performance metrics to identify the data to be retrieved (<code>PerfQuerySpec.metricId</code>). ■ You cannot specify <code>PerfQuerySpec.maxSample</code>. <p>This method is designed for efficient client-server communications. <code>QueryPerfComposite</code> usually generates less network traffic than <code>QueryPerf</code> because it returns a large-grained object, a <code>PerfCompositeMetric</code> data object, that contains all the data.</p>

Retrieving Summary Performance Data

You can obtain near real-time summary information about performance or utilization without using the `PerformanceManager` methods. vSphere servers maintain "quick stats" data objects for hosts (`HostListSummaryQuickStats`), virtual machines (`VirtualMachineQuickStats`), and resource pools (`ResourcePoolQuickStats`). For more information about these objects, see the *vSphere API Reference*.

Performance Counter Metadata

Performance counters are organized by groups of system resources. Examples of performance counter groups are memory, CPU, and disk. The counter groups and specific counters used on any vSphere server depend on the server configuration. The *vSphere API Reference* contains a table for each counter group. The table includes the counter name, type of statistics being collected, unit of measurement, level, and so on. The *vSphere API Reference* page for the `PerformanceManager` managed object contains links to the tables.

PerfCounterInfo

The `PerformanceManager.perfCounter` property is an array of `PerfCounterInfo` data objects. Each object provides metadata for the collected data. A `PerfCounterInfo` object has a unique key, the counter ID. The actual performance data collected at runtime are identified by this counter ID. The following table lists the `PerfCounterInfo` properties.

Property	Description
<code>groupInfo</code>	Name of the resource group to which this counter belongs, such as disk, cpu, or memory.
<code>key</code>	Unique integer that identifies the counter. Also called the counter ID. The value is unique and it is not static—it might, for example, change between system reboots. The counter key on an ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESXi system. However, the system maps the keys from ESXi to vCenter Server systems automatically.
<code>level</code>	Number from 1 to 4 that identifies the level at which data values for this counter are aggregated.
<code>nameInfo</code>	Descriptive name for the counter. The name component of a fully qualified counter name, for example "granted" is the <code>nameInfo</code> property for the <code>mem.granted.AVERAGE</code> counter.
<code>rollupType</code>	Indicates how multiple samples of a counter are transformed into a single statistical value. Examples of rollup types are average, summation, and minimum. No conversion of values occurs for counters that specify absolute values, such as the total number of seconds that the system has been running continuously since startup. The <code>PerfSummaryType</code> is an enumeration containing valid constants for this property.
<code>statsType</code>	Type of statistical data that the value represents over the course of the interval, such as an average, a rate, the minimum value, and so on. The <code>PerfStatsType</code> is an enumeration containing valid constants for this property.
<code>unitInfo</code>	Unit of measure, such as megahertz, kilobytes, kilobytes per second, and so on. The <code>ElementDescription</code> 's <code>key</code> property is populated using one of the constants available in the <code>PerformanceManagerUnit</code> enumeration.

Performance Intervals

The `PerformanceManager` defines performance intervals which specify the period of time between collection events, how much data will be collected, and how long the collected data will be saved.

- An ESXi server has a built-in performance interval that produces discrete data values from counter instances sampled every 20 seconds. The server will maintain this instance data for one hour.
- Additional data collection is specified by historical performance intervals which produce data aggregated from counter instances according to the individual intervals.

The `PerformanceManager.historicalInterval` property is an array of `PerfInterval` objects. The following table lists the `PerfInterval` properties.

Property	Description
samplingPeriod	Number of seconds for the interval. You can modify this property on a vCenter Server only.
length	Period of time for which the server will save the data that it collects. You can modify this property on a vCenter Server only.
level	Level at which the Server collects data. The interval level corresponds to the performance counter level (<code>PerfCounterInfo.level</code>). The Server will collect data for all counters with levels that match <code>PerfInterval.level</code> , and for all counters with levels lower than <code>PerfInterval.level</code> . You can modify this property on a vCenter Server only.
enable	Enable/disable performance data collection. You can modify this property on a vCenter Server only.
key	Unique identifier for the interval. You cannot modify this property.
name	<p>Label for the historical interval; one of the following strings:</p> <ul style="list-style-type: none"> ■ "Past Day" ■ "Past Week" ■ "Past Month" ■ "Past Year" <p>The <code>PerformanceManager</code> uses the <code>samplingPeriod</code>, <code>level</code>, and <code>length</code> properties to determine its collection behavior. It does not interpret the name string. You cannot modify this property.</p>

ESXi Server Performance Intervals

An ESXi server collects performance data for each performance counter every 20 seconds. The `PerformanceManager.historicalInterval` array for an ESXi Server contains a single, readonly `PerfInterval` object that specifies rollup data collection every 5 minutes. You cannot retrieve 5-minute rollup data from an ESXi Server directly. You can use a vCenter Server connection to obtain 5-minute rollup data for an ESXi Server. The following table shows the historical interval property values on an ESXi server. You cannot modify this performance interval.

Property	Value	Description
key	1	Numeric identifier for the <code>PerfInterval</code> .
name	PastDay	Name of the <code>PerfInterval</code> .
samplingPeriod	300	Time interval between data sampling events.
length	129600	Number of seconds that statistics associated with the interval are kept by the vCenter Server.
enabled	true	This <code>PerfInterval</code> is enabled on the system.
level	null	Statistics collection level. For an ESXi system, this property is null. The <code>PerfInterval</code> object on an ESXi system defines the baseline interval.

vCenter Server Performance Intervals

A vCenter Server system aggregates performance data from all ESXi systems that it manages. The amount of data aggregated depends on the level setting configured for the vCenter Server. The level settings are reflected in the `PerformanceManager.historicalInterval` property for the vCenter Server system. `historicalInterval` is an array of `PerfInterval` data objects that define four different level settings, 1 through 4.

The following table lists the default values for the performance intervals on a vCenter Server system.

Key	Name	Sampling Period	Length	Enabled	Level
1	Past Day	300	86400	TRUE	1
2	Past Week	1800	604800	TRUE	1
3	Past Month	7200	2592000	TRUE	1
4	Past Year	86400	31536000	TRUE	1

By default, the collection level is set to 1 for each of the four intervals. Using the default level, a vCenter Server will collect data for all performance counters that specify collection level 1. Using the default length value, a vCenter Server will save collection data for the following time periods:

- 5-minute samples for the past day
- 30-minute samples for the past week
- 2-hour samples for the past month
- 1-day samples for the past year

Data older than a year is purged from the vCenter Server database.

vSphere Performance and Data Storage

The following sections provide information about modifying the operation of the `PerformanceManager` and vSphere Server performance data collection and storage.

- [Modifying Historical Intervals](#)
- [Modifying Performance Counter Collection Levels](#)

Modifying Historical Intervals

Changes to a vCenter performance interval are global and apply to all entities in the system. VMware recommends that you do not modify the historical intervals. The `PerfInterval` data objects in the `PerformanceManager.historicalInterval` array are related. Modifications to a performance interval affects the entire system and may cause problems.

If you must modify a performance interval, use the `PerformanceManager.UpdatePerfInterval` method and follow these guidelines.

- Performance data retention time (`PerfInterval.length`) must be a multiple of the collection interval (`PerfInterval.samplingPeriod`).
- Performance data retention length must increase in each interval compared to its predecessor. The `PerfInterval.length` value for each successive performance interval must be greater than the length property for the previous interval in the historical interval array.
- You cannot modify the value of the `PerfInterval.samplingPeriod` property on ESXi systems.

Modifying Performance Counter Collection Levels

The `PerformanceManager` provides the `UpdateCounterLevelMapping` method to change the collection level for individual performance counters (`PerfCounterInfo.level`). Consider carefully the performance and storage consequences of using the `UpdateCounterLevelMapping` method. If you use this method, you may cause a significant increase in data collection and storage, along with a corresponding decrease in performance. vCenter Server performance and database storage requirements depend on the collection levels defined for the performance intervals (`PerformanceManager.historicalInterval`) and the collection levels specified for individual performance counters (`PerfCounterInfo.level`).

Performance Counter Data Collection

vSphere defines four levels of data collection for performance counters. Each performance counter specifies a level for collection. The historical performance intervals (`PerformanceManager.historicalInterval`) define the sampling period and length for a particular collection level.

The amount of data collected for a performance counter depends on the performance interval and on the type of entity for which the counter is defined. For example, a datastore counter such as `datastoreIops` (the aggregate number of IO operations on the datastore) will generate a data set that corresponds to the number of datastores on a host. If a vCenter Server manages a large number of hosts with a large number of datastores, the Server will collect a large amount of data.

There are other counters for which the vCenter Server collects a relatively smaller amount of data. For example, memory counters are collected as a single counter per virtual machine and a single counter per host.

Performance Counter Data Storage

The performance interval collection level (`PerfInterval.level`) defines the set of counters for which the vCenter Server stores performance data. The Server will store data for counters at the specified level and for counters at all lower levels.

By default, all the performance intervals specify collection level one. Using these defaults, the vCenter Server stores performance counter data in the vCenter database for all counters that specify collection level one. It does not store data for counters that specify collection levels two through four.

Performance Manager Method Interaction

You can use the `UpdateCounterLevelMapping` method to change the collection level for individual counters. You can also use the `UpdatePerfLevel` method to change the collection level for the system-defined performance intervals. These methods can cause a significant increase in the amount of data collected and stored in the vCenter database.

- By default the system-defined performance intervals use collection level one, storing data for all counters that specify collection level one. If you use the `UpdateCounterLevelMapping` method to change the collection level of performance counters to level one, you will increase the amount of stored performance data.
- If you use the `UpdatePerfLevel` method to increase the collection level for the system-defined performance intervals, you will increase the amount of stored performance data.

To restore counter levels to default settings use the `ResetCounterLevelMapping` method.

vSphere Client Management of Performance Statistics

The vSphere Client displays the Performance Manager historical interval collection levels in the vCenter management statistics display. The vSphere Client also displays an estimate of the amount of storage that is required for data collection at the displayed levels. If individual counter levels are modified through the vSphere API (the `UpdateCounterLevelMapping` method), the vSphere Client will show a modified estimate. However, the vSphere Client cannot detect that the method has been called and it cannot display the current levels for individual counters. If you see a significantly increased estimate for storage, be aware that someone may have used the vSphere API to modify data collection.

Sample Code Reference

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java (SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\performance)
Basics.java
History.java
PrintCounters.java
RealTime.java

Java

(SDK\vsphere-ws\java\JAX-
WS\samples\com\vmware\performance)

VITop.java

VIUsage.java

Diagnostics and Troubleshooting

20

vSphere includes several logs, which you can access and customize. You can also use the `DiagnosticManager` service interface for troubleshooting.

This chapter includes the following topics:

- [Troubleshooting Best Practices](#)
- [Overview of Configuration Files and Log Files](#)
- [Modifying the Log Level to Obtain Detailed Information](#)
- [Using DiagnosticManager](#)
- [Using the MOB to Explore the DiagnosticManager](#)
- [Generating Diagnostic Bundles](#)

Troubleshooting Best Practices

Approach troubleshooting and problem-solving systematically, and take notes so you can trace your steps. Follow these guidelines to resolve issues with your client application.

- Do not change more than one thing at a time, and document each change and its result. Try to isolate the problem: Does it seem to be local, to the client? An error message generated from the server? A network problem between client and server?
- Use the logging facilities for your programming language to capture runtime information for the client application.
- Use the following VMware tools for analysis and to facilitate debugging.
 - vSphere Web Services API. The `DiagnosticManager` service interface allows you to obtain information from the server log files, and to create a diagnostic bundle that contains all system log files and all server configuration information. The vSphere Client and the MOB provide graphical and Web based access to the `DiagnosticManager`. `PerformanceManager` supports exploration of bottlenecks. See [Chapter 19 vSphere Performance](#).
 - Managed Object Browser (MOB). The MOB provides direct access to live runtime server-side objects. You can use the MOB to explore the object hierarchy, obtain property values, and invoke methods. See [Chapter 21 Managed Object Browser](#).

- VMware vSphere Client GUI. The vSphere Client allows you to examine log files for ESXi, vCenter Server, and virtual machines, and to change log level settings. Use vSphere Client menu commands to create reports that summarize configuration information, performance, and other details, and to export diagnostic bundles. The vSphere Client maintains its own local log files.

Overview of Configuration Files and Log Files

ESXi and vCenter Server configuration files control the behavior of the system. Most configuration file settings are set during installation, but can be modified after installation. Log files capture messages generated by the kernel and different subsystems and services. ESXi and vCenter Server services maintain separate log files. The following table lists log files or reports, their locations and associated configuration files.

Description	Log Location	Filename or Names	Configuration File
ESXi service log	/var/log/vmware/	hostd.log [hostd-0.log, ...hostd-9.log]	See ConfigStore
vCenter Server agent log	/var/log/vmware/vpx/	vpqa.log	See ConfigStore
Virtual machine kernel core file	/root/	vmkernel-core.<date> vmkernel-log.<date>	See ConfigStore
syslogd log	/var/log/	messages [messages.1, ... messages.4]	See ConfigStore
Service console availability report	/var/log/	vmkernel [vmkernel.1, ... vmkernel.8]	See ConfigStore
VMkernel messages, alerts, and availability reports	/var/log/vmkernel		See ConfigStore
VMkernel warning	/var/log/	vmkwarning [vmkwarning.1 ... 4 for history]	See ConfigStore
Virtual machine log file	vmfs/volume/ <vm_name>	vmware.log	<vm_name>/ <vm_name>.vmx

For developers, the following files are most relevant:

- `hostd.log` – Host daemon log, see [ESXi Log File](#). Can be used as a SOAP monitor when set to trivia log level as in [Generating Logs](#).
- `vpqa.log` – Agent log file found on each managed ESXi system.

- `vmware.log` – Virtual machine log. See [Virtual Machine Log Files](#).

Note Many configuration settings can no longer be edited directly with a web browser. To modify configuration settings in ConfigStore, use `/bin/configstorecli` or `esxcli`.

In addition to viewing log files in real time you can also generate reports and complete diagnostic bundles. See [Generating Diagnostic Bundles](#).

ESXi Log File

The ESXi log (`hostd.log`) captures information of varying specificity and detail, depending on the log level. Each request to the server is logged.

You can view the file using the vSphere Client. The raw text form of an ESXi (`hostd`) log file is shown in [Sample ESXi Log \(hostd.log\) Data](#).

Example: Sample ESXi Log (hostd.log) Data

```
...
[2008-05-07 09:50:04.857 'SOAP' 2260 trivial] Received soap response from
[TCP:myservername.vmware.com:443]: GetInterfaceVersion
[2008-05-07 09:50:04.857 'ClientConnection' 2260 info] UFAD interface version is vmware-
converter-4.0.0
[2008-05-07 09:50:04.857 'SOAP' 2260 trivial] Sending soap request to
[TCP:myservername.eng.vmware.com:443]: logout
[2008-05-07 09:50:04.857 'ProxySvc Req00588' 3136 trivial] Client HTTP stream read error
[2008-05-07 09:50:04.872 'ProxySvc Req00612' 3136 trivial] Request header:
POST /vmc/sdk HTTP/1.1
User-Agent: VMware-client
Content-Length: 435
Content-Type: text/xml; charset=utf-8
Cookie: vmware_soap_session="F127B435-56C7-4580-BAC4-3034DA1E67B6"; $Path=/
Host: myservername.vmware.com

[2008-05-07 09:50:04.872 'ProxySvc Req00588' 3816 trivial] Closed
[2008-05-07 09:50:08.450 'App' 3560 verbose] [VpxdHeartbeat] Invalid heartbeat from
10.17.218.46
[2008-05-07 09:50:10.013 'App' 3560 verbose] [VpxdHeartbeat] Queuing 10.17.218.45:829
(host-55)
[2008-05-07 09:50:10.013 'App' 1928 verbose] [HeartbeatHandler] 50208862-2752-d94c-2a73-
fa2ec9e38ecc:829 (host-55)
```

Virtual Machine Log Files

Each running virtual machine has its own log file, `vmware.log`, stored on the VMFS volume. By default, the log file is rotated whenever the virtual machine is powered on, but file rotation is configurable.

- ESXi maintains six log files that rotate at each power-cycle (the default) or at a configured file size.

- ESXi can be configured to maintain a specific number of log files. When the limit is reached, the oldest file is deleted.
- VMware recommends a log file size of 500 KB.
- Messages that are generated by VMware Tools are logged separately.

Example: VMkernel Availability Report

```

Availability Report for <servername>
Feb 27, 2008 - May 7, 2008

Availability: 99.949%
    Total time: 69 days, 15 hours
    Uptime: 69 days, 14 hours
    Downtime: 51 minutes

Note: Downtime is any time the system isn't capable of running
Virtual Machines. This includes reboots, crashes, configuration and running linux

Downtime Analysis:
    0.1% (51 minutes) downtime caused by:
    13.1% (6 minutes) scheduled downtime
    86.9% (44 minutes) unscheduled downtime

Reasons for scheduled downtime:
    84.9% server rebooting (1 instance)
    9.4% VMkernel unloaded (1 instance)
    5.7% server booting (3 instances)

Reasons for unscheduled downtime:
100.0% unknown (powerfail / reset?) (1 instance)

Stats:
    Current uptime: 8 days, 11 hours
    Longest uptime: 61 days, 2 hours
    Shortest uptime: 38 minutes
    Average uptime: 23 days, 4 hours
    Longest downtime: 44 minutes
    Shortest downtime: 7 seconds
    Average downtime: 8 minutes
    Maximum VMs Sampled: 1
    Average VMs Sampled: 0.94

Server Information:   Number of CPUs: 4 logical 4 cores
                     2 packages, Intel(R) Xeon(R) CPU          5150 @ 2.66GHz
                     Installed Memory: 2096416 kB
                     Current Build: 78591
Report generated Wed May 7 04:02:04 PDT 2008

```

vCenter Server Log Files

vCenter Server log files are located by default in the Documents and Settings subdirectory of the Windows account used to install the software. For example:

C:\Documents and Settings\Administrator\Local Settings\Application Data\VMware\

Important VMware recommends creating a user account especially for vCenter Server installation.

By default, the log files are hidden files. See the procedure for your Windows operating system to make the files visible.

Modifying the Log Level to Obtain Detailed Information

The amount of information captured in the log files varies, depending on the level setting.

Log Level Setting	Description
None	Disables logging.
Error	Logging limited to error messages.
Warning	Error messages plus warning messages are logged.
Info	Default setting on ESXi and vCenter Server systems. Errors, warnings, plus informational messages about normal operations are logged. Acceptable for production environments.
Verbose	Can facilitate troubleshooting and debugging. Not recommended for production environments.
Trivia	Extended verbose logging. Provides complete detail, including content of all SOAP messages between client and server. Use for debugging and to facilitate client application development only. Not recommended for production environments.

For example, the `hostd` service running on ESXi systems has a default log level setting of `info`. The vCenter Server logs are controlled by settings through the vSphere Client.

Setting the Log Level on ESXi Systems

The default log level setting for the ESXi Host Agent is `info`. If you run into issues during development, you can set the log level to `verbose`, or to `trivia` to obtain SOAP message content to use in debugging.

To change the log level for `hostd` on an ESXi system, use the following steps.

Procedure

- 1 Connect to the ESXi system using the vSphere Client.
- 2 On the **Host > Manage** screen, select the **System** tab.
- 3 On the **System** tab screen, select **Advanced Settings**.
- 4 Scroll to find the `Config.HostAgent.log.level` setting.

By default this setting is `info`.

- 5 Select the `Config.HostAgent.log.level` setting and click **Edit option**.

A dialog box opens, where you can enter a new setting.

- 6 Use the drop-down menu to select a new setting, such as **Verbose**, and click **Save**.

The **Recent Tasks** pane updates to confirm that the change completed.

Results

After the service restarts, the new log level is in effect.

Generating Logs

If you are connected to ESXi by SSH, you can use the `tail` command to explicitly create a log file that captures detail about actions that follow. For example, you can use the vSphere Client to create a new virtual machine and then use the content from the log as a model for how to create your own code.

To start the logging process and capture content to a file, use the following steps.

Procedure

- 1 Navigate to the location of the `hostd.log` file:

```
cd /var/log/vmware
```

- 2 Run the `tail` command, passing a filename in which to capture output:

```
tail -f hostd.log > yourfilenamehere
```

- 3 Use the vSphere Client to perform whatever action you are having difficulty modeling in your own code. For example, create a new virtual machine and stop the `tail` process with Ctrl-C when the operation completes.

Results

The file contains the SOAP message content and other log messages sent and received by `hostd` during the execution.

Setting the Log Level on vCenter Server Systems

To change log-level settings on vCenter Server, you must use the vSphere Client.

To set logging level for vCenter Server using the VMware vSphere Client, use the following steps.

Procedure

- 1 Log in to the vSphere Client and connect to the vCenter Server instance.
- 2 Choose **Administration** and click **Server Settings > Logging Options**.
- 3 Choose **Trivia** from the pop-up menu and click **OK**.

Using DiagnosticManager

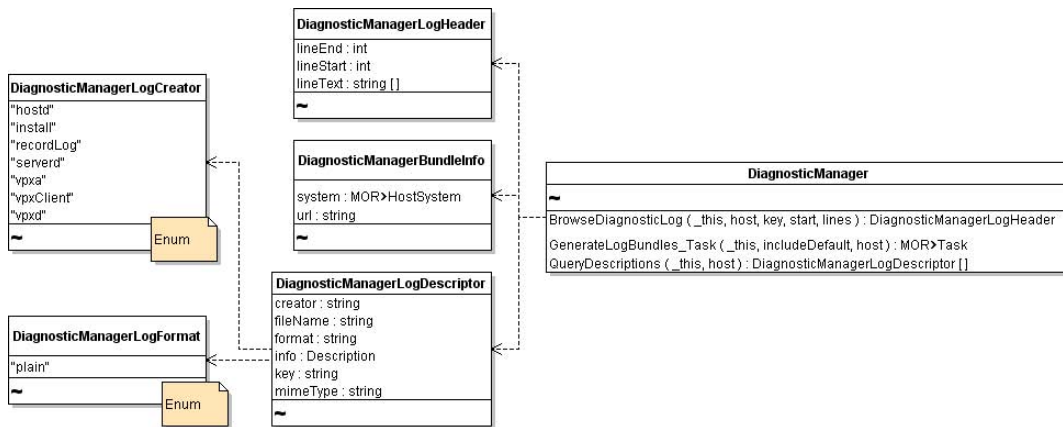
The vSphere API provides access to the `DiagnosticManager`, the service interface for obtaining information from the log files and for generating diagnostic bundles. The logs are populated based on configuration settings, such as `info`, `trivia`, and so on.

The `DiagnosticManager` is a managed object that works service-wide, rather than on a per-session basis. The `DiagnosticManager` has no properties, but provides operations for these tasks:

- Obtaining information about the logs and how they have been defined.
- Generating a diagnostic bundle that can be sent to VMware support for analysis.

Figure 20-1. [DiagnosticManager Managed Object and Associated Data Objects](#) shows a UML class diagram for `DiagnosticManager`, which is available on ESXi and vCenter Server systems.

Figure 20-1. DiagnosticManager Managed Object and Associated Data Objects



As shown in [Figure 20-1. DiagnosticManager Managed Object and Associated Data Objects](#), `DiagnosticManager` supports these methods:

- `BrowseDiagnosticLog`
- `GenerateLogBundleTask`
- `QueryDescriptions`

The `DiagnosticManagerLogDescriptor.creator` property contains the creator of the log, which is the system or subsystem that controls a specific log.

The `creator` value is populated from the `DiagnosticManagerLogCreator` enumeration.

The following table lists all string values currently available from the `DiagnosticManagerLogCreator` enumeration that can populate the `creator` property of the `DiagnosticManagerLogDescriptor` data object.

Name	Description
hostd	Host daemon
install	Installation
recordLog	System record log
serverd	Host server agent
vpva	vCenter agent
vpvClient	vSphere Client
vpvd	vCenter service

Creator	File Name	Format	Info.label	Info.summary	Key	Mime Type
hostd	/var/log/vmware/hostd.log	plain	ESX Log	ESX log in plain format	hostd	text/plain
hostd	/var/log/messages	plain	ESX Log	ESX log in plain format	messages	text/plain
hostd	/var/log/vmkernel	plain	ESX Log	ESX log in plain format	vmkernel	text/plain
hostd	/var/log/vmksummary.txt	plain	ESX Log	ESX log in plain format	vmksummary	text/plain
hostd	/var/log/vmkwarning	plain	ESX Log	ESX log in plain format	vmkwarning	text/plain
vpva	/var/log/vmware/vpx/vpxa.log	plain	vCenter Agent Log	vCenter agent log in plain format	vpva	text/plain

Using the MOB to Explore the DiagnosticManager

The Managed Object Browser (MOB) is a graphical interface that allows you to navigate the objects on a server and to invoke methods. You can access the `DiagnosticManager` using the MOB.

The following steps show how to explore `DiagnosticManager`.

Procedure

- 1 Start the mob by typing the MOB URL (`https://hostname.yourcompany.com/mob`) into a Web browser.
- 2 In the `ServiceContent` data object, click the link (`ha-diagnosticmanager` or `DiagMgr`) in the `Value` column for the `diagnosticManager` property, to navigate to the `DiagnosticManager` for the system.
 - For ESXi, `ha-diagnosticsmanager` is the managed object ID.

- For vCenter Server, `DiagMgr` is typically the managed object ID.

- 3 Click the link to the reference to display the managed object reference to the `DiagnosticManager` in the MOB.

`DiagnosticManager` provides three operations that allow you to obtain information about the descriptions currently available in the log file and log file content.

Because `DiagnosticManager` can track multiple ESXi systems, you can use the `QueryDescriptions` operation to return the names of keys used for all hosts. From this array, select the key for the host from which you want to obtain the log file.

- 4 On `QueryDescriptions`, click the **Invoke Method** link.

The vCenter Server system returns the contents of the log file for the selected host as a string array for the `lineText` property of `DiagnosticManagerLogHeader`.

Results

The string array returned through the MOB in this way is the content of the log file. The content contained in the log file is the same content that is available through the following other mechanisms:

- Displayed in the vSphere Client
- Included in a diagnostic bundle created through the `DiagnosticManager.GenerateLogBundles_Task` method.
- Available in the `hostd.log` file
- Returned to a client application that you write

What to do next

For more information about using the `DiagnosticManager`, see [Chapter 21 Managed Object Browser](#).

Generating Diagnostic Bundles

Typically, customers create diagnostic bundles at the request of VMware technical support. Diagnostic bundles also allow developers to quickly obtain all configuration files and log files in a complete package.

The generated compressed files are packaged in a file having the following pattern:

```
<fqdn-hostname>-esxsupport-yyyy-mm-dd@hh-mm-ss.tgz
```

Export Diagnostic Data By Using the vSphere Client

You can use the vSphere Client interface to generate diagnostic bundles.

Procedure

- 1 Connect to the ESXi system using the vSphere Client.
- 2 On the **Host > Monitor** screen, select the **Logs** tab.

The **Recent Tasks** pane updates to indicate that the system is generating a diagnostic bundle.

Results

When the task completes, the vSphere Client opens a dialog box that allows you to download the bundle.

The Managed Object Browser (MOB) is a graphical interface that allows you to navigate the objects on a server and to invoke methods. Any changes you make through the MOB take effect on the server.

This appendix explains how to use the MOB. The examples invoke `PerformanceManager` query methods to demonstrate how to pass primitive data types, arrays, and complex data types (data objects, including managed object references) using the MOB.

This chapter includes the following topics:

- [Using the MOB to Explore the Object Model](#)
- [Using the MOB to Invoke Methods](#)

Using the MOB to Explore the Object Model

The Managed Object Browser, or MOB, is a Web-based server application available for all ESXi and vCenter Server systems. The MOB lets you examine the objects that exist on the server and navigate through the hierarchy of live objects by clicking on links. The MOB populates the browser with actual runtime information, for example, the names of properties.

Caution Despite the word "browser" in its name, the MOB is not a read-only mechanism. The MOB allows you to make changes on the server by clicking the **InvokeMethod** link associated with methods.

Accessing the MOB

The MOB runs in a web browser and is accessed by using the fully-qualified domain name or IP address for the ESXi or vCenter Server system.

To access the MOB, use the following steps.

Procedure

- 1 Start a Web browser.
- 2 Enter the fully-qualified domain name (or the IP address) for the ESXi or vCenter Server system:

`https://hostname.yourcompany.com/mob`

3 Enter the user account and password for the system.

If warning messages regarding the SSL certificate appear, you can disregard them and continue to log in to the MOB, if VMware is the certificate authority and you are not in a production environment.

Results

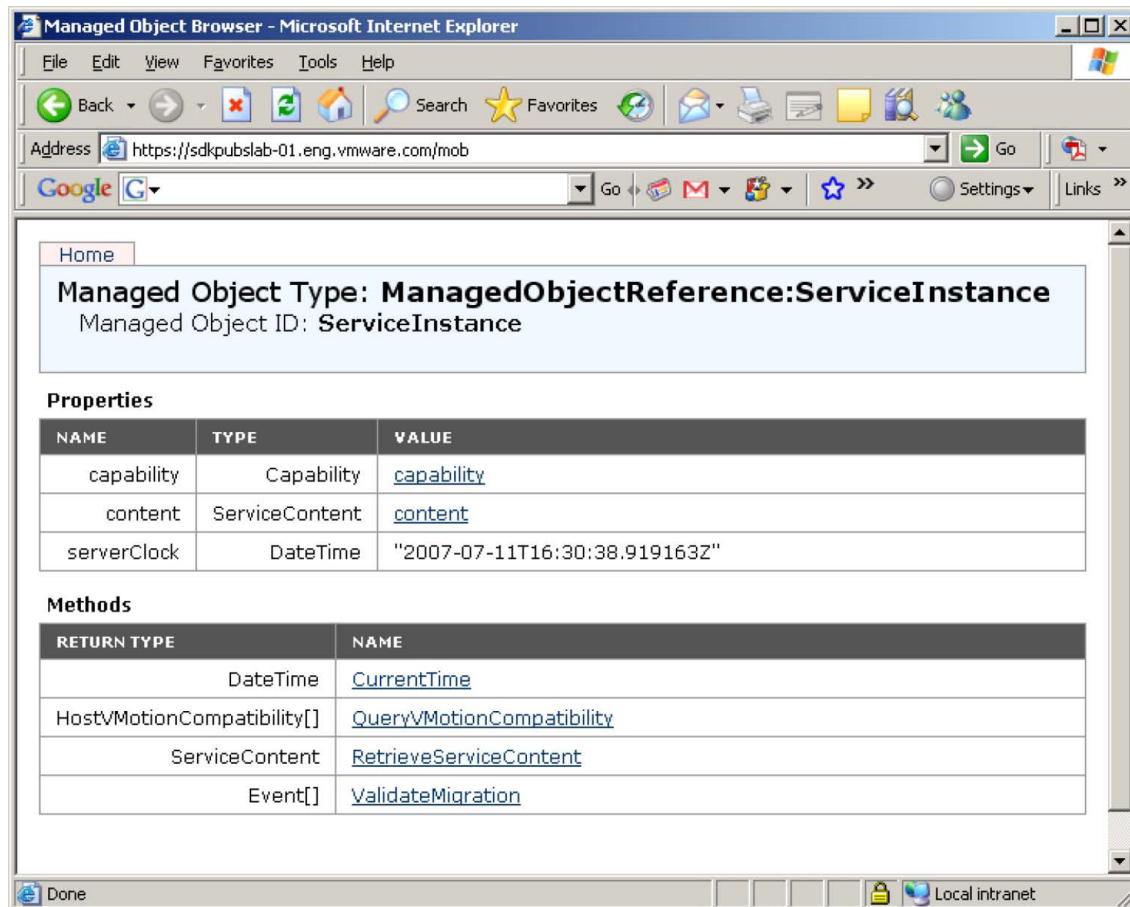
The MOB reveals the underlying structures of the object model. Seeing the structure in conjunction with the *API Reference Guide*, can help with understanding the model.

Using the MOB to Navigate the vSphere Object Model

Upon successful connection to the MOB, the browser displays the managed object reference for `ServiceInstance`. Client applications do not use managed objects directly, but interact with server-side managed objects by reference, using instances of the `ManagedObjectReference` data created for this purpose.

The page lists the properties and methods available through a `ServiceInstance` object. The `ServiceInstance` methods and properties provide access to the entire set of services and inventory objects available on the server. See [Chapter 5 Datacenter Inventory](#).

Figure 21-1. Managed Object Browser Connected to a vCenter Server



The MOB lets you examine the relationships among objects by looking at the properties and their values, and then drilling down into the objects. To explore the objects on the server, click the links in the **Value** column to navigate to the page that displays the object.

For example, to find out more about `ServiceContent`, click the **content** link to display the `ServiceContent` data object instance.

Using the MOB to Invoke Methods

You can use the MOB to invoke methods as follows:

Procedure

- 1 In the display of the object in which the method lives, click the name of the method.

A browser window displays information about the parameter name and type and allows you to specify parameter values.

- 2 Specify parameter values, using the method appropriate for the type, and click **Invoke Method**.

The rest of this section discusses how to pass different types of parameters to the MOB.

Passing Primitive Datatypes to Method

vSphere Web Services SDK data types are defined in the WSDL using XML Schema markup. The primitive data types are specified using the `xsd` namespace. For example, a string value for a property is defined as data type `xsd:string`. Enter a primitive value in the MOB as plain text, without quotation marks or other markup. For example, to enter an integer value of 10, type 10 in the field.

To obtain information about the available performance counters at level 4 on the server, enter a 4 in the `level` field of the `PerformanceManager.QueryPerfCounterByLevel` method. (This method is available only on the vCenter Server `PerformanceManager` API, not from an ESXi system.)

In response to the query, the array of `PerfCounterInfo` data objects and nested objects, with populated values from the server, displays in the Web browser.

Passing Arrays of Primitives to Methods

For an array, use the name of the parameter as the name of the property. For example, the `PerformanceManager.QueryPerfCounter` method requires an array of integers for the `counterId` parameter, as follows:

```
<counterId>58</counterId><counterId>65603</counterId><counterId>65604</counterId>
```

Even if you want to submit a single value for a single array element, you must wrap the parameter name around the value in this way.

Passing Complex Structures to Methods

For complex datatypes, enter the value as defined by the XML Schema in the WSDL. You can obtain the WSDL definition from the *vSphere API Reference* using the **Show WSDL type definition** links. Each data object type has an associated link.

Simple Content

The data object type `ManagedObjectReference` is one of the most commonly required parameters to be passed to the server. For example, the `MOB` for the `PerformanceManager.QueryPerfProviderSummary` method shows that the method requires a single parameter, the managed object reference (an instance of `ManagedObjectReference`) of the entity for which you want to obtain the `PerfProviderSummary` object.

Using the *vSphere API Reference* for `ManagedObjectReference` type, you can obtain the schema information from the **Show WSDL type definition** link at the bottom of the documentation page for `ManagedObjectReference`.

Example: XML Schema Definition of ManagedObjectReference Data Object

```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
  name="ManagedObjectReference">
  <simpleContent>
    <extension base="xsd:string">
      <attribute name="type" type="xsd:string"/>
    </extension>
  </simpleContent>
</complexType>
```

[XML Schema Definition of ManagedObjectReference Data Object](#) shows that a managed object reference is defined as a `<SimpleContent>` element that consists of a string that specifies the attribute `type` with its associated value, also as string. Use this information to construct the appropriate structure by replacing `type` with the parameter name from the `MOB`, setting the value as needed, and submitting in the entry field of the `MOB`. (The value for the `Datacenter` is displayed in the `MOB`.)

```
<entity type="Datacenter">datacenter-21</entity>
```

[Figure 21-2. Using the MOB to Pass Complex Types to a Method](#) shows the result of using the definition listed in [XML Schema Definition of ManagedObjectReference Data Object](#) to specify the managed object reference for a target datacenter to the `PerformanceManager.QueryPerfProviderSummary` method.

Figure 21-2. Using the MOB to Pass Complex Types to a Method

PerfProviderSummary QueryPerfProviderSummary

Parameters		
NAME	TYPE	VALUE
entity (required)	ManagedObjectReference:ManagedObject	<code><entity type="Datacenter">datacenter-21</entity></code>

As another example, one of the parameters required by the `VirtualMachine.CloneVM_Task` method is a `folder`. In this case, the parameter is defined as a managed object reference to a specific `Folder` object. Using the same definition shown in [XML Schema Definition of ManagedObjectReference Data Object](#), the result is as follows:

```
<folder type="Folder">folder-87</folder>
```

Although both examples submit a `ManagedObjectReference` to the MOB, each is specific to the parameter name required by the method (`entity` type for `PerformanceManager.QueryPerfProviderSummary` method, `folder` type for the `VirtualMachine.CloneVM_Task` method).

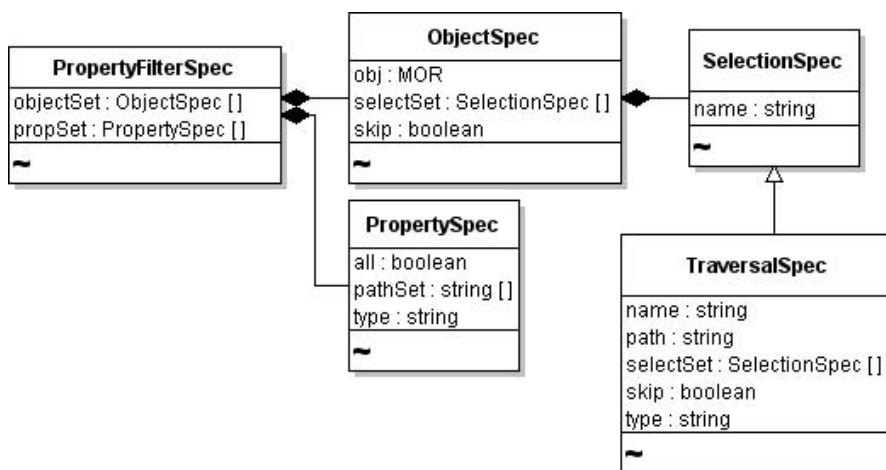
Complex Content

Many of the data objects required for method invocation consist of XML Schema elements defined as `<complexContent>` that can encompass many other elements.

For example, the `PropertyCollector.CreateFilter` method has a `spec` parameter that must be defined before method invocation. The `spec` parameter is defined as an instance of a `PropertyFilterSpec`.

[Figure 21-3. PropertyFilterSpec and Associated Data Objects](#) shows the relationships among several data objects that `PropertyFilterSpec` consists of.

Figure 21-3. PropertyFilterSpec and Associated Data Objects



To submit complex data structures such as this to the MOB, start by navigating the *vSphere API Reference*. Find the `PropertyFilterSpec` data object. Find the **Show WSDL type definition** link, and click it to display the XML Schema definition (see [XML Schema Definition of PropertyFilterSpec Data Object Type](#)).

[XML Schema Definition of PropertyFilterSpec Data Object Type](#) shows that the `PropertyFilterSpec` data object is a `<complexContent>` element that extends the `DynamicData` class with a sequence of two additional properties `propSet` (of type `PropertySpec`) and `objectSet` (of type `ObjectSpec`).

Example: XML Schema Definition of PropertyFilterSpec Data Object Type

```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
name="PropertyFilterSpec">
  <complexContent>
    <extension base="vim25:DynamicData">
      <sequence>
        <element name="propSet" type="vim25:PropertySpec" maxOccurs="unbounded"/>
        <element name="objectSet" type="vim25:ObjectSpec" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Because both elements are defined as a sequence, they must exist in the order listed. To obtain the definitions of `propSet` and `objectSet`, you must navigate further into the *vSphere API Reference*. [XML Schema Extract for PropertySpec](#) shows only the relevant parts of the XML Schema definition for `PropertySpec`. The `minOccurs="0"` attribute means that the element does not have to exist. The `maxOccurs="unbounded"` attribute means that the element can be populated as an array of any size. (When `minOccurs` is not set, but `maxOccurs` is set, the default for `minOccurs` defaults to 1, meaning one instance is required.)

Example: XML Schema Extract for PropertySpec

```
<sequence>
  <element name="type" type="xsd:string"/>
  <element name="all" type="xsd:boolean" minOccurs="0"/>
  <element name="pathSet" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
```

Navigate through the *vSphere API Reference* to the `ObjectSpec` definition. [ObjectSpec Definition as XML Schema](#) shows the excerpt.

Example: ObjectSpec Definition as XML Schema

```
...
<sequence>
  <element name="obj" type="vim25:ManagedObjectReference"/>
  <element name="skip" type="xsd:boolean" minOccurs="0"/>
  <element name="selectSet" type="vim25:SelectionSpec" minOccurs="0"
maxOccurs="unbounded"/>
```

```
</sequence>
...
```

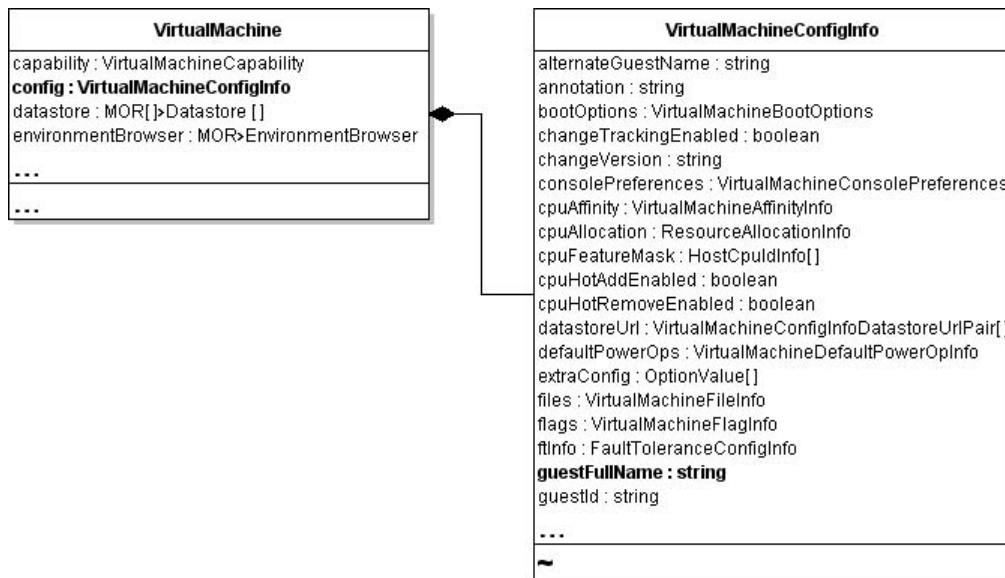
Extrapolating from the WSDL definitions shown in [XML Schema Definition of PropertyFilterSpec Data Object Type](#), [XML Schema Extract for PropertySpec](#), and [ObjectSpec Definition as XML Schema](#) might produce results similar to those shown in [CreateFilter Spec Property Entry](#).

Example: CreateFilter Spec Property Entry

```
<spec>
  <propSet>
    <type>VirtualMachine</type>
    <all>false</all>
    <pathSet>config.guestFullName</pathSet>
  </propSet>
  <objectSet>
    <obj type="Folder">group-v4</obj>
    <skip>true</skip>
  </objectSet>
</spec>
```

In this example, the `<spec>` element identifies the `spec` parameter of the `CreateFilter` method. The order of the element tags is as defined in the XML Schema for the property ([XML Schema Definition of PropertyFilterSpec Data Object Type](#)). The `pathSet` property defines the full path to the nested data object of interest. In [CreateFilter Spec Property Entry](#), the `pathSet` property defines the path to the `guestFullName` property of the target virtual machine. [ObjectSpec Definition as XML Schema](#) shows the UML of these nested data objects.

Figure 21-4. Nested Data Objects



All of these details are available in the *vSphere API Reference*. By examining the WSDL definition, you can construct the strings needed to submit parameters through the MOB.

Using the MOB Along With the API Reference

The following table provides a brief summary of the steps involved when you use the MOB and the *vSphere API Reference* together.

Datatype	How to Input Values for Methods
Primitive	Enter the value as plain text regardless of its data type (<code>int</code> , <code>string</code> , <code>boolean</code>). Do not use quotes or other markup.
Array	Use the name of the parameter as the name of the element, wrap the values in a series of opening and closing tags for each array element.
Complex	Obtain XML Schema format information from the <i>vSphere API Reference</i> for the type (from the Show WSDL type definition link). Use the schema definition to construct the sequence of tags around the value (or values) you want to pass to the MOB.

HTTP Access to vSphere Server Files

22

In most cases, client applications interact with vSphere servers by using the vSphere Web Services SDK. In some cases, direct access to configuration files, log files, and other data on an ESXi or vCenter Server systems is more efficient.

This chapter includes the following topics:

- [Introduction to HTTP Access](#)
- [URL Syntax for HTTP Access](#)

Introduction to HTTP Access

ESXi and vCenter Server systems support file access using HTTP and secure HTTP. You can use HTTP/HTTPS for the following kinds of access.

- Datastore access on ESXi and vCenter Server systems.
- ESXi configuration and log file access on ESXi systems.
- Update bundle access on ESXi systems.

You can use the HTTP methods `GET`, `HEAD`, `PUT`, and `DELETE` to access files. The URL of the HTTP/HTTPS request must contain an embedded keyword that specifies the type of access. The following table shows the server access types with the corresponding URL keyword and HTTP methods.

Server Access	URL Keyword	HTTP Method or Methods
Datastore	folder	GET, HEAD, PUT, DELETE
ESXi configuration file	host	GET, HEAD, PUT (See Host File Access (/host) for the specific methods supported for each file type.)
Update bundle	tmp	PUT

Use the `PUT` method to create new files or overwrite existing files. You can create a subdirectory by using a URL that is consistent with the supported top-level directories. You cannot create datastores or datacenters because the URL must refer to a valid datacenter or datastore.

You can use a Web browser to browse and download files. You cannot use a Web browser to post or delete files.

URL Syntax for HTTP Access

The URL specification in an HTTP request to a vSphere server includes one of the following keywords, which determines the type of access.

- [Datastore Access \(/folder\)](#)
- [Host File Access \(/host\)](#)
- [Update Package Access \(/tmp\)](#)

Datastore Access (/folder)

An HTTP request for datastore access uses the following syntax:

```
http-method http[s]://server/folder[/path]?dcPath=path[&dsName=name]
```

http-method	One of the methods GET, HEAD, PUT, or DELETE.
http:// or https://	Access protocol (standard access or secure access).
server	ESXi or vCenter Server target system. The server value can be an IP address or a DNS name.
/folder	Specifies datastore access on an ESXi or vCenter Server system. The datastore URL can include the following optional elements: <ul style="list-style-type: none"> ■ <i>path</i> – Path to a file or directory in the datastore, relative to the root of the datastore. ■ <i>dcPath</i> – Inventory path to a datacenter. Specify the datacenter path as a name-value pair in the request. For a datacenter named <i>Datacenter</i> located in the root folder, the <i>dcPath</i> value is <i>MyDatacenter</i>. For a datacenter named <i>YourDatacenter</i> located in the folder <i>NorthAmerica</i> which is located in the root folder, the <i>cdPath</i> value is <i>NorthAmerica/YourDatacenter</i>. ■ <i>dsName</i> – Datastore associated with the datacenter. Specify the datastore name as a name-value pair in the request.

The following examples illustrate the syntax. If the target server is an ESXi system, *dcPath=DCPATH&* is optional and defaults to *dcPath=ha-datacenter*.

Example	Description
/folder	Directory listing of known datacenters on this server.
/folder?dcPath=path	Directory listing of all datastores available at the specified datacenter.
/folder?dcPath=path&dsName=name	Top-level directory listing of the datastore.

Example	Description
<code>/folder/path?dcPath=path&dsName=name</code>	Directory listing of all files in a datastore directory.
<code>/folder/path/disk-flat.vmdk? dcPath=path&dsName=name</code>	Access individual files.

Host File Access (/host)

An HTTP request for access to ESXi configuration files uses the following syntax:

```
GET http[s]://my_system/host
http-method http[s]://my_system/host/file
```

Syntax Element	Description
http-method	One of GET, HEAD, or PUT, depending on the type of configuration file (see the following table).
http:// or https://	Access protocol (standard access or secure access).
esx-server	IP address or a DNS name.
/host	List of configuration files that you can access. (Use /host to retrieve the list.)
/host/file	A specific ESXi configuration file.

The following table shows ESXi host configuration files and the corresponding HTTP/HTTPS methods for access. The set of files might change from version to version.

Configuration File	HTTP Access Method(s)	Configuration File	HTTP Access Method(s)
esx.conf	GET, HEAD, PUT	ipmi0_sel.raw	GET, HEAD
openwsman.conf	GET, HEAD, PUT	ipmi0_sel	GET, HEAD
proxy.xml	GET, HEAD, PUT	ipmi0_sdr_content.raw	GET, HEAD
ssl_cert	GET, HEAD, PUT	ipmi0_sdr_header.raw	GET, HEAD
ssl_key	PUT	ipmi0_sensor_readings.raw	GET, HEAD
syslog.conf	GET, HEAD, PUT	ipmil_sel.raw	GET, HEAD
vmware_config	GET, HEAD, PUT	ipmil_sel	GET, HEAD
vmware_configrules	GET, HEAD, PUT	ipmil_sdr_content.raw	GET, HEAD
		ipmil_sdr_header.raw	GET, HEAD
		ipmil_sensor_readings.raw	GET, HEAD

Configuration File	HTTP Access Method(s)	Configuration File	HTTP Access Method(s)
		ipmi2_sel.raw	GET, HEAD
		ipmi2_sel	GET, HEAD
		ipmi2_sdr_content.raw	GET, HEAD
		ipmi2_sdr_header.raw	GET, HEAD
		ipmi2_sensor_readings.raw	GET, HEAD
		ipmi3_sel.raw	GET, HEAD
		ipmi3_sel	GET, HEAD
		ipmi3_sdr_content.raw	GET, HEAD
		ipmi3_sdr_header.raw	GET, HEAD
		ipmi3_sensor_readings.raw	GET, HEAD

Update Package Access (/tmp)

An HTTP request for update package access uses the following syntax:

```
PUT http[s]://esx-server/tmp/file-path
```

http:// or https://	Access protocol.
esx-server	IP address or a DNS name.
/tmp/file-path	Target file on an ESX/ESXi system.

Privilege Requirements for HTTP Access

HTTP access to a vSphere file is access to a datastore object that is associated with the folder structure in the vSphere inventory. HTTP access requires the same privileges needed to obtain these files using any other mechanism, such as the vSphere Client.

The following table shows the required privileges.

Object Associated with File	Portion of URL	Required Privileges
Root folder	/folder	System.View
Datacenter	?dcPath	Datastore.Browse Datastore.FileManagement
Datastore	&dsName	Datastore.Browse Datastore.FileManagement

Object Associated with File	Portion of URL	Required Privileges
Host	/host	Host.Config.AdvancedConfig
	/tmp/	Host.Config.SystemManagement

Sample Program Overview

23

The VMware vSphere Web Services SDK includes samples for the Java platforms. This chapter lists the available sample programs and provides some information about each program. The samples have been re-compiled with JAX-WS bindings, and they use JAX-WS credential store classes that allow you to ignore certificates when you connect to a server with the samples.

This chapter includes the following topics:

- [Java Sample Programs \(JAXWS Bindings\)](#)

Java Sample Programs (JAXWS Bindings)

When you download the SDK, you can find the Java sample programs and related files in the following directories.

- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware` – Top-level directory for Java samples. Details listed in the following table.
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\security` – Credential store utilities
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vim25` – Stub directory. The `vim25` directory contains stubs for release 2.5 and later.
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vm` – samples written for a single VM

Directory	Example	Description
<code>alarms</code>	<code>VMPowerStateAlarm</code>	Create an alarm to monitor a virtual machine's power state.
<code>cim</code>	<code>CIMReader</code>	Walks the Common Information Model (CIM) data associated with an ESXi host.
<code>connection</code>	<code>BasicConnection</code> (among others)	Shows how to set up a vSphere connection with user name and password.
<code>events</code>	<code>EventFormat</code>	Retrieve and format the last event from the host daemon or <code>vpzd</code> . Includes a function that formats the event message.
	<code>EventHistoryCollectorMonitor</code>	Demonstrates how to create and monitor an <code>EventHistoryCollector</code> . Uses the <code>latestPage</code> property of <code>EventHistoryCollector</code> to filter the events.

Directory	Example	Description
	<code>VMEventHistoryCollectorMonitor</code>	Standalone client that demonstrates how to perform the following tasks: (1) Logging into the web service. (2) Creating <code>EventHistoryCollector</code> filtered for a single virtual machine. (3) Monitoring events using the <code>latestPage</code> property of the <code>EventHistoryCollector</code> .
fcd	<code>FcdCreate, FcdDelete</code>	Create and delete a first class disk (FCD) virtual storage object.
	<code>FcdAttachToVM, FcdDetachFromVM</code>	Attach or detach FCD to or from VM.
	<code>FcdRegisterLegacyDisk</code>	Register legacy disk as a first class disk.
general	<code>Browser</code>	Print all managed entities and for each entity its type, reference value, property name, property value, inner object type, inner reference value and inner property value.
	<code>Connect</code>	Connect to an ESXi host or a vCenter Server system.
	<code>Create</code>	Create a managed entity such as a folder, datacenter, or cluster.
	<code>Delete</code>	Delete a managed entity from the inventory tree. The managed entity can be a virtual machine, a <code>ClusterComputeResource</code> , or a folder.
	<code>GetCurrentTime</code>	Retrieve the current time from the vSphere Server.
	<code>GetHostName</code>	Retrieve the hostname of the ESXi Server.
	<code>GetUpdates</code>	Demonstrates how to use the <code>PropertyCollector</code> to monitor one or more properties of one or more managed objects. In particular this sample monitors one or all virtual machines and all hosts or one host for changes to some basic properties.
	<code>LicenseManager</code>	Demonstrates uses of the licensing API using <code>License</code> managed object reference.
	<code>Move</code>	Move a managed entity from its current location in the inventory to a new location in a specified folder
	<code>PropertyCollector</code>	Illustrates the use of the <code>PropertyCollector</code> API.
	<code>RemoveManagedObject</code>	Destroy or unregister a managed inventory object like a <code>Host</code> , <code>VirtualMachine</code> , or <code>Folder</code> .
	<code>Rename</code>	Rename a managed entity object.
	<code>SearchIndex</code>	Illustrates the use of the <code>SearchIndex</code> API.
	<code>SimpleClient</code>	Connect to the server, log in, list the inventory contents (managed entities) at the console, and log out.

Directory	Example	Description
	TaskList	Display a list of tasks performed on a specified managed object.
guest	CreateTemporaryFile	Create a temporary file inside a virtual machine.
	DownloadGuestFile	Download a file from the guest to a specified path on the host where the client is running.
	RunProgram	Run a specified program inside a virtual machine. RunProgram re-directs output to a temporary file inside the guest and downloads the output file.
	UploadGuestFile	Upload a file from the client machine to a specified location inside the guest operating system.
hcl	HCIBatchAddHostAndExtendCluster	Call batch-add-host to add four or more new hosts to an HCI cluster.
	HCIComputeOnlyCluster	Call HCI configure to create a simple cluster configuration.
host	AcquireSessionInfo	Acquire session with a vCenter Server or ESXi host and print a CIM service ticket and related session information to a file.
	AddVirtualNic RemoveVirtualNic	Add a virtual NIC to a port group on a virtual switch. Remove a virtual NIC from a port group.
	AddVirtualSwitch RemoveVirtualSwitch	Add a virtual switch to a host. Remove a virtual switch from the host.
	AddVirtualSwitchPortGroup RemoveVirtualSwitchPortGroup	Add a port group to a virtual switch. Remove a port group from a virtual switch.
	DVSCreate	Create a distributed virtual switch.
	HostProfileManager	Demonstrates the use of HostProfileManager and ProfileComplianceManager.
	NIOCForDVS	Add a network resource pool to a distributed virtual switch.
httpfileaccess	GetVMFiles	Retrieve configuration files, snapshot files, log files, and virtual disk files of a virtual machine; place them on the system on which the program is run.
	PutVMFiles	Put virtual machine files into a specified datacenter and datastore; register and reconfigure the corresponding virtual machine.
performance	Basics	Display available performance counters or other metadata for an ESXi host.
	History	Read performance measurements from the current time, or from a specified start time, for a specified duration.

Directory	Example		Description
	PrintCounters		Write the available counters of a managed entity into the specified file at the specified location. The managed entity can be a host system, a virtual machine, or a resource pool.
	RealTime		Display performance measurements from the current time at the console.
	VITop		An ESXtop-like sample application that lets administrators specify the CPU and memory counters by name to obtain metrics for a specified host.
	VIUsage		Create a GUI for graphical representation of the counters.
scheduling	DeleteOneTimeScheduledTask		Demonstrates deleting a ScheduledTask.
	OneTimeScheduledTask		Demonstrates creating a ScheduledTask using the ScheduledTaskManager.
	WeeklyRecurrenceScheduledTask		Demonstrates creating a weekly recurrent scheduled task.
scsilun	SCSILunName		Display the CanonicalName, Vendor, Model, Data, Namespace and NamespaceId of the host's SCSI LUN.
security	credstore	Base64	A fast, memory efficient class that encodes and decodes to and from BASE64 in full accordance with RFC 2045.
		CredentialStore	Create an example credential store.
		CredentialStoreAdmin	A command-line tool that provides complete access to the credential store backing file on the local machine.
		CredentialStoreCipher	This class uses the JVM provided classes in javax.crypto to encrypt and decrypt text.
		CredentialStoreFactory	Factory class providing instances of a credential store.
		CredentialStoreImpl	Implementation class for CredentialStoreAdmin.
		CredentialStoreStorage	This class provides the same functionality as FileInputStream, except that the close() method is overridden so that FileInputStream class' close() method does not get called.
simpleagent	CreateUser		Create a user account and password and store them in the local credential store.
	SimpleAgent		Access the local credential store to obtain a single user account for login to a server.
storage	CreateStorageDRS		Creates storage DRS.

Directory	Example	Description
	<code>SDRSRecommendation</code>	Runs storage DRS on an SDRS cluster to obtain SDRS recommendations.
	<code>SDRSRules</code>	Configures rules for an SDRS cluster.
vApp	<code>OvfManagerExportVapp</code>	Demonstrates the <code>OvfManager</code> by exporting VMDKs and OVF Descriptors of all VMs in the vApps.
	<code>OvfManagerExportVMDK</code>	Demonstrates how the <code>OvfManager</code> exports VMDKs from a VM to the localSystem.
	<code>OvfManagerImportLocalVapp</code>	Use this class to import or deploy an OVF Appliance from a local drive.
	<code>OvfManagerImportVAppFromUrl</code>	Use this class to import or deploy an OVF Appliance from a specified URL.
vim25	This directory contains the many Java classes that define JAX-WS bindings to the vSphere API.	
vm	<code>VMApplyEvc</code>	Apply a per-VM extended vMotion compatibility (EVC) setting to an existing virtual machine.
	<code>VMClone</code>	Locate an existing virtual machine on the vCenter Server system, make a template from this virtual machine, and deploy instances of the template onto a datacenter.
	<code>VMCreate</code>	Create a virtual machine. Different command-line input creates the virtual machine in different ways.
	<code>VMDeltaDisk</code>	Create a delta disk on top of an existing virtual disk in a virtual machine, and simultaneously removes the original disk using the reconfigure API. Use delta disks in conjunction with linked virtual machines.
	<code>VMDiskCreate</code>	Create a virtual disk.
	<code>VMLinkedClone</code>	Create a linked virtual machine from an existing snapshot.
	<code>VMManageCD</code>	Configure a CDROM for a virtual machine. Also list information about the CDROMs associated with a virtual machine.
	<code>VMManageFloppy</code>	Configure a floppy drive for a virtual machine. Also list information about the floppy drives associated with a virtual machine.
	<code>VMotion</code>	Check whether migration with <code>VMotion</code> is feasible between two hosts. Perform a migration if the hosts are compatible.
	<code>VMpowerOps</code>	Perform power operations on a virtual machine.
	<code>VMPromoteDisks</code>	Consolidate a linked virtual machine by using the <code>VirtualMachine.PromoteDisks</code> method.
	<code>VMReconfig</code>	Reconfigure a virtual machine. Includes reconfiguring the disk size and disk mode.

Directory	Example	Description
	VMRelocate	Relocate a linked virtual machine using disk move type.
	VMSnapshot	Perform virtual machine snapshot operations
	XVCvMotion	Relocate virtual machine across from one vCenter Server to another.
	XVMotion	Relocate virtual machine to the computing resource recommended by the DRS.